aws

デベロッパーガイド

Amazon Keyspaces (Apache Cassandra 向け)

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Keyspaces (Apache Cassandra 向け): デベロッパーガイド

Copyright © 2025 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 Keyspaces とは ... 1
仕組み ... 2
高レベルのアーキテクチャ ... 2
Cassandra のデータモデル ... 5
Amazon Keyspaces へのアクセス ... 6
ユースケース ... 7
CQL とは ... 7
Amazon Keyspaces と Cassandra を比較する ... 9
Apache Cassandra との機能の違い ... 10
Apache Cassandra の API、オペレーション、およびデータ型 ... 11
キースペースとテーブルの非同期的な作成および削除 ... 11
認証と認可 ... 11
バッチ ... 11
変更データキャプチャ (CDC) ... 11
クラスターの設定 ... 12
Connections ... 12
IN キーワード ... 13
FROZEN collections ... 13
軽量トランザクション ... 14
負荷分散 ... 14
Pagination (ページ分割) ... 14
パーティショナー ... 14
プリペアドステートメント ... 15
範囲削除 ... 15
システムテーブル ... 15
タイムスタンプ ... 16
ユーザー定義タイプ (UDTs) ... 16
サポートされている Cassandra API、オペレーション、関数、データ型 ... 16
Cassandra API サポート ... 17
Cassandra コントロールプレーン API サポート ... 18
Cassandra データプレーン API サポート ... 19
Cassandra の関数サポート ... 19
Cassandra データ型サポート ... 20
サポートされている Cassandra の整合性レベル ... 22

書き込みの整合性レベル 22
読み取りの整合性レベル 23
サポートされていない整合性レベル 24
Amazon Keyspaces への移行 25
Cassandra からの移行 26
互換性 27
料金の推定 28
移行戦略 37
オンライン移行 38
オフライン移行 50
ハイブリッド移行 52
移行ツール 56
cqlsh を使用したデータのロード 58
DSBulk を使用したデータのロード 71
Amazon Keyspaces へのアクセス 84
セットアップ AWS Identity and Access Management 84
にサインアップする AWS アカウント 84
管理アクセスを持つユーザーを作成する 84
Amazon Keyspaces を設定する 86
コンソールを使用する場合 87
の使用 AWS CloudShell 87
の IAM アクセス許可の取得 AWS CloudShell 88
を使用した Amazon Keyspaces の操作 AWS CloudShell 89
プログラムによるアクセスのための認証情報の作成 90
サービス固有の認証情報の作成 91
AWS 認証用の IAM 認証情報を作成する 94
サービスエンドポイント 102
ポートとプロトコル 103
グローバルエンドポイント 103
AWS GovCloud (US) Region FIPS エンドポイント 106
中国リージョンのエンドポイント 107
cqlsh の使用 107
cqlsh-expansion の使用 108
TLS の cqlsh 接続を手動で設定する方法 113
の使用 AWS CLI 114
AWS CLIのダウンロードと設定 114

Amazon Keyspaces AWS CLI での の使用 ... 115
API を使用する ... 119
Cassandra クライアントドライバーの使用 ... 119
Cassandra Java クライアントドライバーの使用 ... 120
Cassandra Python クライアントドライバーの使用 ... 133
Cassandra Node.js クライアントドライバーの使用 ... 137
Cassandra .NET Core クライアントドライバーの使用 ... 141
Cassandra Go クライアントドライバーの使用 ... 143
Cassandra Perl クライアントドライバーの使用 ... 148
クロスアカウントアクセスの設定 ... 150
共有 VPC のクロスアカウントアクセスの設定 ... 150
共有 VPC を利用しないクロスアカウントアクセスの設定 ... 155
入門 ... 160
前提条件 ... 161
キースペースの作成 ... 161
キースペースの作成ステータスの確認 ... 165
テーブルの作成 ... 166
テーブルの作成ステータスの確認 ... 174
CRUD オペレーション ... 175
作成 ... 176
読み取り ... 179
更新 ... 183
削除 ... 185
テーブルの削除 ... 187
キースペースの削除 ... 189
チュートリアルとソリューション ... 193
VPC エンドポイントによる接続 ... 193
前提条件 ... 194
ステップ 1: Amazon EC2 インスタンスを起動する ... 194
ステップ 2: Amazon EC2 インスタンスを設定する ... 196
ステップ 3: Amazon Keyspaces 用の VPC エンドポイントを作成する ... 199
ステップ 4: VPC エンドポイント接続の権限を設定する ... 204
ステップ 5: モニタリングを設定する ... 208
ステップ 6: (オプション) 接続に関するベストプラクティス ... 209
ステップ 7: (オプション) クリーンアップする ... 211
Apache Spark との統合 ... 213

前提条件 ... 214
ステップ 1: Amazon Keyspaces を設定する ... 214
ステップ 2: Apache Cassandra Spark コネクタを設定する ... 216
ステップ 3: アプリ設定ファイルを作成する ... 218
ステップ 4: ソースデータとターゲットテーブルを準備する ... 220
ステップ 5: Amazon Keyspaces データを書き込み、読み取る ... 222
トラブルシューティング ... 225
Amazon EKS からの接続 ... 227
前提条件 ... 227
ステップ 1: Amazon EKS クラスターの設定 ... 230
ステップ 2: アプリケーションを設定する ... 235
ステップ 3: アプリケーションイメージの作成 ... 238
ステップ 4: アプリケーションの Amazon EKS へのデプロイ ... 239
ステップ 5: (オプション) クリーンアップする ... 245
Amazon S3 へのデータのエクスポート。 ... 247
前提条件 ... 248
ステップ 1: Amazon S3 バケットの作成、ツールのダウンロード、環境の設定 ... 249
ステップ 2: AWS Glue ジョブを設定する ... 252
ステップ 3: からエクスポート AWS Glue ジョブを実行する AWS CLI ... 255
ステップ 4: (オプション) エクスポートジョブをスケジュールする ... 256
ステップ 5: (オプション) クリーンアップする ... 258
サーバーレスリソースの管理 ... 260
行サイズの推定 ... 261
エンコードされた列のサイズを推定する ... 262
データ型に基づいてデータ値のエンコードされたサイズを推定する ... 263
Amazon Keyspaces 機能が行サイズに与える影響を考慮する ... 264
行のエンコードされたサイズを計算する適切な式を選択する ... 265
行サイズ計算の例 ... 266
キャパシティ消費量の推定 ... 267
範囲クエリのキャパシティ消費量の推定 ... 268
限定クエリの読み取りキャパシティ消費量を推定する ... 269
テーブルスキャンの読み取りキャパシティ消費量を推定する ... 270
LWT のキャパシティ消費量の推定 ... 271
静的列のキャパシティ消費量の推定 ... 272
マルチリージョンテーブルのキャパシティの推定 ... 276
CloudWatch によるキャパシティ消費量の推定 ... 277

読み取り/書き込みキャパシティモードの設定 ... 278
オンデマンドキャパシティモードを設定する ... 278
プロビジョンドスループットキャパシティモードを設定する ... 281
テーブルのキャパシティモードの表示 ... 283
キャパシティモードを変更する ... 284
オンデマンドキャパシティに備えた新規テーブルの事前ウォーミング ... 287
オンデマンドキャパシティに備えた既存のテーブルの事前ウォーミング ... 290
自動スケーリングによるスループットキャパシティの管理 ... 293
Amazon Keyspaces オートスケーリングの仕組み ... 294
マルチリージョンテーブルに対する自動スケーリングの仕組み ... 296
使用に関する注意事項 ... 296
自動スケーリングポリシーの設定と更新 ... 297
バーストキャパシティの使用 ... 311
Amazon Keyspaces の機能の使用 ... 313
システムキースペース ... 314
system ... 315
system_schema ... 316
system_schema_mcs ... 317
system_multiregion_info ... 320
ユーザー定義タイプ (UDTs) ... 323
アクセス許可の設定 ... 324
UDT を作成する ... 327
UDTsを表示する ... 331
UDT を削除する ... 334
CQL クエリの使用 ... 336
IN SELECT の使用 ... 336
結果の並べ替え ... 340
結果のページ分割 ... 341
CDC ストリームの使用 ... 342
CDC ストリームの仕組み ... 343
CDC ストリームを使用する ... 347
パーティショナーの操作 ... 380
パーティショナーの変更 ... 381
クライアントサイドのタイムスタンプ ... 383
AWS サービスとの統合 ... 384
クライアント側のタイムスタンプが有効なテーブルの作成 ... 385

クライアント側のタイムスタンプの設定 388
クエリでのクライアント側のタイムスタンプの使用 391
マルチリージョンレプリケーション 392
利点 392
キャパシティモードと料金 393
仕組み 394
使用に関する注意事項 399
マルチリージョンレプリケーションの設定 401
ポイントインタイムリカバリによるバックアップと復元 430
仕組み 431
ポイントインタイムリカバリの使用 436
TTL によるデータの期限切れ 449
AWS サービスとの統合 451
デフォルト TTL 値を設定したテーブルの作成 451
テーブルのデフォルト TTL 値の更新 455
カスタム TTL のテーブルの作成 459
テーブルのカスタム TTL の更新 461
INSERT を使用した新しい行のカスタム TTL の設定 463
UPDATE を使用した行と列のカスタム TTL 値の設定 464
AWS SDKs の使用 465
タグの操作 466
タグ指定の制限 467
キースペース、テーブル、ストリームにタグを付ける 468
コスト配分レポートの作成 487
AWS CloudFormation リソースの作成 488
Amazon Keyspaces と AWS CloudFormation テンプレート 488
の詳細 AWS CloudFormation 488
NoSQL Workbench 489
ダウンロード 490
入門 490
データモデルの可視化 492
データモデルの作成 496
データモデルの編集 499
データモデルのコミット 501
サンプルデータモデル 511
リリース履歴 512

コードの例 514
基本 519
Hello Amazon Keyspaces 520
基本を学ぶ 525
アクション 587
ツールとライブラリ 633
ライブラリと例 633
Amazon Keyspaces (Apache Cassandra 向け) のデベロッパーツールキット 633
Amazon Keyspaces (Apache Cassandra 向け) の事例 633
AWS 署名バージョン 4 (SigV4) 認証プラグイン 634
ハイライトされたサンプルおよびデベロッパーツールのリポジトリ 634
Amazon Keyspaces のプロトコルバッファ 634
AWS CloudFormation Amazon Keyspaces (Apache Cassandra 向け) メトリクス用の Amazon CloudWatch ダッシュボードを作成するための テンプレート 634
での Amazon Keyspaces (Apache Cassandra 向け) の使用 AWS Lambda 635
Spring と Amazon Keyspaces (Apache Cassandra 向け) の使用 635
Scala と Amazon Keyspaces (Apache Cassandra 向け) の使用 635
での Amazon Keyspaces (Apache Cassandra 向け) の使用 AWS Glue 635
Amazon Keyspaces (Apache Cassandra 向け) Cassandra クエリ言語 (CQL) から AWS CloudFormation コンバーターへ 636
Java 向け Apache Cassandra ドライバーの Amazon Keyspaces (Apache Cassandra 向け) ヘルパー 636
Amazon Keyspaces (Apache Cassandra 向け) 圧縮の簡単なデモ 636
Amazon Keyspaces (Apache Cassandra 向け) と Amazon S3 Codec のデモ 636
ベストプラクティス 637
NoSQL 設計 638
NoSQL と RDBMS の比較 639
2 つの重要な概念 639
一般的なアプローチ 640
Connections 641
働き 641
接続を設定する方法 642
再試行ポリシーを設定する方法 644
VPC エンドポイント接続 644
接続をモニタリングする方法 646
接続エラーの処理方法 647

データモデリング 647
パーティションキーの設計 648
コスト最適化 650
テーブルレベルでコストを評価する 651
テーブルのキャパシティモードの評価 653
テーブルの Application Auto Scaling 設定を評価する 657
未使用のリソースを特定する 665
テーブルの使用パターンを評価する 670
適切なサイズのプロビジョニングを行うために、プロビジョンドキャパシティを評価する 671
トラブルシューティング 681
一般的なエラー 682
一般的なエラー 682
接続エラー 683
Amazon Keyspaces エンドポイントへの接続中のエラー 684
キャパシティ管理エラー 696
サーバーレス容量エラー 696
データ定義言語エラー 702
データ定義言語エラー 702
Amazon Keyspaces のモニタリング 707
CloudWatch によるモニタリング 708
メトリクスの使用 709
メトリクスとディメンション 710
メトリクスの表示 734
アラームの作成 735
CloudTrail によるログ記録 735
CloudTrail でのログファイルエントリの設定 736
CloudTrail での DDL 情報 737
CloudTrail での DML 情報 738
ログファイルエントリについて 739
セキュリティ 751
データ保護 752
保管データ暗号化 753
転送中の暗号化 774
インターネットトラフィックのプライバシー 775
AWS Identity and Access Management 776

対象者 ... 777
アイデンティティを使用した認証 ... 778
ポリシーを使用したアクセスの管理 ... 781
Amazon Keyspaces で IAM が機能する仕組み ... 784
アイデンティティベースのポリシーの例 ... 790
AWS マネージドポリシー ... 798
トラブルシューティング ... 810
サービスにリンクされたロールの使用 ... 813
コンプライアンス検証 ... 823
耐障害性 ... 825
インフラストラクチャセキュリティ ... 826
インターフェイス VPC エンドポイントの使用 ... 827
ストリームでのインターフェイス VPC エンドポイントの使用 ... 834
Amazon Keyspaces の設定と脆弱性の分析 ... 838
セキュリティに関するベストプラクティス ... 838
予防的セキュリティのベストプラクティス ... 838
発見的セキュリティベストプラクティス ... 840
CQL 言語リファレンス ... 843
言語要素 ... 843
識別子 ... 844
定数 ... 844
用語 ... 844
データ型 ... 845
Amazon Keyspaces データ型の JSON エンコード ... 849
DDL ステートメント ... 853
Keyspaces ... 853
テーブル ... 857
型 ... 871
DML ステートメント ... 874
SELECT ... 874
INSERT ... 877
UPDATE ... 879
DELETE ... 880
組み込み関数 ... 881
スカラー関数 ... 881
クォータ ... 884

Amazon Keyspaces サービスクォータ 884
スループットの増加または減少 (プロビジョニングされたテーブルの場合) 890
プロビジョニングされたスループットの増加 890
プロビジョニングされたスループットの減少 890
Amazon Keyspaces の保管データ暗号化 890
Amazon Keyspaces のユーザー定義タイプ (UDTsクォータとデフォルト値 891
Amazon Keyspaces UDT クォータとデフォルト値 891
ドキュメント履歴 892
..... cmvi

Amazon Keyspaces (Apache Cassandra 向け) とは

Amazon Keyspaces (Apache Cassandra 向け) は、スケーラブルで可用性の高い、Apache Cassandra 互換のマネージドデータベースサービスです。Amazon Keyspaces を使用すると、サーバーのプロビジョニング、パッチ、管理、さらには、ソフトウェアのインストール、メンテナンス、運用も、ユーザーが行う必要がなくなります。

Amazon Keyspaces はサーバーレスであるため、使用するリソースに対してのみ課金され、アプリケーショントラフィックに応じてテーブルの拡大と縮小が自動的に行われます。スループットとストレージが実質的に無制限で、毎秒数千件のリクエストを処理できるアプリケーションを構築できます。

> **Note**
>
> Apache Cassandra は、データの大量を可能にするオープンソースのワイド列データストアです。詳細については、「Apache Cassandra」を参照してください。

Amazon Keyspaces を使用すると、 AWS クラウドにおける Cassandra ワークロードの移行、実行、スケーリングが簡単になります。 AWS マネジメントコンソールまたは数行のコードをクリックするだけで、インフラストラクチャをデプロイしたりソフトウェアをインストールしたりすることなく、Amazon Keyspaces でキースペースとテーブルを作成できます。

Amazon Keyspaces では、現在使用している AWS のと同じ Cassandra アプリケーションコードと開発者ツールを使用して、 で既存の Cassandra ワークロードを実行できます。

Github で利用可能な Amazon Keyspaces (Apache Cassandra 向け) の料金計算ツールを使用すると、既存の Apache Cassandra ワークロードに基づいて Amazon Keyspaces の月額コストを見積もることができます。Amazon Keyspaces の Cassandra nodetool ステータス出力と意図したサーバーレス設定からメトリクスを入力して、2 つのソリューション間の直接コストを比較します。この計算ツールは、既存の Cassandra デプロイと比較した Amazon Keyspaces の運用コストにのみ重点を置いていることに注意してください。インフラストラクチャのメンテナンス、運用オーバーヘッド、Cassandra のサポートコストなどの総所有コスト (TCO) 要因は含まれません。

使用可能な エンドポイント AWS リージョン と エンドポイントのリストについては、「Amazon Keyspaces のサービスエンドポイント」を参照してください。

最初に以下のセクションを読んでおくことをお勧めします。

トピック

- Amazon Keyspaces: 仕組み
- Amazon Keyspaces のユースケース
- Cassandra クエリ言語 (CQL) とは

Amazon Keyspaces: 仕組み

Amazon Keyspaces では、Cassandra の管理にかかる管理オーバーヘッドが取り除かれます。その理由を理解するには、Cassandra アーキテクチャについて調べた上で Amazon Keyspaces と比較するとよいでしょう。

トピック

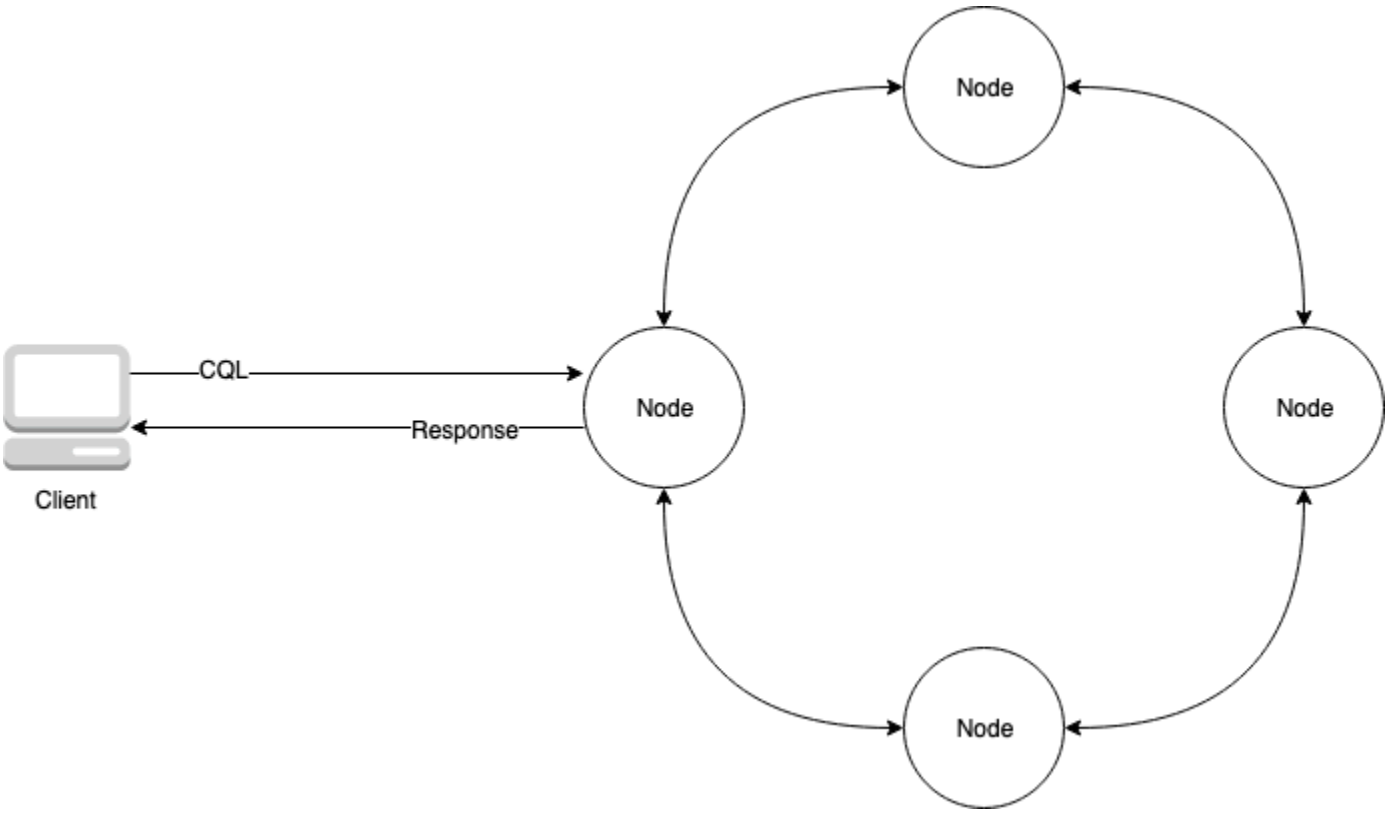
- 高レベルのアーキテクチャ: Apache Cassandra と Amazon Keyspaces の比較
- Cassandra のデータモデル
- アプリケーションからの Amazon Keyspaces へのアクセス

高レベルのアーキテクチャ: Apache Cassandra と Amazon Keyspaces の比較

従来の Apache Cassandra は 1 つ以上のノードで構成されるクラスターにデプロイされます。各ノードの管理と、クラスターのスケールに応じたノードの追加および削除は、ユーザーに責任があります。

クライアントプログラムは、いずれかのノードに接続して Cassandra クエリ言語 (CQL) ステートメントを発行することにより Cassandra にアクセスします。CQL は、リレーショナルデータベースで使用される一般的な言語である SQL に似ています。Cassandra はリレーショナル・データベースではありませんが、CQL は Cassandra 内のデータのクエリと操作のための使い慣れたインタフェースを提供します。

次の図は、4 つのノードで構成されるシンプルな Apache Cassandra クラスターを示しています。



本番稼働用の Cassandra のデプロイは、1 つまたは複数の物理データセンターで数百台の物理コンピュータ上で動作する数百のノードで構成されている可能性があります。ソフトウェアのインストール、保守、および運用に加えて、サーバーのプロビジョニング、パッチ適用、管理を行わなければならないアプリケーションデベロッパーに、運用上の負担がかかる可能性があります。

Amazon Keyspaces (Apache Cassandra 向け) では、サーバーのプロビジョニング、パッチ適用、管理が不要なため、より優れたアプリケーションを構築することに集中できます。Amazon Keyspaces では、読み取りと書き込みの 2 つのスループットキャパシティモード (オンデマンドとプロビジョニング) があります。テーブルのスループットキャパシティモードを選択すれば、ワークロードの予測可能性と変動性に基づいて読み取りと書き込みの料金を最適化できます。

オンデマンドモード (デフォルト) - アプリケーションにより実行される実際の読み取りと書き込みに対してのみ料金が発生します。テーブルのスループットキャパシティを事前に指定する必要はありません。Amazon Keyspaces は、アプリケーショントラフィックの上昇と下降にほぼ瞬時に対応するため、トラフィックが予測不能であるアプリケーションに適しています。

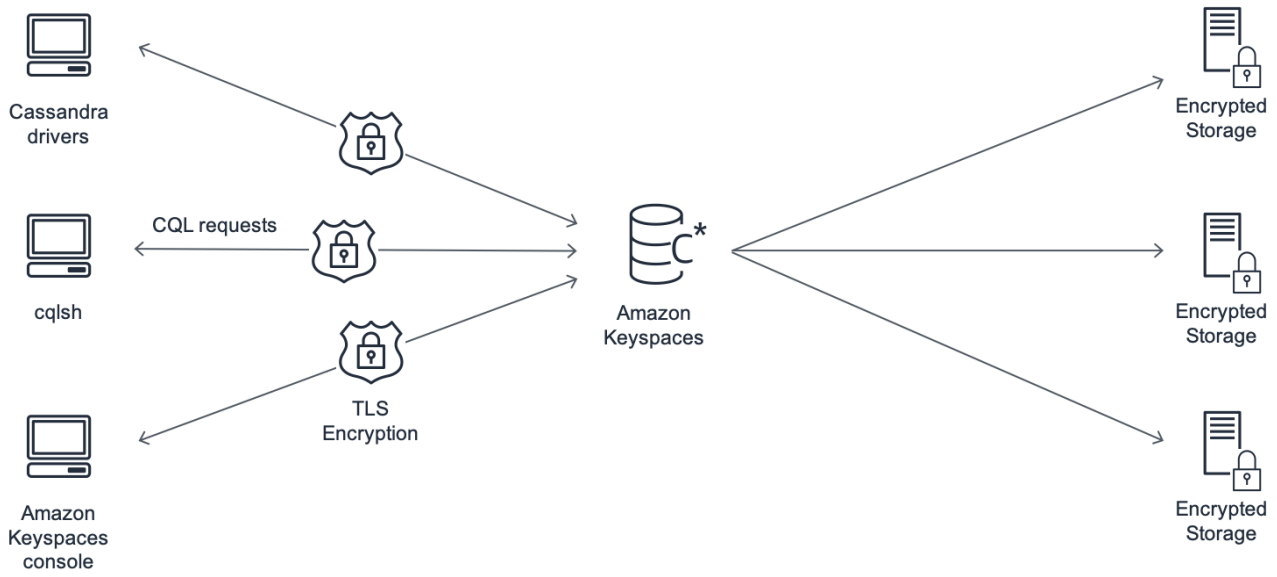
プロビジョンドキャパシティモードでは、予測可能なアプリケーショントラフィックがあり、テーブルのキャパシティ要件を事前に予測できる場合に、スループットの料金を最適化できます。プロビジョンドキャパシティモードを使用する場合、アプリケーションに必要な 1 秒あたりの読み取り回

数と書き込み回数を指定します。オートスケーリングを有効にすると、テーブルのプロビジョンドキャパシティが自動的に調整されます。

ワークロードのトラフィックパターンについて詳しく学んだとおり、大量のテーブルトラフィックの発生が予想される主要なイベントなど、トラフィックの大幅なバーストが予想される場合は、テーブルのキャパシティモードを 1 日 1 回変更することができます。読み取り/書き込みキャパシティのプロビジョニングについては、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。

Amazon Keyspaces (Apache Cassandra 向け) では、耐久性と高可用性のために、3 つのデータコピーが複数のアベイラビリティーゾーンに保存されます。さらに、データセンターと、セキュリティを非常に重視する組織の要件を満たせるように構築されたネットワークアーキテクチャーというメリットがあります。保管データ暗号化は、新しい Amazon Keyspaces テーブルを作成し、すべてのクライアント接続に Transport Layer Security (TLS) が必要になった場合に、自動的に有効化されます。その他の AWS セキュリティ機能には、モニタリングAWS Identity and Access Management、、仮想プライベートクラウド (VPC) エンドポイントなどがあります。使用可能なすべてのセキュリティ機能の概要については、「セキュリティ」を参照してください。

Amazon Keyspaces のアーキテクチャを次の図に示します。



クライアントプログラムは、事前に定義されたエンドポイント (ホスト名とポート番号) に接続し、CQL ステートメントを発行して、 Amazon Keyspaces にアクセスします。利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

Cassandra のデータモデル

Amazon Keyspaces から最適なパフォーマンスの達成を目指す上で、ビジネスケースのデータをどうモデル化するかが重要になります。データモデルが貧弱だとパフォーマンスが大幅に低下する可能性があります。

CQL は SQL に似ていますが、Cassandra とリレーショナルデータベースではバックエンドが大きく異なっているため、別の方法でアプローチする必要があります。次に、考慮すべき重要な問題をいくつか取り上げます。

[Storage (ストレージ)]

テーブルで Cassandra データを可視化することができます。各行はレコードを表し、各列はそのレコード内のフィールドを表します。

テーブルデザイン: クエリ重視

CQL には `JOIN` がありません。したがって、データの形状と、ビジネスユースケースのためにデータにどうアクセスする必要があるかを考慮して、テーブルを設計する必要があります。これにより、重複データがある非正規化が生じる可能性があります。各テーブルを特定のアクセスパターンに合わせて設計する必要があります。

パーティション

データはディスク上のパーティションに保存されます。データが保存されているパーティションの数とパーティション間でデータの分散状況は、パーティションキーによって決まります。パーティションキーの定義方法は、クエリのパフォーマンスに大きな影響を与える可能性があります。ベストプラクティスについては、the section called “パーティションキーの設計”を参照してください。

プライマリキー

Cassandra では、データはキーバリューペアとして保存されます。Cassandra のすべてのテーブルには、テーブルの各行を一意に識別するプライマリキーが必要です。プライマリキーは、必須のパーティションキー 1 つと、オプションのクラスタリング列 1 つ以上で構成されます。プライマリキーを構成するデータは、テーブルのすべてのレコードに対して一意でなければなりません。

- パーティションキー — プライマリキーのパーティションキー部分は必須で、データが保存されるクラスターのパーティションを決定します。パーティションキーは、1 つの列である場合もあれば、2 つ以上の列で構成される複合値である場合もあります。単一列パーティションキーを使用すると、ほとんどのデータが単一のパーティションまたは非常に少数のパーティ

ションに保存され、そのためにディスク I/O オペレーションの大部分を単一列パーティションキーが占める結果となる場合は、複合パーティションキーを使用した方がよいでしょう。

- クラスタリング列 – プライマリキーのオプションのクラスタリング列部分は、各パーティションにおけるデータのクラスター処理方法とソート方法を決定するものです。プライマリキーにクラスタリング列を含めると、クラスタリング列には 1 つ以上の列が存在します。クラスタリング列内に複数の列が存在する場合、クラスタリング列内の列の順序 (左から右) によってソート順序が決まります。

NoSQL 設計と Amazon Keyspaces の詳細については、「the section called “NoSQL 設計”」を参照してください。Amazon Keyspaces およびデータモデリングの詳細については、「the section called “データモデリング”」を参照してください。

アプリケーションからの Amazon Keyspaces へのアクセス

すでに使用している CQL および Cassandra ドライバーを使用できるように、Amazon Keyspaces (Apache Cassandra 向け) には Apache Cassandra クエリ言語 (CQL) API が実装されています。アプリケーションの更新は、Cassandraドライバーの更新や、Amazon Keyspaces サービスエンドポイントを指すための `cqlsh` 設定と同じくらい簡単です。必要な認証情報の詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

> **Note**
>
> 開始にあたっては、GitHub で Amazon Keyspaces コードサンプルリポジトリにあるさまざまな Cassandra クライアントドライバーを使用して、Amazon Keyspaces への接続に関するエンドツーエンドのコードサンプルを利用することができます。

次の Python プログラムで Cassandra クラスターに接続してテーブルをクエリする場合を考えてみましょう。

```
from cassandra.cluster import Cluster
#TLS/SSL configuration goes here

ksp = 'MyKeyspace'
tbl = 'WeatherData'

cluster = Cluster(['NNN.NNN.NNN.NNN'], port=NNNN)
session = cluster.connect(ksp)
```

```
session.execute('USE ' + ksp)

rows = session.execute('SELECT * FROM ' +  tbl)
for row in rows:
    print(row)
```

Amazon Keyspaces に対して同じプログラムを実行するには、次の作業を行う必要があります。

- クラスターのエンドポイントとポートの追加: 例えば、ホストを `cassandra.us-east-2.amazonaws.com` やポート番号 9142 などのサービスエンドポイントに置き換えることができます。
- TLS/SSL 設定の追加: Cassandra クライアントの Python ドライバーを使用して Amazon Keyspaces に接続するための TLS/SSL 設定の追加については、「Cassandra Python クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス」を参照してください。

Amazon Keyspaces のユースケース

以下は、Amazon Keyspaces の使用方法のほんの一部です。

- 低レイテンシーを必要とするアプリケーションを構築する - 産業機器のメンテナンス、取引のモニタリング、フリート管理、ルートの最適化など、1 桁ミリ秒のレイテンシーを必要とするアプリケーション向けにデータを高速で処理します。
- オープンソーステクノロジーを使用してアプリケーションを構築する – Java、Python、Ruby、Microsoft .NET、Node.js、PHP、C++、Perl、Go など、幅広いプログラミング言語で使用できるオープンソースの Cassandra APIs とドライバー AWS を使用して、 でアプリケーションを構築します。コード例については、ツールとライブラリ を参照してください。
- Cassandra ワークロードをクラウドに移動 — Cassandra テーブルをユーザー自身が管理する場合は時間とコストがかかります。Amazon Keyspaces を使用すると、インフラストラクチャを管理する AWS クラウド ことなく、 で Cassandra テーブルをセットアップ、保護、スケーリングできます。詳細については、「サーバーレスリソースの管理」を参照してください。

Cassandra クエリ言語 (CQL) とは

Cassandra クエリ言語 (CQL) は Apache Cassandra との通信に使用される主要言語です。Amazon Keyspaces (Apache Cassandra 向け) は、CQL 3.x API (バージョン 2.x との下位互換性あり) との互換性があります。

CQL では、データはテーブル、列、行に格納されます。この点で、CQL は構造化クエリ言語 (SQL) と似ています。以下は CQL の主要な概念です。

- CQL 要素 – CQL の基本要素には、識別子、定数、項、データ型があります。
- データ定義言語 (DDL) – DDL ステートメントは、Amazon Keyspaces の AWS リソースであるキースペースやテーブルなどのデータ構造を管理するために使用されます。DDL ステートメントは、 のコントロールプレーンオペレーションです AWS。
- データ操作言語 (DML) – DML ステートメントは、テーブル内のデータを管理するために使用されます。DML ステートメントは、データの選択、挿入、更新、削除に使用されます。これらは、 でのデータプレーンオペレーションです AWS。
- 組み込み関数 – Amazon Keyspaces は、さまざまな組み込みスカラー関数をサポートしています。これらの関数を CQL ステートメントで使用できます。

CQL の詳細については、「Amazon Keyspaces (Apache Cassandra 向け) の CQL 言語リファレンス」を参照してください。Apache Cassandra との機能の違いについては、「the section called "Apache Cassandra との機能の違い"」を参照してください。

CQL クエリを実行するには、次の作業のいずれかを実行します。

- AWS Management Consoleで CQL エディタを使用する。
- AWS CloudShell と cqlsh-expansion を使用します。
- `cqlsh` クライアントを使用する。
- Apache 2.0 ライセンスの Cassandra クライアントドライバーを使用する。

CQL に加えて、 AWS SDKs と を使用して Amazon Keyspaces でデータ定義言語 (DDL) オペレーションを実行できます AWS Command Line Interface。

これらの方法を使用して Amazon Keyspaces にアクセスする方法については、「Amazon Keyspaces (Apache Cassandra 向け) へのアクセス」を参照してください。

Amazon Keyspaces (Apache Cassandra 向け) と Apache Cassandra の比較について

Amazon Keyspaces への接続を確立するには、パブリック AWS サービスエンドポイント、または Amazon Virtual Private Cloud 内でインターフェイス VPC エンドポイント (AWS PrivateLink) を使用したプライベートエンドポイントのいずれかを使用できます。使用するエンドポイントに応じて、クライアントからの Amazon Keyspaces の見え方は次のいずれかになります。

AWS サービスエンドポイント接続

パブリックエンドポイントを介して確立された接続です。この場合、Amazon Keyspaces は、クライアントからは 9 ノードの Apache Cassandra 3.11.2 クラスターのように見えます。

インターフェイス VPC エンドポイント接続

インターフェイス VPC エンドポイントを使用して確立されたプライベート接続です。この場合、Amazon Keyspaces は、クライアントからは 3 ノードの Apache Cassandra 3.11.2 クラスターのように見えます。

接続タイプやクライアントから認識できるノードの数に関係なく、Amazon Keyspaces は実質的に無制限のスループットとストレージを提供します。そのために、Amazon Keyspaces はノードをロードバランサーにマッピングし、このバランサーが、基盤となる多くのストレージパーティションの 1 つにクエリをルーティングします。接続の詳細については、「the section called “働き”」を参照してください。

Amazon Keyspaces では、データはパーティションに格納されます。パーティションとは、テーブル用に割り当てられたストレージ領域であり、そのストレージの提供元はソリッドステートドライブ (SSD) です。Amazon Keyspaces は、耐久性と高可用性 AWS リージョン のために、 内の複数のアベイラビリティーゾーンにデータを自動的にレプリケートします。スループットやストレージの需要が増えるに従い、パーティション管理が自動で処理され、必要な追加のパーティションが自動的にプロビジョニングされます。

Amazon Keyspaces は、キースペースとテーブルの作成、データの読み取り、データの書き込みなど、一般的に使用されるあらゆる Cassandra データプレーンオペレーションに対応しています。Amazon Keyspaces はサーバーレスであるため、サーバーのプロビジョニング、パッチ、管理を行う必要はありません。ソフトウェアのインストール、保守、操作も不要です。したがって、Amazon Keyspaces では、クラスターとノードの設定を管理するための Cassandra コントロールプレーン API オペレーションを使用する必要がありません。

レプリケーション係数や整合性レベルなどの設定は自動で行われ、高可用性、耐久性、一桁ミリ秒単位のパフォーマンスが実現されます。耐障害性と低レイテンシーのローカル読み取りをさらに強化するために、Amazon Keyspaces はマルチリージョンレプリケーションにも対応しています。

トピック

- 機能の違い: Amazon Keyspaces と Apache Cassandra
- サポートされている Cassandra API、オペレーション、関数、データ型
- Apache Cassandra でサポートされている読み取り/書き込み整合性レベルと関連コスト

機能の違い: Amazon Keyspaces と Apache Cassandra

Amazon Keyspaces と Apache Cassandra の機能上の違いは以下のとおりです。

トピック

- Apache Cassandra の API、オペレーション、およびデータ型
- キースペースとテーブルの非同期的な作成および削除
- 認証と認可
- バッチ
- 変更データキャプチャ (CDC)
- クラスターの設定
- Connections
- IN キーワード
- FROZEN collections
- 軽量トランザクション
- 負荷分散
- Pagination (ページ分割)
- パーティショナー
- プリペアドステートメント
- 範囲削除
- システムテーブル
- タイムスタンプ

- ユーザー定義タイプ (UDTs)

Apache Cassandra の API、オペレーション、およびデータ型

Amazon Keyspaces は、キースペースとテーブルの作成、データの読み取り、データの書き込みなど、一般的に使用されるあらゆる Cassandra データプレーンオペレーションに対応しています。現在サポートされているものについては、「サポートされている Cassandra API、オペレーション、関数、データ型」を参照してください。

キースペースとテーブルの非同期的な作成および削除

Amazon Keyspaces は、キースペース 、テーブル、タイプを非同期的に作成および削除するなど、データ定義言語 (DDL) オペレーションを実行します。リソースの作成状況を監視する方法については、「the section called “キースペースの作成ステータスの確認”」と「the section called “テーブルの作成ステータスの確認”」を参照してください。CQL 言語リファレンスの DDL ステートメントのリストについては、「the section called “DDL ステートメント”」を参照してください。

認証と認可

Amazon Keyspaces (Apache Cassandra 向け) では、 AWS Identity and Access Management (IAM) がユーザーの認証と認可に使用され、Apache Cassandra と同等の認可ポリシーがサポートされています。そのため、Amazon Keyspaces では Apache Cassandra のセキュリティ設定コマンドをサポートしていません。

バッチ

Amazon Keyspaces では、バッチ内に最大 30 個のコマンドが含まれており、ログに記録されていないバッチコマンドがサポートされています。無条件の INSERT、UPDATE、または DELETE コマンドだけがバッチで許可されます。ログに記録されたバッチはサポートしていません。

変更データキャプチャ (CDC)

Apache Cassandra では、変更データキャプチャ (CDC) は、テーブルの更新をキャプチャするコミットログセグメントをノード上の指定されたディレクトリにコピーします。CDC ログの各変更レコードには、複数のキースペースとテーブルにわたる更新を含めることができます。行が更新または削除されると、Apache Cassandra CDC ログには変更された列のみが表示されます。ログには、変更前のデータの以前の状態は含まれません。データ変更の詳細をすべて理解するには、特定の Cassandra ツールを使用してこれらのログを逆シリアル化する必要があります。

Amazon Keyspaces CDC ストリームでは、CDC ストリームが行ごとに収集する情報のタイプを選択できます。デフォルトでは、Amazon Keyspaces CDC ストリームは変更前後の行のバージョンをキャプチャします。Amazon Keyspaces CDC は、重複排除された変更レコードと順序付けられた変更レコードをテーブルレベルで提供します。

Amazon Keyspaces では、各 CDC ストリームは Amazon AWS リソースネーム (ARN) を持つ リソースであり、アプリケーションは利用可能なエンドポイントのいずれかを使用して Amazon Keyspaces CDC API を使用して Amazon Keyspaces CDC ストリームを消費できます。

Amazon Keyspaces CDC の詳細については、「」を参照してくださいthe section called “CDC ストリームの使用”。

クラスターの設定

Amazon Keyspaces はサーバーレスであるため、クラスター、ホスト、Java 仮想マシン (JVM) の設定は不要です。Cassandra のコンパクション、圧縮、キャッシュ、ガベージコレクション、ブルームフィルタリングの各設定は、Amazon Keyspaces には適用されず、指定すれば無視できます。

Connections

既存の Cassandra ドライバーで Amazon Keyspaces と通信できますが、ドライバーには異なる設定が必要です。Amazon Keyspaces は、TCP 接続 1 つにつき 1 秒あたり最大 3,000 の CQL クエリに対応していますが、ドライバーが確立できる接続数に制限はありません。

ほとんどのオープンソース Cassandra ドライバーで、Cassandra への接続プールが確立され、その接続プールでクエリのロードバランスが行われます。Amazon Keyspaces では 9 つのピア IP アドレスがドライバーに公開されており、ほとんどのドライバーのデフォルトの動作は、各ピアの IP アドレスに対して接続を 1 つずつ確立することです。したがって、デフォルト設定を使用するドライバーの最大 CQL クエリスループットは、1 秒あたり 27,000 CQL クエリになります。

この数を増やすには、接続プールでドライバーにより維持されている各 IP アドレスの接続数を増やすことをお勧めします。例えば、IP アドレスあたりの最大接続数を 2 に設定すると、ドライバーの最大スループットが 1 秒あたり 54,000 CQL クエリの 2 倍になります。

ベストプラクティスとして、オーバーヘッドを考慮して配信を改善するために、接続ごとに 1 秒あたり 500 件の CQL クエリを使用するようにドライバーを設定することをお勧めします。このシナリオでは、1 秒あたり 18,000 件の CQL クエリが想定される場合、36 本の接続が必要です。9 つのエンドポイントにまたがる 4 本の接続でドライバーを設定する場合、36 本の接続で 1 秒あたり 500 件のリクエストを処理することになります。接続に関する詳しいベストプラクティスについては、「the section called “Connections”」を参照してください。

VPC エンドポイントに接続すると、使用可能なエンドポイント数が少なくなる可能性があります。したがって、ドライバー設定内の接続数を増やす必要があります。VPC 接続のベストプラクティスの詳細については、「the section called “VPC エンドポイント接続”」を参照してください。

IN キーワード

Amazon Keyspaces は、SELECT ステートメント内の IN キーワードをサポートしています。 IN は UPDATE と DELETE ではサポートしていません。SELECT ステートメントで IN キーワードを使用すると、クエリ結果は SELECT ステートメントにキーが示されている順序で返されます。Cassandra では、結果は辞書順に並べられます。

ORDER BY を使用するとき、ページ分割を無効にした状態の完全な並べ替えはサポートしていません。結果はページ内で順序付けられます。IN キーワードではスライスクエリはサポートしていません。 TOKENS は、IN キーワードではサポートしていません。Amazon Keyspaces は、サブクエリを作成して IN キーワードのあるクエリを処理します。各サブクエリは、TCP 接続あたり 1 秒あたり 3,000 件の CQL クエリという制限に対する接続としてカウントされます。詳細については、「the section called “IN SELECT の使用”」を参照してください。

FROZEN collections

Cassandra FROZEN のキーワードでは、コレクションデータ型の複数のコンポーネントが 1 つの不変の値にシリアル化され、その値は BLOB のように扱われます。INSERT ステートメントと UPDATE ステートメントはコレクション全体を上書きします。

Amazon Keyspaces は、デフォルトでフリーズコレクションに対して最大 8 レベルのネストをサポートしています。詳細については、「the section called “Amazon Keyspaces サービスクォータ”」を参照してください。

Amazon Keyspaces は、条件 UPDATE ステートメントまたは SELECT ステートメントでフリーズコレクション全体を使用する不等式比較をサポートしていません。コレクションとフリーズコレクションの動作は Amazon Keyspaces でも同じです。

クライアント側のタイムスタンプでフリーズコレクションを使用している場合、書き込みオペレーションのタイムスタンプが、有効期限が切れていないか、廃棄されていない既存の列のタイムスタンプと同じ場合、Amazon Keyspaces は比較を実行しません。代わりに、サーバーが最新のライターを決定し、最新のライターが優先されます。

フリーズコレクションの詳細については、「the section called “コレクション型”」を参照してください。

軽量トランザクション

Amazon Keyspaces (Apache Cassandra 向け) では、INSERT コマンド、UPDATE コマンド、および DELETE コマンドの比較機能と設定機能をすべてサポートしています。これらの機能は、Apache Cassandra では軽量トランザクション (LWT) という名称です。Amazon Keyspaces (Apache Cassandra 向け) はサーバーレス製品として、軽量トランザクションを含め、あらゆる規模で一貫性のあるパフォーマンスを提供します。Amazon Keyspaces では、軽量トランザクションを使用してもパフォーマンス上のペナルティはありません。

負荷分散

`system.peers` テーブルエントリは、Amazon Keyspaces ロードバランサーに対応します。最良の結果を得るために、ラウンドロビンロードバランシングポリシーを使用し、アプリケーションのニーズに合わせて IP ごとに接続数を調整することをお勧めします。

Pagination (ページ分割)

Amazon Keyspaces では、結果セットで返された行数ではなく、リクエストを処理するために読み取られた行の数に基づいて結果のページ分割が行われます。その結果、一部のページの行数が、フィルタリングされたクエリに対して PAGE SIZE で指定した行数よりも少なくなる可能性があります。Amazon Keyspaces ではさらに、1 MB のデータの読み取り後に結果を自動的にページ分割して、一貫した 1 桁台のミリ秒の読み取りパフォーマンスをお客様にお届けします。詳細については、「the section called “結果のページ分割”」を参照してください。

静的列があるテーブルでは、Apache Cassandra および Amazon Keyspaces のいずれでも、クエリの結果が複数ページにまたがる場合、各ページの開始時にそのパーティションの静的列の値が設定されます。テーブルに大きなデータ行がある場合、Amazon Keyspaces のページ分割の仕様上、Apache Cassandra よりも Amazon Keyspaces の方が、範囲読み取りオペレーションの結果のページ数が多くなりやすい傾向にあります。その結果、Amazon Keyspaces では、静的列の並列更新が行われると、範囲読み取りの結果セットの異なるページで静的列の値が異なる可能性が高くなります。

パーティショナー

Amazon Keyspaces デフォルトパーティショナーは Cassandra 互換の `Murmur3Partitioner` です。さらに、Amazon Keyspaces `DefaultPartitioner` と Cassandra 互換の `RandomPartitioner` のどちらを使用するかを選択できます。

Amazon Keyspaces を使用すると、Amazon Keyspaces データを再読み込みしなくても、アカウントのパーティショナーを安全に変更できます。約 10 分かかる設定変更が完了すると、クライアントが次に接続したときに新しいパーティショナー設定が自動的に表示されます。詳細については、「the section called “パーティショナーの操作”」を参照してください。

プリペアドステートメント

Amazon Keyspaces では、データの読み取りや書き込みなどのデータ操作言語 (DML) オペレーションに対するプリペアドステートメントの使用がサポートされています。Amazon Keyspaces は現在、テーブルやキースペースの作成などのデータ定義言語 (DDL) オペレーションでのプリペアドステートメントの使用をサポートしていません。DDL オペレーションは、プリペアドステートメントの外部で実行する必要があります。

範囲削除

Amazon Keyspaces では、一定範囲内の行の削除がサポートされています。範囲とは、パーティション内の連続する行のセットです。DELETE オペレーションで範囲を指定するには、WHERE 句を使用します。範囲をパーティション全体に指定することもできます。

さらに、関係演算子 (「>」、「<」など) を使用することによって、または、パーティションキーを含めて 1 つ以上のクラスタリング列を省略することによって、パーティション内の連続する行のサブセットとして範囲を指定できます。Amazon Keyspaces を使用すると、1 回のオペレーションで 1 つの範囲内の行を最大で 1,000 行削除できます。

範囲削除は分離されていません。範囲削除の処理中でも、個々の行の削除は他のオペレーションから認識できる状態です。

システムテーブル

Amazon Keyspaces は、Apache 2.0 オープンソースの Cassandra ドライバーに必要なシステムテーブルに情報を入力します。クライアントに表示されるシステムテーブルには、認証されたユーザーに固有の情報が含まれています。システムテーブルは Amazon Keyspaces によって完全に制御されるものであり、読み取り専用です。詳細については、「the section called “システムキースペース”」を参照してください。

システムテーブルへの読み取り専用アクセス権限が必要です。この権限は、IAM アクセスポリシーで制御できます。詳細については、「the section called “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。Cassandra ドライバーと開発者ツールを使用して AWS SDK または Cassandra クエリ言語 (CQL) API コールを使用するかどうかに応じて、システムテーブルのタグベースのアクセスコントロールポリシーを異なる方法で定義する必要があります。システムテーブルのタグベー

スのアクセスコントロールに関する詳細については、「the section called “ タグに基いた Amazon Keyspaces リソースアクセス”」を参照してください。

Amazon VPC エンドポイントで Amazon Keyspaces にアクセスすると、Amazon キKeyspaces が表示権限を持っている各 Amazon VPC エンドポイントのエントリが `system.peers` テーブルに表示されます。その結果、Cassandra ドライバーによって `system.peers` テーブル内のコントロールノードそのものに関する警告メッセージが表示される場合があります。この警告は無視しても問題ありません。

タイムスタンプ

Amazon Keyspaces では、Apache Cassandra のデフォルトのタイムスタンプと互換性のあるセルレベルのタイムスタンプはオプトイン機能です。

`USING TIMESTAMP` 句と `WRITETIME` 関数は、テーブルのクライアント側のタイムスタンプがオンになっているときにのみ使用できます。Amazon Keyspaces のクライアント側のタイムスタンプの詳細については、「the section called “クライアントサイドのタイムスタンプ”」を参照してください。

ユーザー定義タイプ (UDTs)

不等式演算子は、Amazon Keyspaces の UDTs ではサポートされていません。

Amazon UDTs を使用する方法については、「」を参照してくださいthe section called “ユーザー定義タイプ (UDTs)”。

キースペースごとにサポートされている UDTs の数、サポートされているネストレベル、および UDTs「」を参照してくださいthe section called “Amazon Keyspaces のユーザー定義タイプ (UDTs クォータとデフォルト値”。

サポートされている Cassandra API、オペレーション、関数、データ型

Amazon Keyspaces (Apache Cassandra 用) は、Cassandra クエリ言語 (CQL) 3.11 API (バージョン 2.x との下位互換性あり) と互換性があります。

Amazon Keyspaces は、キースペースとテーブルの作成、データの読み取り、データの書き込みなど、一般的に使用されるあらゆる Cassandra データプレーンオペレーションに対応しています。

次のセクションではサポートされている機能のリストを示します。

トピック

- Cassandra API サポート
- Cassandra コントロールプレーン API サポート
- Cassandra データプレーン API サポート
- Cassandra の関数サポート
- Cassandra データ型サポート

Cassandra API サポート

API オペレーション	サポート
CREATE KEYSPACE	はい
ALTER KEYSPACE	あり
DROP KEYSPACE	あり
CREATE TABLE	あり
ALTER TABLE	あり
DROP TABLE	あり
CREATE INDEX	なし
DROP INDEX	なし
UNLOGGED BATCH	あり
LOGGED BATCH	なし
SELECT	あり
INSERT	あり
DELETE	あり
UPDATE	あり

API オペレーション	サポート
USE	あり
CREATE TYPE	あり
ALTER TYPE	なし
DROP TYPE	あり
CREATE TRIGGER	なし
DROP TRIGGER	なし
CREATE FUNCTION	なし
DROP FUNCTION	なし
CREATE AGGREGATE	なし
DROP AGGREGATE	なし
CREATE MATERIALIZED VIEW	なし
ALTER MATERIALIZED VIEW	なし
DROP MATERIALIZED VIEW	なし
TRUNCATE	なし

Cassandra コントロールプレーン API サポート

Amazon Keyspaces が管理されているため、クラスターとノードの設定を管理するための Cassandra コントロールプレーン API オペレーションは不要です。したがって、Cassandra の以下の機能は適用されません。

機能	理由
耐久性のある書き込みトグル	すべての書き込みに耐久性がある

機能	理由
読み取り修復設定	該当しない
GC 猶予期間 (秒)	該当しない
ブルームフィルター設定	該当しない
圧縮設定	該当しない
[Compression settings] (圧縮設定)	該当しない
キャッシュ設定	該当しない
セキュリティ設定	IAM に置き換わった

Cassandra データプレーン API サポート

機能	サポート対象
SELECT ステートメントと INSERT ステートメントの JSON サポート	あり
静的列	あり
有効期限 (TTL)	あり

Cassandra の関数サポート

サポートされている関数の詳細については、「the section called “組み込み関数”」を参照してください。

関数	サポート
`Aggregate 関数`	なし
`Blob 変換`	はい

関数	サポート
Cast	あり
Datetime 関数	あり
時間変換関数	あり
TimeUuid 関数	はい
Token	あり
User defined functions (UDF)	なし
Uuid	あり

Cassandra データ型サポート

次の表に、Amazon Keyspaces でサポートされている Apache Cassandra データ型を示します。Amazon Keyspaces のデータ型の詳細については、「」を参照してくださいthe section called "データ型"。

データ型	サポート
ascii	はい
bigint	あり
blob	あり
boolean	あり
counter	あり
date	あり
decimal	あり
double	あり

データ型	サポート	
float	あり	
frozen	あり	
inet	あり	
int	あり	
list	あり	
map	あり	
set	あり	
smallint	あり	
text	あり	
time	あり	
timestamp	あり	
timeuuid	あり	
tinyint	あり	
tuple	あり	
user-defined types (UDTs)	あり	
uuid	あり	
varchar	あり	
varint	はい	

Apache Cassandra でサポートされている読み取り/書き込み整合性レベルと関連コスト

このセクションのトピックでは、Amazon Keyspaces (Apache Cassandra 向け) における読み取りオペレーションと書き込みオペレーションに対してサポートされている Apache Cassandra 整合性レベルについて説明します。

トピック

- 書き込みの整合性レベル
- 読み取りの整合性レベル
- サポートされていない整合性レベル

書き込みの整合性レベル

Amazon Keyspaces では、耐久性と高可用性を確保するために、複数のアベイラビリティーゾーンにおいて、全ての書き込みオペレーションが 3 回レプリケートされます。書き込みは、書き込みが承認される前に LOCAL_QUORUM 整合性レベルを使用して永続的に保存されます。1 KB の書き込み 1 回につき、プロビジョンドキャパシティモードを使用するテーブルの場合は 1 書き込みキャパシティユニット (WCU)、オンデマンドモードを使用するテーブルの場合は 1 書き込みリクエストユニット (WRU) が課金されます。

cqlsh を使用すると、現在のセッションのすべてのクエリの整合性を次のコードで LOCAL_QUORUM に設定できます。

```
CONSISTENCY LOCAL_QUORUM;
```

整合性レベルをプログラムで構成するには、適切な Cassandra クライアントドライバーで整合性を設定します。たとえば、4.x バージョンの Java ドライバーでは、以下に示すように app config ファイルの整合性レベルを設定できます。

```
basic.request.consistency = LOCAL_QUORUM
```

3.x バージョンの Java Cassandra ドライバーを使用している場合は、次のコード例のように .withQueryOptions(new QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM) 追加するとセッションの整合性レベルを指定できます。

```
Session session = Cluster.builder()
                        .addContactPoint(endPoint)
                        .withPort(portNumber)
                        .withAuthProvider(new SigV4AuthProvider("us-east-2"))
                        .withSSL()
                        .withQueryOptions(new
 QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM)
                        .build()
                        .connect();
```

特定の書き込み操作の整合性レベルを設定するには、Java ドライバーを使用しているときに setConsistencyLevel 引数を指定して QueryBuilder.insertInto を呼び出すときの整合性を定義します。

読み取りの整合性レベル

Amazon Keyspaces では、ONE、LOCAL_ONE、LOCAL_QUORUM という 3 つの読み取り整合性レベルがサポートされています。Amazon Keyspaces では、LOCAL_QUORUM 読み取りの処理中に、過去に成功したすべての書き込みオペレーションから最新の更新が反映されているレスポンスが返されます。整合性レベル ONE または LOCAL_ONE を使用すると、読み込みリクエストのパフォーマンスと可用性を向上させることができますが、最近完了した書き込みの結果がレスポンスに反映されない可能性があります。

ONE または LOCAL_ONE の整合性を採用した 4 KB の読み取り 1 回につき、プロビジョンドキャパシティモードを使用するテーブルの場合は 0.5 読み込みキャパシティユニット (RCU)、オンデマンドモードを使用するテーブルの場合は 0.5 読み込みリクエストユニット (RRU) が課金されます。LOCAL_QUORUM の整合性を採用した 4 KB の読み取り 1 回につき、プロビジョンドキャパシティモードを使用するテーブルの場合は 1 読み込みキャパシティユニット (RCU)、オンデマンドモードを使用するテーブルの場合は 1 読み込みリクエストユニット (RRU) が課金されます。

テーブルごとの読み取り整合性と読み取りキャパシティスループットモードに基づいた 4 KB 読み取り 1 回あたりの課金

整合性レベル	プロビジョン済み	[オンデマンド]
ONE	0.5 RCU	0.5 RRU
LOCAL_ONE	0.5 RCU	0.5 RRU
LOCAL_QUORUM	1 RCU	1 RRU

読み取り操作に別の整合性を指定するには、Java ドライバーを使用しているときに `setConsistencyLevel` 引数を指定して `QueryBuilder.select` を呼び出します。

サポートされていない整合性レベル

次の整合性レベルは Amazon Keyspaces ではサポートされていないため、例外が発生します。

サポートされていない整合性レベル

Apache Cassandra	Amazon Keyspaces
EACH_QUORUM	サポートされません
QUORUM	サポートされません
ALL	サポートされません
TWO	サポートされません
THREE	サポートされません
ANY	サポートされません
SERIAL	サポートされません
LOCAL_SERIAL	サポートされません

Amazon Keyspaces (Apache Cassandra 向け) への移行

Amazon Keyspaces (Apache Cassandra 向け) への移行は、企業や組織にとって魅力的なさまざまなメリットをもたらします。Amazon Keyspaces には主に次のような利点があるため、移行先として魅力的な選択肢となっています。

- スケーラビリティ – Amazon Keyspaces は、大量のワークロードを処理し、データ量とトラフィックの増加に合わせてシームレスにスケールできるように設計されています。従来の Cassandra では、スケーリングはオンデマンドでは実行されず、将来のピークに備えて計画しておく必要がありました。Amazon Keyspaces では、需要に応じてデータベースを簡単にスケールアップまたはスケールダウンできるため、アプリケーションのパフォーマンスを損なうことなくトラフィックの急増に対応できます。
- パフォーマンス – Amazon Keyspaces では低レイテンシーでのデータアクセスが可能であるため、並外れた速度でデータをアプリケーションに取得して処理できます。分散型アーキテクチャにより、読み取りと書き込みの操作が複数のノードに分散され、高いリクエストレートでも 1 桁ミリ秒単位の応答時間を一貫して実現できます。
- フルマネージド – Amazon Keyspaces は、 AWSが提供するフルマネージドサービスです。つまり、 は、プロビジョニング、設定、パッチ適用、バックアップ、スケーリングなど、データベース管理の運用面 AWS を処理します。これにより、企業はデータベースの管理タスクよりもアプリケーションの開発に集中できます。
- サーバーレスアーキテクチャ – Amazon Keyspaces はサーバーレスです。前もってキャパシティをプロビジョニングしておく必要がなく、消費したキャパシティの分だけ料金を支払います。サーバーを管理する必要も、インスタンスを選択する必要もありません。このリクエストベースの課金モデルでは、使用したリソースに対してのみ支払いを行うことになり、キャパシティのプロビジョニングや監視の必要がないため、コスト効率が向上し、運用上のオーバーヘッドも最小限に抑えられます。
- NoSQL によるスキーマの柔軟性 – Amazon Keyspaces は NoSQL データモデルを採用しているため、柔軟なスキーマ設計が可能です。Amazon Keyspaces では、構造化データ、半構造化データ、非構造化データを保存でき、変化し続ける多様なデータ型の処理に最適です。また、Amazon Keyspaces は書き込み時にスキーマ検証を実行するため、データモデルを一元的に進化させることができます。この柔軟性により、開発サイクルが短縮され、変化するビジネス要件に容易に適応できます。
- 高可用性と耐久性 – Amazon Keyspaces は、 内の複数のアベイラビリティーゾーンにデータをレプリケートし AWS リージョン、高可用性とデータ耐久性を確保します。レプリケーション、フェイルオーバー、リカバリが自動的に処理されるため、データ損失やサービス中断のリスクが最小限

に抑えられます。Amazon Keyspaces の可用性 SLA は最大 99.999% です。耐障害性と低レイテンシーのローカル読み取りをさらに強化するために、Amazon Keyspaces はマルチリージョンレプリケーションにも対応しています。

- セキュリティとコンプライアンス – Amazon Keyspaces と AWS Identity and Access Management の統合によって、きめ細かいアクセスコントロールが実現します。保管中も転送中もデータを暗号化してセキュリティを強化します。Amazon Keyspaces は、HIPAA、PCI DSS、SOC などの特定のプログラムに対するセキュリティとコンプライアンスについて、第三者監査機関の評価を受けているため、規制要件を満たすことが可能です。詳細については、「the section called “コンプライアンス検証”」を参照してください。
- AWS エコシステムとの統合 – エコシステムの一部として AWS 、Amazon Keyspaces は AWS CloudFormation Amazon CloudWatch AWS のサービスや などの他の とシームレスに統合されます AWS CloudTrail。この統合により、サーバーレスアーキテクチャの構築、Infrastructure as Code の活用、リアルタイムのデータ主導型アプリケーションの作成が可能になります。詳細については、「Amazon Keyspaces のモニタリング」を参照してください。

トピック

- Apache Cassandra から Amazon Keyspaces への移行の計画を立てる
- Amazon Keyspaces へのデータの一括アップロードまたは移行に適したツールを選択する方法

Apache Cassandra から Amazon Keyspaces への移行の計画を立てる

Apache Cassandra から Amazon Keyspaces への移行を成功させるために、適応可能な移行の概念とベストプラクティスを確認し、取り得る選択肢を比較検討することをお勧めします。

このトピックでは、いくつかの主要な概念と利用可能なツールや手法を紹介しながら、移行プロセスの仕組みを概説します。さまざまな移行戦略を評価することで、独自の要件に最適な戦略を選定できます。

トピック

- 機能の互換性
- Amazon Keyspaces の料金を推定する
- 移行戦略を選択する
- Amazon Keyspaces へのオンライン移行: 戦略とベストプラクティス

- オフライン移行プロセス: Apache Cassandra から Amazon Keyspaces への移行
- ハイブリッド移行ソリューションの使用: Apache Cassandra から Amazon Keyspaces への移行

機能の互換性

移行の前に、Apache Cassandra と Amazon Keyspaces の機能面の違いを慎重に検討しておきましょう。Amazon Keyspaces は、キースペースとテーブルの作成、データの読み取り、データの書き込みなど、一般的に使用されるあらゆる Cassandra データプレーンオペレーションに対応しています。

ただし、Amazon Keyspaces がサポートしていない Cassandra API も一部あります。サポートされている API の詳細については、「the section called “サポートされている Cassandra API、オペレーション、関数、データ型”」を参照してください。Amazon Keyspaces と Apache Cassandra のすべての機能の違いについては、「the section called “Apache Cassandra との機能の違い”」で概要をまとめて紹介しています。

現在使用中の Cassandra の API やスキーマを、Amazon Keyspaces でサポートされている機能と比較検討するために、GitHub の Amazon Keyspaces ツールキットから入手可能な互換性スクリプトを実行できます。

互換性スクリプトを使用する方法

1. GitHub から Python 互換性スクリプトをダウンロードし、既存の Apache Cassandra クラスターへのアクセスが可能な場所に移動します。
2. 互換性スクリプトは、CQLSH と類似したパラメータを使用します。`--host` および `--port` には、クラスター内のいずれかの Cassandra ノードへの接続とクエリ実行に使用する IP アドレスとポートを入力します。

   Cassandra クラスターで認証を使用している場合は、`-username` と `-password` も指定する必要があります。互換性スクリプトを実行するには、次のコマンドを使用します。

   ```
   python toolkit-compat-tool.py --host hostname or IP -u "username" -p "password" --
   port native transport port
   ```

Amazon Keyspaces の料金を推定する

ここでは、Amazon Keyspaces の推定コストを計算するために、Apache Cassandra テーブルから収集する必要がある情報をかいつまんで説明します。各テーブルは異なるデータ型を必要とし、サポートする必要がある CQL クエリも異なり、それぞれが特有の読み取り/書き込みトラフィックを維持しています。

要件をテーブルごとに考えれば、Amazon Keyspaces ではリソースの分離や読み取り/書き込みスループットキャパシティモードがテーブル単位になっているため、うまく適合します。Amazon Keyspaces では、テーブルの読み取り/書き込みキャパシティと自動スケーリングポリシーを個別に定義できます。

テーブルの要件を理解すれば、機能、コスト、移行の負荷に基づいて、移行対象のテーブルに優先順位を付けやすくなります。

移行前に、Cassandra のテーブルについて次のメトリクスを収集しておきましょう。これらの情報を基に、Amazon Keyspaces におけるワークロードのコストを推定できます。

- テーブル名 – キースペースとテーブルの完全修飾名。
- 説明 – テーブルの説明 (使用方法や保存するデータの型など)。
- 1 秒あたりの平均読み取り数 — 長期間におけるテーブルへのコーディネーターレベルの読み取りの平均数。
- 1 秒あたりの平均書き込み数 – 長期間におけるテーブルへのコーディネーターレベルの書き込みの平均数。
- 平均行サイズ (バイト単位) – バイト単位の行サイズの平均値。
- ストレージサイズ (GB 単位) – テーブルの raw ストレージサイズ。
- 読み取り整合性の内訳 – 結果整合性 (`LOCAL_ONE` または `ONE`) と強整合性 (`LOCAL_QUORUM`) のある読み取りの割合。

次の表は、移行を計画するにあたって揃える必要があるテーブル関連情報の例を示しています。

テーブル名	説明	1 秒あたりの平均読み取り数	1 秒あたりの平均書き込み数	平均行サイズ (バイト単位)	ストレージサイズ (GB 単位)	読み取り整合性の内訳
mykeyspace.mytable	ショッピングカート履歴の保存用	10,000	5,000	2,200	2,000	100% LOCAL_ONE
mykeyspace.mytable2	最新のプロファイル情報の保存用	20,000	1,000	850	1,000	25% LOCAL_QUORUM 75% LOCAL_ONE

テーブルのメトリクスを収集する方法

このセクションでは、既存の Cassandra クラスターから必要なテーブルメトリクスを収集する手順をステップバイステップで説明します。行サイズ、テーブルサイズ、1 秒あたりの読み取り/書き込みリクエスト数 (RPS) などのメトリクスが該当します。これらの情報から、Amazon Keyspaces のテーブルのスループットキャパシティ要件を評価し、料金を推定できます。

Cassandra ソーステーブルのテーブルメトリクスを収集する方法

1. 行サイズを調べる

   行のサイズは、Amazon Keyspaces における読み取りと書き込みのキャパシティ使用率を判断する上で重要です。次の図は、Cassandra のトークン範囲における典型的なデータ分散を示しています。

GitHub から入手可能な行サイズのサンプルスクリプトを使用して、Cassandra クラスター内の各テーブルの行サイズメトリクスを収集できます。

このスクリプトは、`cqlsh` と `awk` を使用して Apache Cassandra からテーブルデータをエクスポートし、任意で指定したテーブルデータのサンプルセットに基づいて、行サイズの最小値、最大値、平均値、標準偏差を計算します。行サイズのサンプルスクリプトに指定した引数が `cqlsh` に渡されるため、同じパラメータを使用して Cassandra クラスターに接続し、データを読み取ることができます。

以下のステートメントは、この例です。

```
./row-size-sampler.sh 10.22.33.44 9142 \\
   -u "username" -p "password" --ssl
```

Amazon Keyspaces で行サイズを計算する方法の詳細については、「the section called “行サイズの推定”」を参照してください。

2. テーブルサイズを調べる

Amazon Keyspaces では、ストレージを事前にプロビジョニングする必要がありません。テーブルの請求対象サイズが継続的に監視され、ストレージ料金が決定されます。ストレージは GB/月単位で請求されます。Amazon Keyspaces のテーブルサイズは、単一レプリカの raw サイズ (非圧縮) に基づいています。

Amazon Keyspaces でテーブルサイズを監視するには、メトリクス `BillableTableSizeInBytes` を使用できます。これは、 AWS Management Consoleでテーブルごとに表示されます。

Amazon Keyspaces テーブルの請求対象サイズは、次の 2 とおりの方法で推定できます。

- 行サイズの平均値に行数を掛ける。

  Amazon Keyspaces テーブルのサイズは、行サイズの平均値に Cassandra ソーステーブルの行数を掛けることで推定できます。前のセクションで紹介した行サイズのサンプルスクリプトを使用して、行サイズの平均値を取得してください。行数を取得するには、`dsbulk count` などのツールを使用して、ソーステーブル内の行の総数を判断できます。

- `nodetool` を使用してテーブルメタデータを収集する。

  `Nodetool` は、Apache Cassandra ディストリビューションで提供されている管理ツールです。Cassandra プロセスの状態に関するインサイトを提供し、テーブルのメタデータを返します。`nodetool` を使用して、テーブルサイズに関するメタデータをサンプリングし、その情報を基に Amazon Keyspaces でのテーブルサイズを推定することができます。

  使用するコマンドは `nodetool tablestats` です。tablestats は、テーブルのサイズと圧縮率を返します。テーブルのサイズは、テーブルの `tablelivespace` として保存されています。この値を `compression ratio` で割り、そのサイズ値にノード数を掛け、最後に、レプリケーション係数 (通常は 3) で割ります。

  この計算の完全な式は次のとおりです。これを使用して、テーブルのサイズを評価できます。

```
((tablelivespace / compression ratio) * (total number of nodes))/ (replication
 factor)
```

  Cassandra クラスターにノードが 12 あると想定しましょう。`nodetool tablestats` コマンドを実行した結果、`tablelivespace` として 200 GB、`compression ratio` として 0.5 が返されました。キースペースのレプリケーション係数は 3 です。

  この例の場合、計算式は次のようになります。

```
(200 GB / 0.5) * (12 nodes)/ (replication factor of 3)
                        = 4,800 GB / 3
                        = 1,600 GB is the table size estimate for Amazon
 Keyspaces
```

3. 読み取り数と書き込み数を調べる

Amazon Keyspaces テーブルのキャパシティとスケーリングの要件を判断するために、移行前に Cassandra テーブルの読み取りと書き込みのリクエストレートを調べておきましょう。

Amazon Keyspaces はサーバーレスであり、使用した分だけ料金を支払います。一般に、Amazon Keyspaces の読み取り/書き込みスループットの料金は、リクエストの数とサイズに基づいて決まります。

Amazon Keyspaces には 2 つのキャパシティモードがあります。

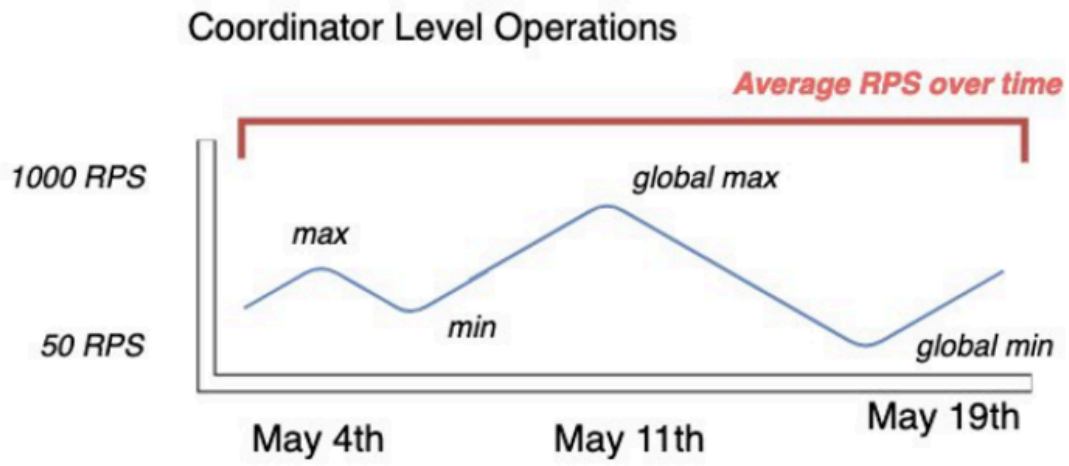
- オンデマンド – キャパシティプランニングの必要なく、1 秒あたり数千のリクエスト数を処理できる柔軟な請求オプションです。読み取りおよび書き込みのリクエスト数ごとの従量課金であるため、使用した分だけを支払います。
- プロビジョンド – プロビジョンドスループットキャパシティモードを選択した場合は、アプリケーションに必要な 1 秒あたりの読み込みと書き込みの数を指定します。これにより、Amazon Keyspaces の使用状況を管理して、定義されたリクエストレート以下を維持し、予測可能性を維持できます。

  プロビジョンドモードでは、自動スケーリングを使用して、プロビジョニングしておいたレートを自動調整してスケールアップまたはスケールダウンし、運用効率を高めることができます。サーバーレスリソース管理の詳細については、「サーバーレスリソースの管理」を参照してください。

Amazon Keyspaces では読み取りと書き込みのスループットキャパシティを個別にプロビジョニングするため、既存のテーブルの読み取りと書き込みのリクエストレートを個別に測定する必要があります。

既存の Cassandra クラスターから最も正確な使用率メトリクスを収集するには、テーブルへのコーディネーターレベルの読み取り/書き込みオペレーションについて、1 秒あたりのリクエスト数 (RPS) の平均値を長期間にわたって観察します。単一のデータセンター内のすべてのノードにわたる集計値から平均値を求めます。

少なくとも数週間にわたる RPS の平均値を取ることで、次の図に示すように、トラフィックパターンのピークと谷を捉えることができます。



Cassandra テーブルの読み取りと書き込みのリクエストレートを判断するには、2 つの選択肢があります。

- 既存の Cassandra モニタリングを使用する

  次の表に示すメトリクスを使用して、読み取りリクエスト数と書き込みリクエスト数を観察できます。メトリクスの名前は、使用しているモニタリングツールによって異なる場合があります。

ディメンション	Cassandra JMX メトリクス
writes	`org.apache.cassandra.metrics:type=ClientRequest,scope=Write,name=Latency#Count`
reads	`org.apache.cassandra.metrics:type=ClientRequest,scope=Read,name=Latency#Count`

- `nodetool` を使用する

  `nodetool tablestats` および `nodetool info` を使用して、テーブルに対する読み取り/書き込みオペレーション数の平均値を求めます。`tablestats` は、ノードが開始された時点からの読み取り数と書き込み数の合計を返します。`nodetool info` は、ノードの稼働時間を秒単位で返します。

読み取りと書き込みの 1 秒あたりの平均数を求めるには、読み取り数と書き込み数をノードの稼働時間 (秒数) で割ります。その後、読み取り数については整合性レベルで割り、書き込み数についてはレプリケーション係数で割って求めます。これらの計算は、次の式で表すことができます。

1 秒あたりの平均読み取り数の式:

```
((number of reads * number of nodes in cluster) / read consistency quorum
 (2)) / uptime
```

1 秒あたりの平均書き込み数の式:

```
((number of writes * number of nodes in cluster) / replication factor of 3) /
 uptime
```

クラスターに 12 のノードがあり、4 週間稼働していると想定しましょう。nodetool info が返した稼働時間は 2,419,200 秒、nodetool tablestats が返した書き込み数は 10 億件、読み取り数は 20 億件でした。この例の場合、計算式は次のとおりです。

```
((2 billion reads * 12 in cluster) / read consistency quorum (2)) / 2,419,200
 seconds
=  12 billion reads / 2,419,200 seconds
=  4,960 read request per second
                        ((1 billion writes * 12 in cluster) / replication
 factor of 3) / 2,419,200 seconds
=  4 billion writes / 2,419,200 seconds
=  1,653 write request per second
```

4. テーブルのキャパシティ使用率を調べる

   平均キャパシティ使用率を推定するには、まず、Cassandra ソーステーブルの平均リクエストレートと平均行サイズを確認します。

   Amazon Keyspaces では、読み取りキャパシティユニット (RCU) と書き込みキャパシティユニット (WCU) に基づいて、テーブルの読み取りと書き込み用にプロビジョニングされたスループットキャパシティを測定します。この推定では、移行後の新しい Amazon Keyspaces テーブルの読み取りキャパシティと書き込みキャパシティのニーズを、これらのユニットに基づいて算出します。

このトピックの後半で、プロビジョンドキャパシティモードとオンデマンドキャパシティモードのどちらを選択すると、請求にどのような影響があるかを検討します。ただし、この例でキャパシティ使用率を推定するにあたっては、テーブルがプロビジョンドモードであると想定します。

- 読み取り – 1 RCU で、4 KB までの行に対して、LOCAL_QUORUM の読み取りリクエストを 1 回、または LOCAL_ONE の読み取りリクエストを 2 回実行できます。4 KB より大きい行を読み取る必要がある場合、読み取りオペレーションには追加の RCU が使用されます。必要な RCU の総数は、行のサイズと、必要な読み取り整合性 (LOCAL_QUORUM または LOCAL_ONE) によって異なります。

  例えば、8 KB の行を読み取るには、読み取り整合性が LOCAL_QUORUM の場合は 2 RCU、読み取り整合性が LOCAL_ONE の場合は 1 RCU が必要です。
- 書き込み – 1 WCU で、1 KB までの行に対して書き込みを 1 回実行できます。すべての書き込みでは LOCAL_QUORUM 整合性が使用されており、軽量トランザクション (LWT) の使用には追加料金はかかりません。

  必要な WCU の総数は、行サイズに応じて異なります。1 KB より大きい行を書き込む必要がある場合、書き込みオペレーションでは追加の WCU が使用されます。例えば、行のサイズが 2 KB の場合、書き込みリクエストを 1 回実行するには 2 WCU が必要です。

次の式を使用して、必要な RCU と WCU を推定できます。

- 読み取りキャパシティ (RCU 単位) を求めるには、1 秒あたりの読み取り数に、1 回で読み取る行数を掛け、その結果に、平均行サイズを 4 KB で割って最も近い整数に切り上げた結果を掛け合わせます。
- 書き込みキャパシティ (WCU 単位) を求めるには、リクエスト数に、平均行サイズを 1 KB で割って最も近い整数に切り上げた数を掛け合わせます。

これは、次の式で表されます。

```
Read requests per second * ROUNDUP((Average Row Size)/4096 per unit) =  RCUs per
 second

Write requests per second * ROUNDUP(Average Row Size/1024 per unit) = WCUs per
 second
```

例えば、行のサイズが 2.5 KB の Cassandra テーブルで 4,960 件の読み取りリクエストを実行している場合、Amazon Keyspaces では 4,960 RCU が必要です。行のサイズが 2.5 KB の Cassandra テーブルで 1 秒あたり 1,653 件の書き込みリクエストを実行している場合、Amazon Keyspaces では 1 秒あたり 4,959 WCU が必要になります。

この例は、次の式で表されます。

```
4,960 read requests per second * ROUNDUP( 2.5KB /4KB bytes per unit)
= 4,960 read requests per second * 1 RCU
= 4,960 RCUs

1,653 write requests per second * ROUNDUP(2.5KB/1KB per unit)
= 1,653 requests per second * 3 WCUs
= 4,959 WCUs
```

eventual consistency を使用すれば、読み取りリクエストごとのスループットキャパシティを最大で半減させることができます。結果整合性のある読み込みでは、1 件あたり最大で 8 KB を処理することができます。結果整合性のある読み込みの場合は、次の式に示すとおり、前の計算結果に 0.5 を掛けて求めることができます。

```
4,960 read requests per second * ROUNDUP( 2.5KB /4KB per unit) * .5
= 2,480 read request per second * 1 RCU
= 2,480 RCUs
```

5. Amazon Keyspaces の月額の推定利用料金を計算する

読み取り/書き込みのキャパシティスループットに基づいてテーブルの月額の請求料金を推定するには、オンデマンドモードとプロビジョンドモードの料金を別々の式で求め、テーブルで利用できる選択肢を比較できます。

プロビジョンドモード – 読み取りおよび書き込みのキャパシティ消費量は、1 秒あたりのキャパシティユニットに基づいて時間単位で請求されます。まず、そのレートを 0.7 で割り、自動スケーリングのデフォルトのターゲット使用率である 70% 相当分を求めます。次に、暦日の 30、1 日の時間数 24、そしてリージョン別料金を掛け合わせます。

この計算をまとめた式は、次のとおりです。

```
(read capacity per second / .7) * 24 hours * 30 days * regional rate
```

```
                (write capacity per second / .7) * 24 hours * 30 days * regional
 rate
```

オンデマンドモード – 読み取りキャパシティと書き込みキャパシティは、リクエストレート単位で請求されます。まず、リクエストレートに暦日の 30、1 日の時間数 24 を掛けます。次に、100 万リクエストユニットで割ります。最後に、リージョン別料金を掛けます。

この計算をまとめた式は、次のとおりです。

```
((read capacity per second * 30 * 24 * 60 * 60) / 1 Million read request units) *
 regional rate
                ((write capacity per second * 30 * 24 * 60 * 60) / 1 Million write
 request units) * regional rate
```

移行戦略を選択する

Apache Cassandra から Amazon Keyspaces に移行する場合は、次のいずれかの移行戦略を選択できます。

- オンライン – ライブで移行します。デュアル書き込みを使用して、Amazon Keyspaces と Cassandra クラスターに同時に新しいデータの書き込みを始めます。アプリケーションでゼロダウンタイムの移行と書き込み後の読み取り整合性が必要な場合は、この移行手法が推奨されます。

  オンライン移行の戦略を計画し、実装する方法の詳細については、「the section called “オンライン移行”」を参照してください。
- オフライン – ダウンタイム期間を設けて、その間に Cassandra から Amazon Keyspaces にデータセットをコピーします。オフライン移行では、アプリケーションの変更や、履歴データと新しい書き込み間の競合解決が不要なため、移行プロセスを簡素化できます。

  オフライン移行の計画方法の詳細については、「the section called “オフライン移行”」を参照してください。
- ハイブリッド – ほぼリアルタイムで変更を Amazon Keyspaces にレプリケートできますが、書き込み後の読み取り整合性は保証されません。

  ハイブリッド移行の計画方法の詳細については、「the section called “ハイブリッド移行”」を参照してください。

このトピックで説明した移行手法とベストプラクティスを確認したら、取り得る選択肢をディシジョンツリーに書き込み、独自の要件と利用可能なリソースに基づいて移行戦略を立てることができます。

Amazon Keyspaces へのオンライン移行: 戦略とベストプラクティス

Apache Cassandra から Amazon Keyspaces への移行中もアプリケーションの可用性を維持する必要がある場合は、このトピックで説明する主要コンポーネントを実装して、カスタムのオンライン移行戦略を準備できます。オンライン移行に関するこれらのベストプラクティスに従うことで、移行プロセス全体でアプリケーションの可用性と書き込み後の読み取り整合性を維持し、ユーザーへの影響を最小限に抑えることができます。

Apache Cassandra から Amazon Keyspaces へのオンライン移行の戦略を立てる際は、以下の主なステップについて検討する必要があります。

1. 新しいデータの書き込み
   - Amazon Keyspaces 移行用の ZDM デュアル書き込みプロキシ – Github で利用可能な ZDM デュアル書き込みプロキシを使用して、Apache Cassandra から Amazon Keyspaces へのダウンタイムのない移行を実行します。ZDM Proxy は、既存のアプリケーションをリファクタリングすることなくデュアル書き込みを実行し、クエリ検証のためにデュアル読み取りを実行します。
   - アプリケーションのデュアル書き込み: 既存の Cassandra クライアントライブラリとドライバーを使用して、アプリケーションにデュアル書き込みを実装できます。1 つのデータベースをリーダー、もう 1 つをフォロワーとして指定します。フォロワーデータベースへの書き込み失敗は、分析用としてデッドレターキュー (DLQ) に記録されます。
   - メッセージング層のデュアル書き込み: または、既存のメッセージングプラットフォームで追加のコンシューマーを使用して、Cassandra と Amazon Keyspaces の両方に書き込みを送信するように設定できます。最終的には、両方のデータベース間で一貫したビューが作成されます。
2. 履歴データの移行
   - 履歴データのコピー: AWS Glue またはカスタムの抽出、変換、ロード (ETL) スクリプトを使用して、Cassandra から Amazon Keyspaces に履歴データを移行できます。デュアル書き込みや一括ロードの間に生じる競合は、軽量トランザクションやタイムスタンプなどの手法を用いて解決します。
   - Time-To-Live (TTL) の使用: データ保持期間が短い場合は、Cassandra と Amazon Keyspaces の両方で TTL を使用して、不要な履歴データのアップロードを防ぐことができます。古いデータは Cassandra で期限切れになり、デュアル書き込みで新しいデータが書き込まれるため、最終的には Amazon Keyspaces が追いつきます。

3. データの検証
   - デュアル読み取り: Cassandra (プライマリ) データベースと Amazon Keyspaces (セカンダリ) データベースの両方からのデュアル読み取りを実装し、結果を非同期的に比較します。差分はログに記録されるか、DLQ に送信されます。
   - サンプル読み取り: Λ 関数を使用して、両方のシステムから定期的にデータをサンプリングして比較し、不一致があれば DLQ に記録します。
4. アプリケーションの移行
   - ブルー/グリーン戦略: Amazon Keyspaces をプライマリ、Cassandra をセカンダリのデータストアとして扱うように、アプリケーションを一度に切り替えます。パフォーマンスをモニタリングして、問題が発生した場合はロールバックします。
   - カナリアデプロイ: 最初は一部のユーザーのみを対象に移行を段階的に進め、Amazon Keyspaces をプライマリとするトラフィックを徐々に増やしていき、すべて移行したら完了です。
5. Cassandra の廃止

   アプリケーションが完全に Amazon Keyspaces に移行し、データ整合性が検証されたら、データ保持ポリシーに基づいて Cassandra クラスターの廃止を計画できます。

上記のコンポーネントを取り入れてオンライン移行の戦略を立てることで、ダウンタイムや中断を最小限に抑えながら、フルマネージド型の Amazon Keyspaces サービスにスムーズに移行できます。以降のセクションでは、コンポーネントごとに詳しく検討します。

トピック

- オンライン移行中の新しいデータの書き込み
- オンライン移行中の履歴データのアップロード
- オンライン移行中のデータ整合性の検証
- オンライン移行中のアプリケーションの移行
- オンライン移行後の Cassandra の廃止

オンライン移行中の新しいデータの書き込み

オンライン移行計画の最初のステップは、アプリケーションが新たに書き込むデータが、既存の Cassandra クラスターと Amazon Keyspaces の両データベースに保存されるようにすることです。目標は、2 つのデータストア間のビューの一貫性を確保することです。そのためには、新しい書き込

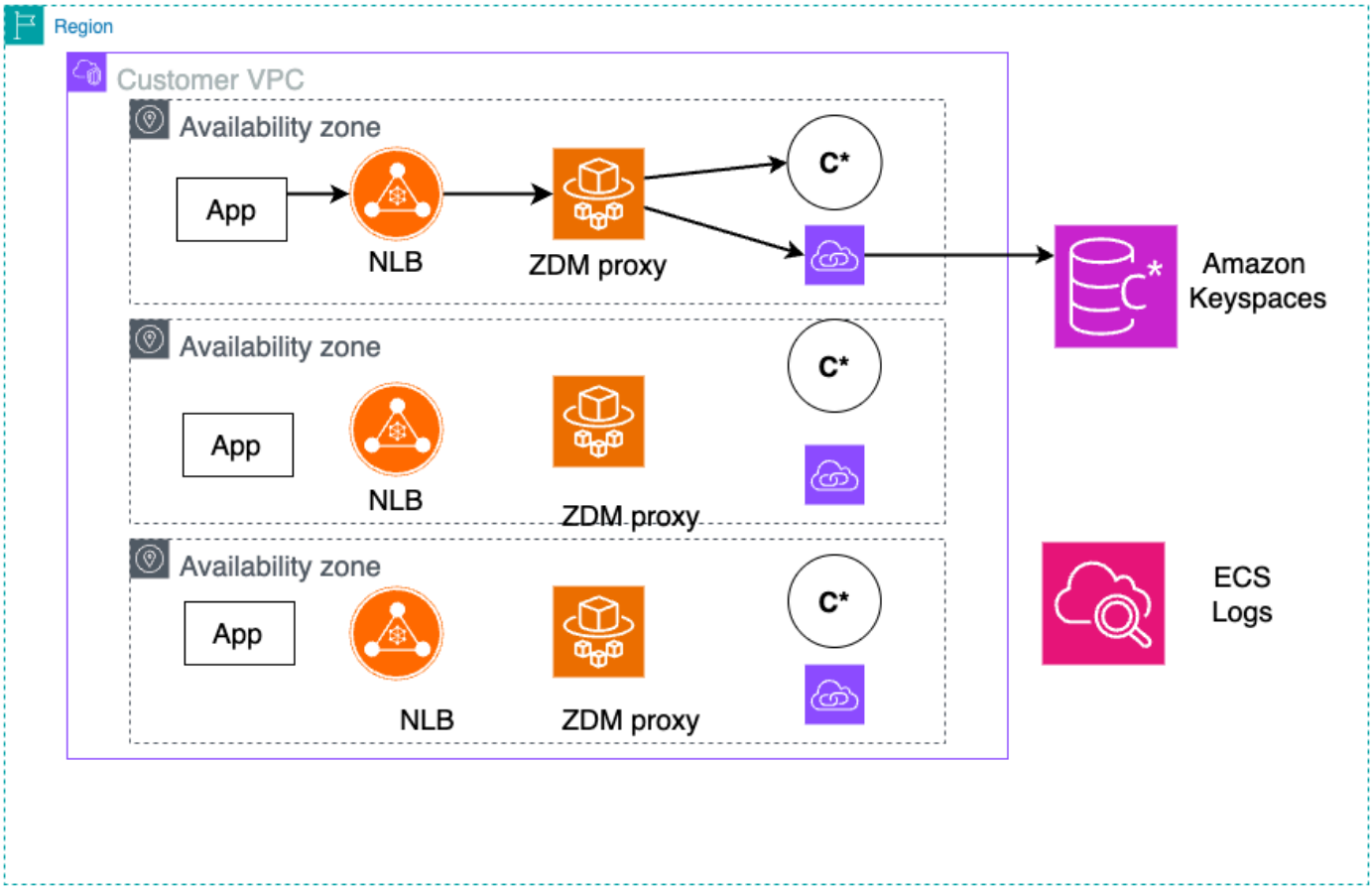
みをすべて両方のデータベースに適用します。デュアル書き込みを実装するには、次の 3 つのオプションのいずれかを検討してください。

- Amazon Keyspaces 移行用の ZDM デュアル書き込みプロキシ – Github で利用可能な Amazon Keyspaces 用の ZDM プロキシを使用すると、アプリケーションのダウンタイムなしで Apache Cassandra ワークロードを Amazon Keyspaces に移行できます。この拡張ソリューションは、AWS ベストプラクティスを実装し、公式の ZDM Proxy 機能を拡張します。
  - Apache Cassandra と Amazon Keyspaces の間でオンライン移行を実行します。
  - アプリケーションのリファクタリングを行わずに、ソーステーブルとターゲットテーブルの両方に同時にデータを書き込みます。
  - デュアル読み取りオペレーションを使用してクエリを検証します。

  このソリューションでは、 AWS と Amazon Keyspaces を操作するために以下の機能強化を提供しています。

  - コンテナデプロイ – VPC からアクセス可能なデプロイには、Amazon Elastic Container Registry (Amazon ECR) の事前設定された Docker イメージを使用します。
  - Infrastructure as Code – AWS CloudFormation テンプレートを使用してデプロイし、自動セットアップを行います AWS Fargate。
  - Amazon Keyspaces の互換性 – Amazon Keyspaces のカスタム適応を使用してシステムテーブルにアクセスします。

  このソリューションは、Fargate を使用して Amazon ECS で実行され、ワークロードの需要に基づいてサーバーレスのスケーラビリティを提供します。Network Load Balancer は、受信アプリケーショントラフィックを複数の Amazon ECS タスクに分散して高可用性を実現します。



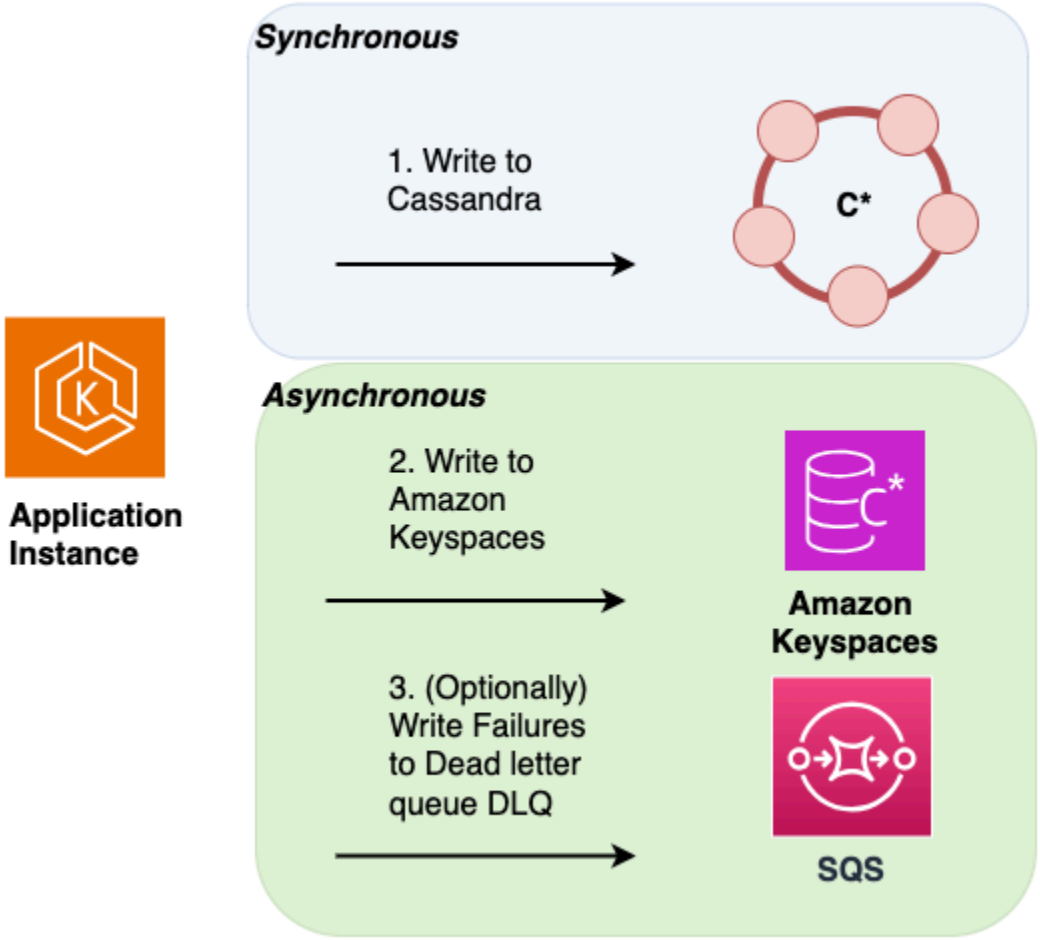
- アプリケーションのデュアル書き込み – 既存の Cassandra クライアントライブラリとドライバーを活用することで、アプリケーションコードの変更を最小限に抑えてデュアル書き込みを実装できます。既存のアプリケーションにデュアル書き込みを実装するか、アーキテクチャに新しいレイヤーを作成してデュアル書き込みを処理することができます。詳しい情報や、既存のアプリケーションにデュアル書き込みを実際したお客様の導入事例については、Cassandra 移行の導入事例を参照してください。

  デュアル書き込みを実装する場合は、1 つのデータベースをリーダー、もう 1 つのデータベースをフォロワーとして指定できます。これにより、元のソース (リーダーデータベース) への書き込みを続けることができ、フォロワー (宛先データベース) への書き込みが失敗しても、アプリケーションのクリティカルパスに支障をきたすことがありません。

  フォロワーへの書き込みが失敗した場合は再試行する代わりに、Amazon Simple Queue Service を使用して、失敗した書き込みをデッドレターキュー (DLQ) に記録できます。DLQ を使用して、フォロワーへの失敗した書き込みを分析し、宛先データベースで正常に処理されなかった理由を判断できます。

より高度なデュアル書き込み実装では、Saga パターンを使用して一連のローカルトランザクションを設計するための AWS ベストプラクティスに従うことができます。saga パターンでは、トランザクションが失敗した場合、補償トランザクションを実行して、以前のトランザクションによるデータベース変更を元に戻します。

オンライン移行でデュアル書き込みを使用する場合は、saga パターンに従ってデュアル書き込みを設定して、各書き込みをローカルトランザクションとして行い、異種データベースにまたがるオペレーションをアトミックに処理します。に推奨される設計パターンを使用した分散アプリケーションの設計の詳細については AWS クラウド、「クラウド設計パターン、アーキテクチャ、実装」を参照してください。



- メッセージング層のデュアル書き込み – アプリケーションレイヤーにデュアル書き込みを実装する代わりに、既存のメッセージング層を使用して、Cassandra と Amazon Keyspaces へのデュアル書き込みを実行できます。

そのためには、メッセージングプラットフォームにコンシューマーを追加して、両方のデータストアに書き込みを送信するように設定できます。このアプローチなら、メッセージング層を利用したシンプルなローコード戦略で、両方のデータベース間で結果整合性のある 2 つのビューを作成できます。

オンライン移行中の履歴データのアップロード

デュアル書き込みを実装して新しいデータを両方のデータストアにリアルタイムで書き込むようにした後、移行計画の次のステップでは、Cassandra から Amazon Keyspaces にコピーまたは一括アップロードする必要がある履歴データの量を評価します。これにより、アプリケーションの移行前に、新しいデータと履歴データの両方を新しい Amazon Keyspaces データベースに用意できます。

データ保持の要件 (組織のポリシーに基づいて保管しておく必要がある履歴データの量など) に応じて、次の 2 つの選択肢のいずれかを検討できます。

- 履歴データの一括アップロード – 既存の Cassandra デプロイから Amazon Keyspaces への履歴データの移行は、データの抽出、変換、ロード (ETL) に AWS Glue やカスタムスクリプトを使用するなど、さまざまな手法を通じて実現できます。 AWS Glue を使用して履歴データをアップロードする方法については、「the section called “オフライン移行”」を参照してください。

  履歴データの一括アップロードを計画する場合は、まさにアップロード中のデータを新しい書き込みが更新しようとした場合に生じる競合を、どのように解決するのか考えておく必要があります。一括アップロードは、最終的な一貫性が担保されます。つまり、データは最終的にすべてのノードに行き届きます。

  同じデータの更新作業が新しい書き込みによって同時に行われた場合、アップロードされた履歴データの方で上書きされないようにする必要があります。一括インポート中でもデータに最新の更新を確実に反映するには、一括アップロードのスクリプトかデュアル書き込みのアプリケーションロジックのいずれかに競合解決を織り込む必要があります。

  例えば、the section called “軽量トランザクション” (LWT) を使用してオペレーションを比較し、設定できます。そのためには、変更時間または状態を表すフィールドをデータモデルに追加します。

  さらに、Amazon Keyspaces は Cassandra の `WRITETIME` タイムスタンプ関数をサポートしています。Amazon Keyspaces のクライアント側タイムスタンプを使用して、ソースデータベースのタイムスタンプを維持し、Last-Writer-Wins (最後の書き込みを優先) による競合解決を実装できます。詳細については、「the section called “クライアントサイドのタイムスタンプ”」を参照してください。

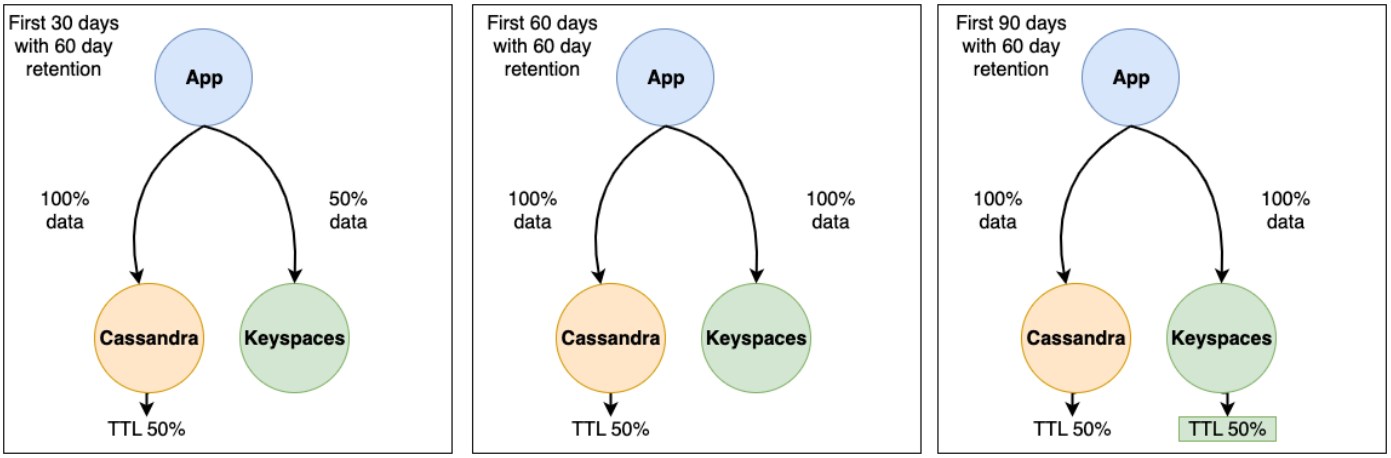
- Time-to-Live (TTL) の使用 – データ保持期間が 30、60、または 90 日未満の場合、移行中に Cassandra および Amazon Keyspaces で TTL を使用して、不要な履歴データを Amazon Keyspaces にアップロードしないようにすることができます。TTL に指定した期間が経過したら、データが自動的にデータベースから削除されます。

  移行フェーズでは、履歴データを Amazon Keyspaces にコピーする代わりに、TTL を設定して履歴データを古いシステム (Cassandra) で自動的に期限切れにし、デュアル書き込み手法を用いて新しい書き込みのみを Amazon Keyspaces に適用することができます。時間が経つにつれて、Cassandra クラスターで古いデータが順次期限切れになり、デュアル書き込み手法で新しいデータが書き込まれていき、自ずと Amazon Keyspaces が追いつき、Cassandra と同じデータが揃います。

  このアプローチなら、移行対象のデータ量を大幅に削減して、より効率的、合理的に移行プロセスを進めることができます。データ保持の要件がさまざまに異なる大規模なデータセットを処理する場合は、このアプローチを検討できます。TTL の詳細については、「the section called “TTL によるデータの期限切れ”」を参照してください。

  TTL によるデータの有効期限を利用して、Cassandra から Amazon Keyspaces に移行する次の例を検討してみましょう。この例では、両方のデータベースの TTL を 60 日間に設定し、90 日の期間にわたって移行プロセスがどのように進行するかを見ていきます。両方のデータベースにはこの期間中、デュアル書き込み手法で新しいデータが書き込まれます。移行を 30 日間ずつの各フェーズに分けて検討します。

  各フェーズで移行がどのように進むかを次の図にまとめています。



1. 最初の 30 日が過ぎた時点で、Cassandra クラスターと Amazon Keyspaces は新しい書き込みを受信しています。Cassandra クラスターには、60 日間の保持期間がまだ過ぎていない履歴データも含まれていて、これがクラスター内のデータの 50% を占めています。

60 日以上経ったデータは、TTL を使用して Cassandra クラスターから自動的に削除されます。この時点で、Amazon Keyspaces には Cassandra クラスターに保存されているデータの 50% が含まれており、これは新しい書き込みから履歴データを除いた分です。

2. 60 日後、Cassandra クラスターと Amazon Keyspaces の両方に、過去 60 日間に書き込まれた同じデータが揃います。
3. 90 日経過するまでの間に、Cassandra と Amazon Keyspaces の両方に同じデータが揃い、同じ速さでデータが期限切れになります。

この例で、TTL を使用して有効期限を 60 日に設定し、履歴データのアップロードを回避する方法がわかりました。

オンライン移行中のデータ整合性の検証

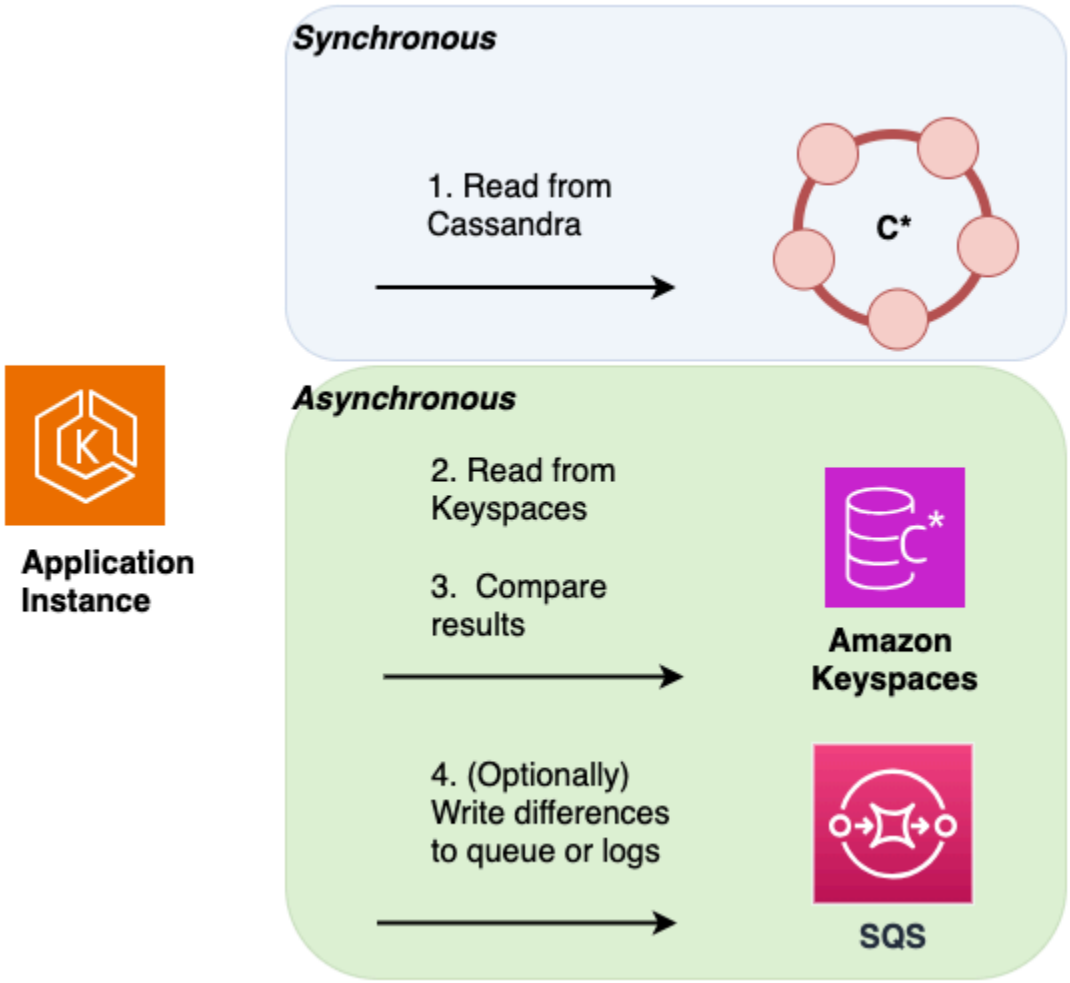
オンライン移行プロセスの次のステップは、データの検証です。デュアル書き込みで Amazon Keyspaces データベースに新しいデータを追加し、履歴データの移行は一括アップロード、または TTL によるデータの有効期限のいずれかを利用して完了しました。

これで、検証フェーズを使用して、両方のデータストアが実際に同じデータを含み、同じ読み取り結果を返すことを確認できます。次の 2 つの選択肢のいずれかを使用して、両方のデータベースに同じデータが含まれていることを検証できます。

- デュアル読み取り – ソースデータベースと宛先データベースの両方に、新しく書き込まれたデータと履歴データの同じセットが揃っていることを検証するために、デュアル読み取りを実装できます。それには、デュアル書き込み手法と同様に、プライマリ Cassandra データベースとセカンダリ Amazon Keyspaces データベースの両方からデータを読み取り、その結果を非同期的に比較します。

  プライマリデータベースの結果がクライアントに返され、セカンダリデータベースの結果がプライマリの結果セットに対して検証されます。検出された差分は、後で照合するためにログに記録するか、デッドレターキュー (DLQ) に送信できます。

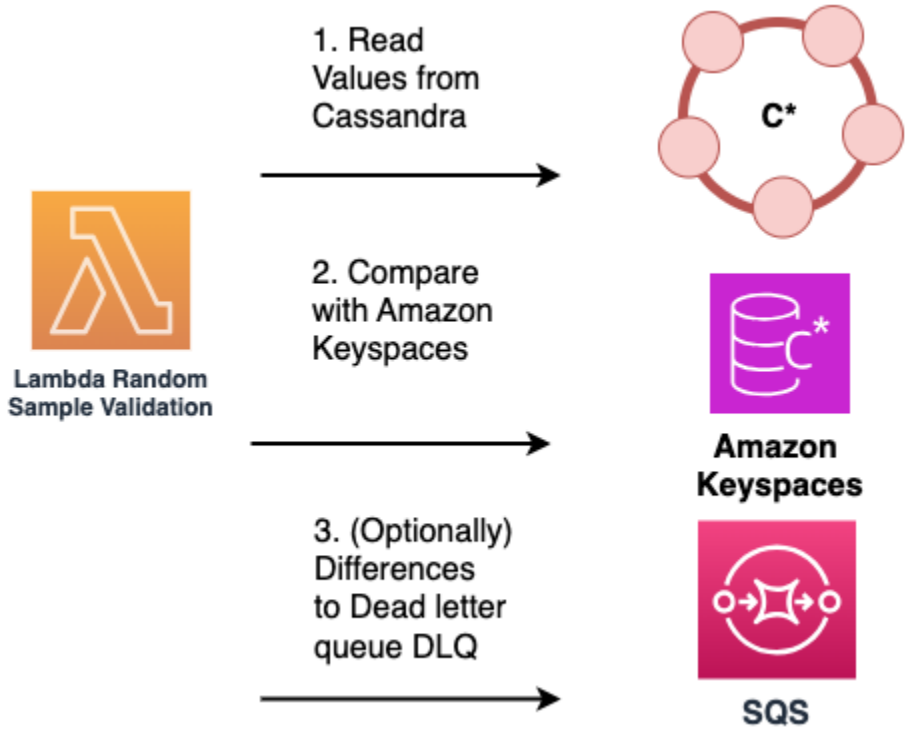
  次の図では、アプリケーションは Cassandra (プライマリデータストア) からの同期読み取りと、Amazon Keyspaces (セカンダリデータストア) からの非同期読み取りを実行しています。



- サンプル読み取り – アプリケーションコードの変更を必要としない代替ソリューションは、 AWS Lambda 関数を使用して、ソース Cassandra クラスターと宛先 Amazon Keyspaces データベースの両方から定期的かつランダムにデータをサンプリングすることです。

  これらの Lambda 関数は、定期的な間隔で実行するように設定できます。Lambda 関数は、ソースシステムと宛先システムの両方からデータのランダムなサブセットを取得し、そのサンプルデータの比較を実行します。2 つのデータセット間の不一致や不整合があった場合は、後で照合するために記録し、専用のデッドレターキュー (DLQ) に送信できます。

  次の図表は、このプロセスを示したものです。



オンライン移行中のアプリケーションの移行

オンライン移行の第 4 フェーズでは、アプリケーションを移行し、プライマリデータストアを Amazon Keyspaces に切り替えます。つまり、アプリケーションが読み書きを直接 Amazon Keyspaces との間で行うようになります。ユーザーへの中断を最小限に抑えるため、入念な計画を練り、関係各所と調整した上で進める必要があります。

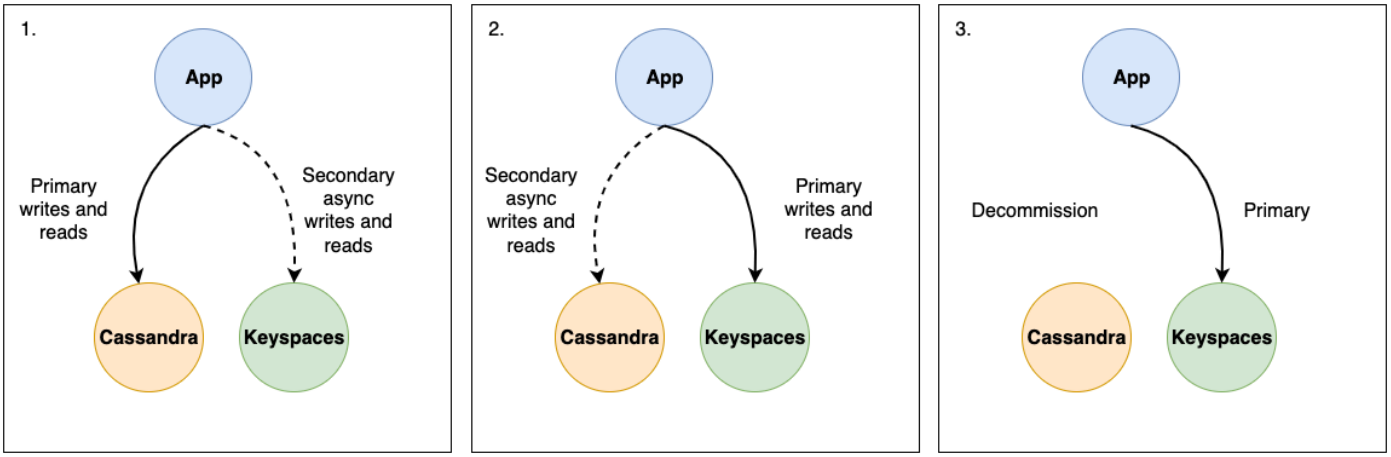
アプリケーションの移行には、ブルー/グリーンカットオーバー戦略とカナリアカットオーバー戦略という 2 つの異なる推奨ソリューションを使用できます。以下のセクションでは、これらの戦略について詳しく説明します。

- ブルー/グリーン戦略 – Amazon Keyspaces をプライマリデータストアとして、Cassandra をセカンダリデータストアとして扱うように、アプリケーションを一度に切り替えます。これを行うには、 AWS AppConfig 機能フラグを使用して、アプリケーションインスタンス全体のプライマリデータストアとセカンダリデータストアの選択を制御します。詳細については、「Creating a feature flag configuration profile in AWS AppConfig」を参照してください。

  Amazon Keyspaces をプライマリデータストアにした後、アプリケーションの動作とパフォーマンスをモニタリングして、Amazon Keyspaces が要件を満たしているか、移行が成功したかを確認します。

例えば、アプリケーションにデュアル読み取りを実装した場合は、アプリケーション移行フェーズの間に、プライマリの読み取りを Cassandra から Amazon Keyspaces に、セカンダリの読み取りを Amazon Keyspaces から Cassandra に切り替えます。切り替え後もモニタリングを続け、データ検証のセクションの説明どおりに結果を比較して、Cassandra を廃止する前に、両方のデータベース間で整合性が取れていることを確認します。

問題が検出された場合は、Cassandra をプライマリデータストアに戻すことで、以前の状態にすばやくロールバックできます。Amazon Keyspaces がプライマリデータストアとしてすべてのニーズを満たす場合に限り、移行の廃止フェーズに進みます。

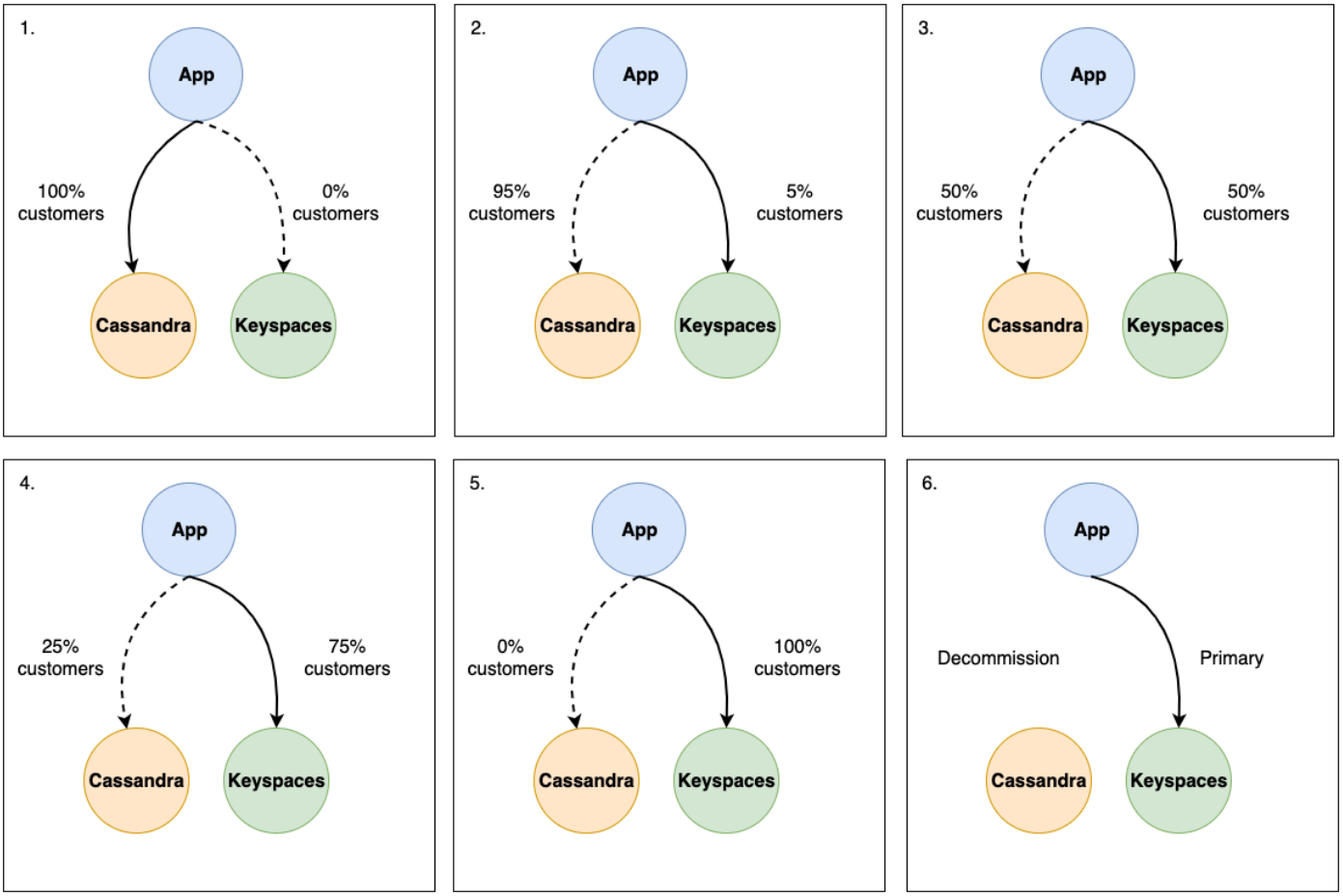


- カナリア戦略 – 一部のユーザーまたはトラフィックを対象に、移行を徐々にロールアウトします。最初は、アプリケーションのトラフィックのうち少量、例えば全体の 5% だけを Amazon Keyspaces をプライマリデータストアとして使用するバージョンにルーティングし、残りのトラフィックは引き続きプライマリデータストアとして Cassandra を使用します。

  これにより、実際のトラフィックで移行後のバージョンを徹底的にテストし、そのパフォーマンス、安定性を監視し、潜在的な問題を調査することができます。問題が検出されなければ、Amazon Keyspaces にルーティングするトラフィックの割合を徐々に増やし、最終的にすべてのユーザーとトラフィックが Amazon Keyspaces をプライマリデータストアとして使用するまで続けます。

  このように段階的にロールアウトしていくことで、サービス全体で障害が発生するリスクを最小限に抑え、移行プロセスを制御下に置くことができます。カナリアデプロイの最中に重大な問題が発生した場合は、影響を受けたトラフィックセグメントについて、プライマリデータストアとして Cassandra を使用する以前のバージョンにすばやくロールバックできます。Amazon Keyspaces が 100% のユーザーとトラフィックを正常に処理していることが検証されてはじめて、移行の廃止フェーズに進みます。

次の図は、カナリア戦略の各ステップを示しています。



オンライン移行後の Cassandra の廃止

アプリケーションの移行が完了し、アプリケーションが完全に Amazon Keyspaces で実行されるようになり、一定期間にわたるデータ整合性が検証されたら、Cassandra クラスターの廃止計画を立てることができます。このフェーズでは、Cassandra クラスターに残っているデータをアーカイブする必要があるか、削除してもよいかを評価できます。どちらにするかは、データ処理と保持に関する組織のポリシーによって決まります。

この戦略に従い、このトピックで説明した推奨ベストプラクティスを検討して Cassandra から Amazon Keyspaces へのオンライン移行を計画することで、アプリケーションの書き込み後の読み取り整合性や可用性を維持しながら、Amazon Keyspaces へのシームレスな移行を実現できます。

Apache Cassandra から Amazon Keyspaces への移行には、運用上のオーバーヘッドの削減、自動スケーリング、セキュリティの強化、コンプライアンス目標の達成を支援するフレームワークなど、数多くのメリットがあります。デュアル書き込み、履歴データのアップロード、データ検証、段階的

なロールアウトを踏まえてオンライン移行戦略を計画することで、アプリケーションとそのユーザーへの影響を最小限に抑えながら、スムーズな移行を実現できます。

このトピックで説明したオンライン移行戦略を実装すれば、移行の結果を検証し、問題があれば特定して対処し、最終的には既存の Cassandra デプロイを廃止して、フルマネージド型の Amazon Keyspaces サービスに完全移行できます。

オフライン移行プロセス: Apache Cassandra から Amazon Keyspaces への移行

オフライン移行は、移行時にダウンタイムを許容できる場合に適しています。企業では、パッチの適用や大規模リリース、またはハードウェアのアップグレードやメジャーアップグレードによるダウンタイムに備えて、メンテナンスウィンドウを設けることが一般的です。オフライン移行では、このウィンドウを利用してデータをコピーし、アプリケーショントラフィックを Apache Cassandra から Amazon Keyspaces に切り替えることができます。

オフライン移行の場合、Cassandra と Amazon Keyspaces の双方と同時に通信する必要がないため、アプリケーションの変更の手間を省けます。また、データフローを一時停止して、そのままの状態をコピーでき、途中変更の管理も不要です。

ここで紹介する例では、オフライン移行中のデータのステージングエリアとして Amazon Simple Storage Service (Amazon S3) を活用し、ダウンタイムを最小限に抑えます。Spark Cassandra コネクタと AWS Glueを使用して、Amazon S3 に Parquet 形式で保存されているデータを Amazon Keyspaces テーブルに自動的にインポートできます。この後のセクションでは、このプロセスの大筋を説明します。このプロセスのコード例は、Github で公開されています。

Amazon S3 を使用して Apache Cassandra から Amazon Keyspaces にオフラインで移行するには、次の AWS Glue ジョブ AWS Glue が必要です。

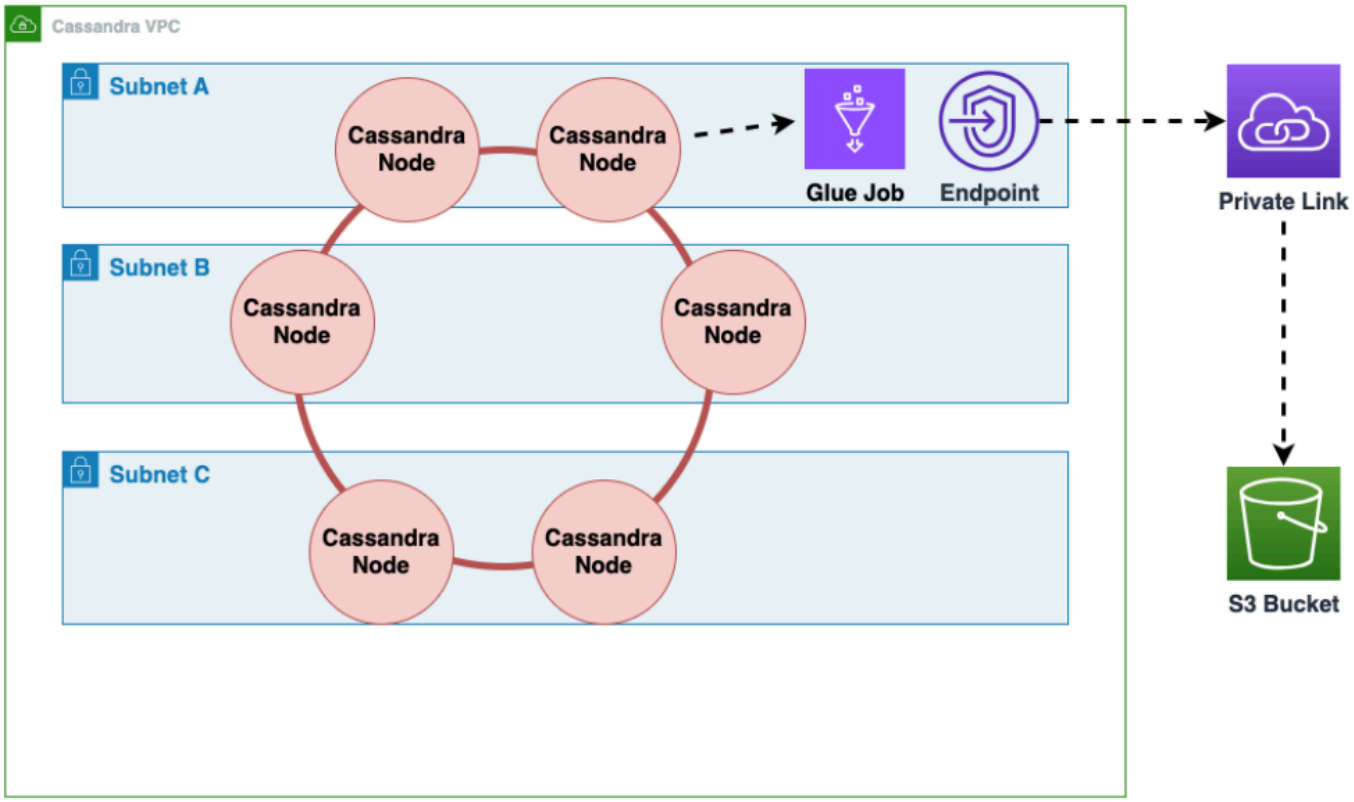
1. CQL データを抽出して変換し、Amazon S3 バケットに保存する ETL ジョブ。
2. バケットから Amazon Keyspaces にデータをインポートする 2 つ目のジョブ。
3. 増分データをインポートする 3 つ目のジョブ。

Amazon Virtual Private Cloud の Amazon EC2 で実行されている Cassandra から Amazon Keyspaces へのオフライン移行の実行方法

1. まず AWS Glue 、 を使用して Cassandra から Parquet 形式でテーブルデータをエクスポートし、Amazon S3 バケットに保存します。Cassandra を実行している Amazon EC2 インスタンスが存在する VPC への AWS Glue コネクタを使用して AWS Glue ジョブを実行する必要があ

ります。その後、Amazon S3 プライベートエンドポイントを使用して、Amazon S3 バケットにデータを保存できます。

次の図は、これらの手順の流れを示しています。



2. Amazon S3 バケット内のデータをシャッフルして、データのランダム性を高めます。データを均等にインポートすれば、ターゲットテーブルでトラフィックをより分散させることができます。

   この手順は、パーティションが大きい (1000 行を超えるパーティション) Cassandra からデータをエクスポートして、Amazon Keyspaces に挿入する場合に、ホットキーのパターンを回避するために必要です。ホットキーの問題が生じると、Amazon Keyspaces で `WriteThrottleEvents` が発生し、ロード時間が長引きます。

3. 別の AWS Glue ジョブを使用して、Amazon S3 バケットから Amazon Keyspaces にデータをインポートします。シャッフル後のデータは Amazon S3 バケット内に Parquet 形式で保存されます。

オフライン移行プロセスの詳細については、 を使用した Amazon Keyspaces AWS Glue ワークショップを参照してください。

ハイブリッド移行ソリューションの使用: Apache Cassandra から Amazon Keyspaces への移行

以下に紹介する移行ソリューションは、オンライン移行とオフライン移行を組み合わせたハイブリッドソリューションと言えます。このハイブリッド手法では、データがほぼリアルタイムで宛先のデータベースに書き込まれますが、書き込み後の読み取り整合性は保証されません。つまり、新たに書き込まれたデータをすぐには利用できず、遅延が想定されます。書き込み後の読み取り整合性が必要な場合は、「the section called “オンライン移行”」を参照してください。

Apache Cassandra から Amazon Keyspaces にほぼリアルタイムで移行する場合は、次の 2 つの方法から選択できます。

- CQLReplicator – (推奨) CQLReplicator はオープンソースのユーティリティで、Github から入手できます。このユーティリティを利用して、Apache Cassandra から Amazon Keyspaces にほぼリアルタイムでデータを移行できます。

  宛先データベースに伝達する書き込みと更新を決定するために、CQLReplicator は Apache Cassandra トークン範囲をスキャンし、 AWS Glue ジョブを使用して重複イベントを削除し、書き込みと更新を Amazon Keyspaces に直接適用します。
- 変更データキャプチャ (CDC) – Cassandra CDC をよくご存じの場合は、Apache Cassandra に組み込まれている CDC 機能でコミットログを個別の CDC ディレクトリにコピーし、変更を追跡できるため、ハイブリッド移行を実装するための 2 つ目の選択肢となるでしょう。

その場合は、データ変更を Amazon Keyspaces にレプリケートすることで、データ移行シナリオの別の選択肢として CDC を活用できます。

書き込み後の読み取り整合性が不要な場合は、CQLReplicator または CDC パイプラインを使用して、各ソリューションで AWS のサービス 使用されているツールと習熟度に基づいて、Apache Cassandra から Amazon Keyspaces にデータを移行できます。これらの方法でデータをほぼリアルタイムで移行することは、オンライン移行に代わるハイブリッドな移行手法と見なすことができます。

この戦略がハイブリッドな手法だと言えるのは、このトピックで概説している選択肢に加えて、過去のデータのコピーや、オンライン移行のトピックで解説するアプリケーション移行戦略など、オンラインで移行を進めるためのステップを一部実装する必要があるためです。

以下のセクションでは、これらのハイブリッド移行の選択肢について詳説します。

トピック

- CQLReplicator を使用してデータを移行する
- 変更データキャプチャ (CDC) を使用してデータを移行する

CQLReplicator を使用してデータを移行する

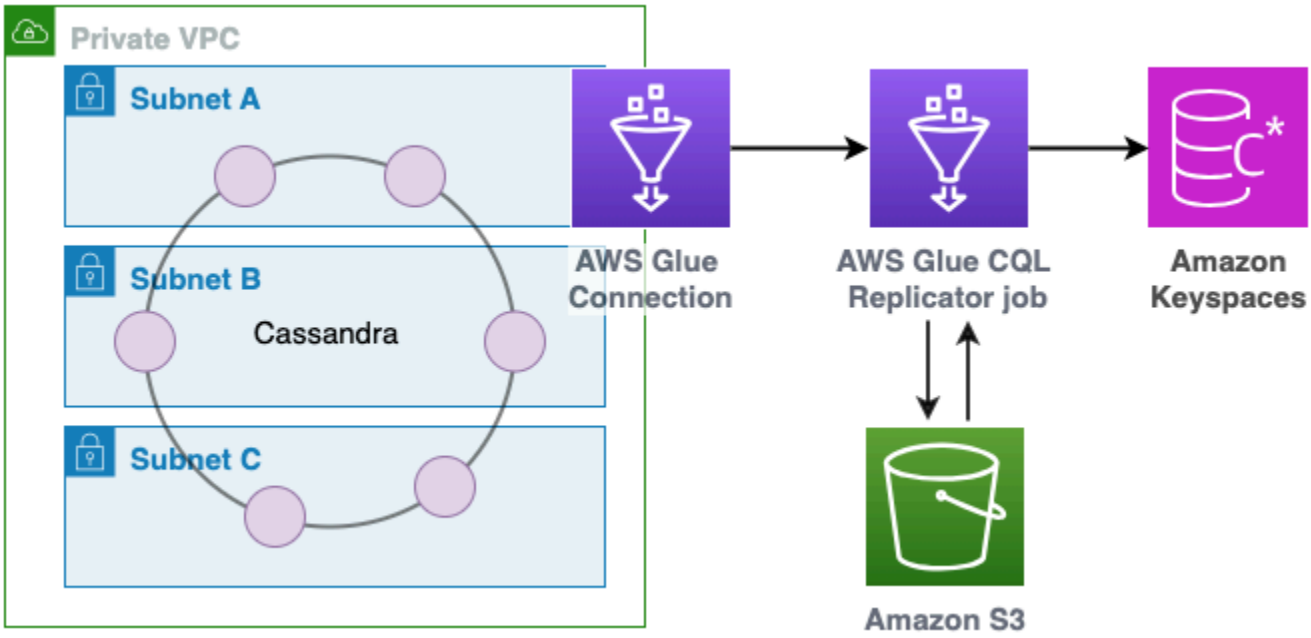
CQLReplicator を使用すると、CQL クエリを使用して Cassandra トークンリングをインテリジェントにスキャンすることで、Apache Cassandra からほぼリアルタイムでデータを読み取ることができます。CQLReplicator は Cassandra CDC を使用しません。その代わりにキャッシュ戦略を実装して、フルスキャンによるパフォーマンスへの影響を軽減しています。

宛先への書き込み数を減らすために、重複するレプリケーションイベントは自動的に削除されます。CQLReplicator を使用すると、ソースデータベースから宛先データベースへの変更のレプリケーションを調整できるため、Apache Cassandra から Amazon Keyspaces にほぼリアルタイムでデータを移行できます。

次の図は、 AWS Glueを使用した CQLReplicator ジョブの一般的なアーキテクチャを示しています。

1. プライベート VPC で実行されている Apache Cassandra へのアクセスを許可するには、 AWS Glue 接続タイプ Network を使用して接続を設定します。

2. CQLReplicator ジョブで重複を削除してキーのキャッシュを有効にするには、Amazon Simple Storage Service (Amazon S3) を設定します。
3. CQLReplicator ジョブが、ソースデータベースの検証済みの変更内容を Amazon Keyspaces に直接ストリーミングします。



CQLReplicator を使用した移行プロセスの詳細については、 AWS データベースブログの「CQLReplicator を使用して Cassandra ワークロードを Amazon Keyspaces に移行する」および AWS 「 を使用して Apache Cassandra ワークロードを Amazon Keyspaces に移行する AWS Glue」の記事を参照してください。

変更データキャプチャ (CDC) を使用してデータを移行する

Debezium での変更データキャプチャ (CDC) パイプラインの設定に慣れている場合は、CQLReplicator の代わりにこの方法を選んで Amazon Keyspaces にデータを移行できます。Debezium は、CDC 用のオープンソースの分散型プラットフォームです。データベースを監視し、行レベルの変更を確実に捉えるように設計されています。

Apache Cassandra 用の Debezium コネクタが Amazon Managed Streaming for Apache Kafka (Amazon MSK) に変更をアップロードし、その変更をダウンストリームのコンシューマーが使用および処理できるようになり、最終的にそれらのコンシューマーが Amazon Keyspaces にデータを書き込みます。詳細については、「Guidance for continuous data migration from Apache Cassandra to Amazon Keyspaces」を参照してください。

データ整合性の潜在的な問題に対処するために、コンシューマーが Cassandra と Amazon Keyspaces でキーやパーティションを比較するプロセスを Amazon MSK で実装できます。

このソリューションを問題なく実装できるように、次の点を検討することをお勧めします。

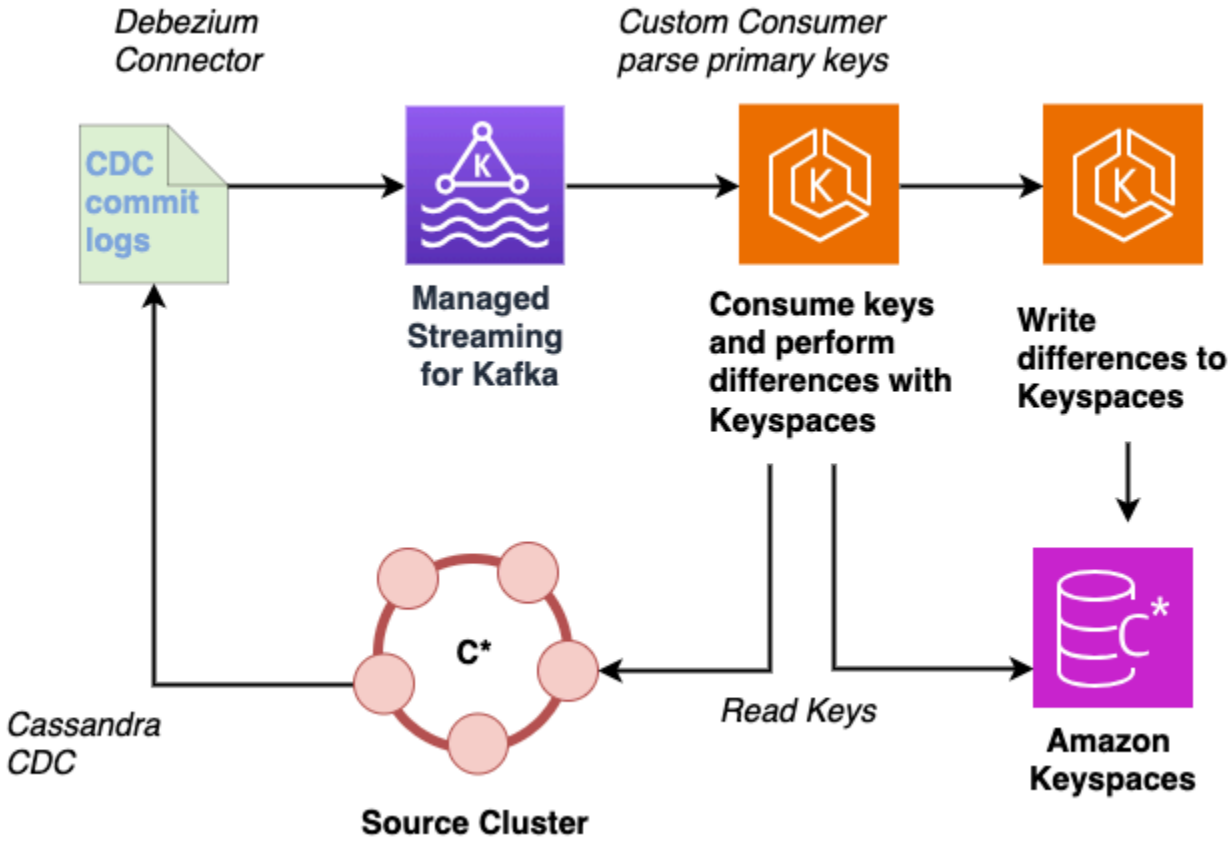
- CDC コミットログを解析する方法 (重複イベントを削除する方法など)。
- CDC ディレクトリを維持する方法 (古いログを削除する方法など)。
- Apache Cassandra で部分的な障害に対処する方法 (例えば、書き込みが 3 つのレプリカの 1 つでのみ成功した場合など)。
- リソースの割り当てを処理する方法 (ノードで発生する CDC プロセス用の CPU、メモリ、ディスク、IO の追加要件を考慮して、インスタンスのサイズを増やすなど)。

このパターンでは、Cassandra からの変更を、キーが以前の状態から変更された可能性があるという「ヒント」として扱います。宛先データベースに伝達すべき変更があるかを判断するには、まずソースとなる Cassandra クラスターから `LOCAL_QUORUM` オペレーションを使用して最新のレコードを取得する必要があり、その後、そのレコードを Amazon Keyspaces に書き込みます。

範囲指定の削除や更新の場合、パーティション全体との比較を実行しないと、宛先データベースに書き込む必要がある書き込みイベントや更新イベントを判断できない可能性があります。

書き込みがべき等でない場合は、Amazon Keyspaces に書き込む前に、書き込み内容を宛先データベースの既存データと比較する必要も生じます。

次の図は、Debezium と Amazon MSK を使用した CDC パイプラインの一般的なアーキテクチャを示しています。



Amazon Keyspaces へのデータの一括アップロードまたは移行に適したツールを選択する方法

このセクションでは、Amazon Keyspaces にデータを一括アップロードまたは移行するために使用できるさまざまなツールを確認し、ニーズに応じて適切なツールを選定する方法を学びます。さらに、このセクションでは、Amazon Keyspaces にデータをインポートする方法を示す、利用可能なstep-by-stepのチュートリアルの概要とユースケースについて説明します。

Apache Cassandra から Amazon Keyspaces にワークロードを移行するための戦略を確認するには、「the section called “Cassandra からの移行”」を参照してください。

- 移行ツール
  - Github で利用可能な Amazon Keyspaces (Apache Cassandra 向け) の料金計算ツールを使用すると、既存の Apache Cassandra ワークロードに基づいて Amazon Keyspaces の月額コストを見積もることができます。Amazon Keyspaces の Cassandra nodetool ステータス出力と意図したサーバーレス設定からメトリクスを入力して、2 つのソリューション間の直接コストを比較します。この計算ツールは、既存の Cassandra デプロイと比較した Amazon Keyspaces の運用コ

ストにのみ重点を置いていることに注意してください。インフラストラクチャのメンテナンス、運用オーバーヘッド、Cassandra のサポートコストなどの総所有コスト (TCO) 要因は含まれません。

- Amazon Keyspaces 移行用の ZDM デュアル書き込みプロキシ – Github で利用可能な ZDM デュアル書き込みプロキシは、Apache Cassandra から Amazon Keyspaces へのダウンタイムのない移行をサポートしています。
- CQLReplicator – CQLReplicator はオープンソースのユーティリティで、Github から入手できます。このユーティリティを利用して、Apache Cassandra から Amazon Keyspaces にほぼリアルタイムでデータを移行できます。

  詳細については、「the section called “CQLReplicator”」を参照してください。
- Amazon Managed Streaming for Apache Kafka を使用して、デュアル書き込みによるオンライン移行プロセスを実装する方法の詳細については、「Guidance for continuous data migration from Apache Cassandra to Amazon Keyspaces」を参照してください。
- 大規模な移行については、抽出、変換、ロード (ETL) ツールの使用を検討してください。 AWS Glue を使用すると、データ変換移行を迅速かつ効果的に実行できます。詳細については、「the section called “オフライン移行”」を参照してください。
- Apache Cassandra Spark コネクタを使用して Amazon Keyspaces にデータを書き込む方法については、「the section called “Apache Spark との統合”」を参照してください。
- cqlsh `COPY FROM` コマンドを使用してデータを Amazon Keyspaces にロードすることから始めます。cqlsh は、Apache Cassandra に含まれており、小さなデータセットまたはテストデータのロードに最適です。手順については、「the section called “cqlsh を使用したデータのロード”」を参照してください。
- また、Apache Cassandra 用の DataStax Bulk Loader を使用すれば、`dsbulk` コマンドを使用して Amazon Keyspaces にデータをロードできます。DSBulk は、インポート機能が cqlsh よりも強力で、GitHub リポジトリから入手できます。手順については、「the section called “DSBulk を使用したデータのロード”」を参照してください。

Amazon Keyspaces へのデータアップロードに関する一般的な考慮事項

- データを小さな単位に分けてアップロードします。

  生データサイズの観点から、次の移行単位とその潜在的なフットプリントを考慮してください。1 つまたは複数のフェーズで少量ずつデータをアップロードすれば、移行を簡単に進めることができます。

  - クラスター別 — すべての Cassandra データを一度に移行します。このアプローチはクラスターが小さければおそらく問題ありません。
  - キースペース別またはテーブル別 - 移行をキースペースまたはテーブルのグループに分割します。このアプローチは、各ワークロードの要件に基づいてフェーズでデータを移行する場合に役立ちます。
  - データ別 - データサイズを縮小するために、特定のユーザーまたは製品グループのデータを移行することを検討します。
- 最初に優先的にアップロードするデータを処理のしやすさに基づいて決めます。

  特定の時間に変更されないデータ、夜間のバッチジョブのデータ、オフライン時間に使用されないデータ、内部アプリのデータなど、最初に簡単に移行できるデータがあるかどうかを検討します。

トピック

- チュートリアル: cqlsh を使用した Amazon Keyspaces へのデータのロード
- チュートリアル: DSBulk を使用した Amazon Keyspaces へのデータのロード

チュートリアル: cqlsh を使用した Amazon Keyspaces へのデータのロード

このチュートリアルガイドでは、`cqlsh COPY FROM` コマンドを使用して Apache Cassandra から Amazon Keyspaces にデータを移行する手順を案内します。`cqlsh COPY FROM` コマンドは、学術的な目的やテスト用に小さなデータセットを Amazon Keyspaces に迅速かつ簡単にアップロードするのに便利です。本稼働ワークロードの移行方法の詳細については、「the section called “オフライン移行”」を参照してください。このチュートリアルでは、次の手順を実行します。

前提条件 – 認証情報を使用して AWS アカウントを設定し、証明書の JKS トラストストアファイルを作成し、Amazon Keyspaces に接続する`cqlsh`ように を設定します。

1. ソース CSV とターゲットテーブルの作成 – ソースデータとして CSV ファイルを準備し、Amazon Keyspaces でターゲットのキースペースとテーブルを作成します。
2. データの準備 – CSV ファイル内のデータをランダム化して分析し、行サイズの平均値と最大値を求めます。
3. スループットキャパシティの設定 – データサイズと必要なロード時間に基づいて必要な書き込みキャパシティユニット (WCU) を計算し、テーブルにプロビジョニングされるキャパシティを設定します。

4. cqlsh パラメータの設定 – ワークロードを均等に分散させるために、INGESTRATE、NUMPROCESSES、MAXBATCHSIZE、CHUNKSIZE などの cqlsh COPY FROM パラメータの最適値を求めます。
5. **cqlsh COPY FROM** コマンドの実行 – cqlsh COPY FROM コマンドを実行して、CSV ファイルから Amazon Keyspaces テーブルにデータをアップロードし、進行状況を監視します。

トラブルシューティング – 無効なリクエスト、パーサーエラー、キャパシティエラー、cqlsh エラーなど、データアップロードの処理中によく起きる問題を解決します。

トピック

- 前提条件: cqlsh COPY FROM を使用してデータをアップロードする前に完了する手順
- ステップ 1: データアップロード用のソース CSV ファイルとターゲットテーブルを作成する
- ステップ 2: データを正常にアップロードできるようにソースデータを準備する
- ステップ 3: テーブルのスループットキャパシティを設定する
- ステップ 4: cqlsh COPY FROM を設定する
- ステップ 5: cqlsh COPY FROM コマンドを実行して CSV ファイルからターゲットテーブルにデータをアップロードする
- トラブルシューティング

前提条件: **cqlsh COPY FROM** を使用してデータをアップロードする前に完了する手順

このチュートリアルを開始する前に、次のタスクを完了しておく必要があります。

1. まだサインアップしていない場合は、「」の手順に従って にサインアップ AWS アカウント しますthe section called “セットアップ AWS Identity and Access Management”。
2. the section called “サービス固有の認証情報の作成” のステップに従って、サービス固有の認証情報を作成します。
3. Cassandra クエリ言語シェル (cqlsh) 接続をセットアップし、the section called “cqlsh の使用” のステップに従って Amazon Keyspaces に接続できることを確認します。

ステップ 1: データアップロード用のソース CSV ファイルとターゲットテーブルを作成する

このチュートリアルでは、keyspaces_sample_table.csv という名前のカンマ区切り値 (CSV) ファイルをデータ移行用のソースファイルとして使用します。提供されたサンプルファイルには、book_awards という名前のテーブルに関する数行のデータが含まれています。

1. ソースファイルを作成します。次のオプションのいずれかを選択します。
   - 次のアーカイブファイル samplemigration.zip に含まれているサンプル CSV ファイル (keyspaces_sample_table.csv) をダウンロードします。アーカイブを解凍し、keyspaces_sample_table.csv へのパスをメモしておきます。
   - Apache Cassandra データベースに保存されている独自のデータを CSV ファイルに入力するには、次の例に示すように、cqlsh COPY TO ステートメントを使用してソース CSV ファイルに入力します。

     ```
     cqlsh localhost 9042 -u "username" -p "password" --execute
      "COPY mykeyspace.mytable TO 'keyspaces_sample_table.csv' WITH HEADER=true";
     ```

     作成する CSV ファイルが以下の要件を満たしていることを確認してください。

     - 最初の行に列名が含まれています。
     - ソース CSV ファイルの列名がターゲットテーブルの列名と一致してます。
     - データがカンマで区切られてます。
     - すべてのデータ値が有効な Amazon Keyspaces データ型です。「the section called “データ型”」を参照してください。
2. Amazon Keyspaces でターゲットのキースペースとテーブルを作成します。
   a. cqlsh を使用して Amazon Keyspaces に接続し、次の例のサービスエンドポイント、ユーザー名、およびパスワードをそれぞれ独自の値に置き換えます。

      ```
      cqlsh cassandra.us-east-2.amazonaws.com 9142 -u "111122223333" -
      p "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY" --ssl
      ```

   b. 次の例に示すように、catalog という名前の新しいキースペースを作成します。

      ```
      CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
      ```

c. 新しいキースペースが利用可能になったら、次のコードを使用してターゲットテーブル book_awards を作成します。

```
CREATE TABLE "catalog.book_awards" (
   year int,
   award text,
   rank int,
   category text,
   book_title text,
   author text,
   publisher text,
   PRIMARY KEY ((year, award), category, rank)
   );
```

Apache Cassandra が元のデータソースである場合、ヘッダーが一致している Amazon Keyspaces ターゲットテーブルを作成する簡単な方法は、次のステートメントに示すように、ソーステーブルから CREATE TABLE ステートメントを生成する方法です。

```
cqlsh localhost 9042  -u "username" -p "password" --execute "DESCRIBE
 TABLE mykeyspace.mytable;"
```

次に、Amazon Keyspaces で、列名とデータ型が Cassandra ソーステーブルの説明と一致しているターゲットテーブルを作成します。

ステップ 2: データを正常にアップロードできるようにソースデータを準備する

効率的な転送のためのソースデータの準備には、2 つのステップがあります。まず、データをランダム化します。第 2 のステップでは、データが正常にアップロードされるように、データを分析して、適切な cqlsh パラメータ値と必要なテーブル設定を判断します。

データをランダム化する

cqlsh COPY FROM コマンドは、CSV ファイルに表示される順序と同じ順序でデータの読み取りと書き込みを行います。cqlsh COPY TO コマンドを使用してソースファイルを作成すると、データはキーソートされた順序で CSV に書き込まれます。内部的には、Amazon Keyspaces でパーティションキーを使用してデータが分割されます。Amazon Keyspaces には、同一のパーティションキーに対するロードバランスリクエストに役立つロジックが内蔵されていますが、順序をランダム化すると、データのロードが高速かつ効率的になります。これは、Amazon Keyspaces で異なるパー

ティションにデータが書き込まれたときに発生する組み込みのロードバランシングを利用できるためです。

パーティション間で書き込みを均等に分散させるには、ソースファイル内のデータをランダム化する必要があります。アプリケーションを書き込んでこれを実行することができます。また、Shuf などのオープンソースツールを使用することもできます。Shuf は、Linux ディストリビューション、macOS (コアユーティリティを homebrew にインストールする)、Windows (Windows Subsystem for Linux (WSL) を使用する) で無料で使用できます。このステップで列名を含むヘッダー行がシャッフルされないようにするには、追加のステップが 1 つ必要です。

ヘッダーを維持した状態でソースファイルをランダム化するには、次のコードを入力します。

```
tail -n +2 keyspaces_sample_table.csv | shuf -o keyspace.table.csv && (head
 -1 keyspaces_sample_table.csv && cat keyspace.table.csv ) > keyspace.table.csv1 &&
 mv keyspace.table.csv1 keyspace.table.csv
```

Shuf により、`keyspace.table.csv` という新しい CSV ファイルにデータが書き換えられます。これで、不要になった `keyspaces_sample_table.csv` ファイルを削除できます。

データを分析する

データを分析して、平均行サイズと最大行サイズを決定します。

この作業を行う理由は次のとおりです。

- 転送されるデータの総量を見積もる場合に、平均行サイズが役立ちます。
- データのアップロードに必要な書き込みキャパシティをプロビジョニングする際には、平均行サイズが必要になります。
- 各行のサイズが 1 MB 未満 (Amazon Keyspaces の行サイズの上限) であることを確認できます。

> **Note**
>
> このクォータは、パーティションサイズではなく、行サイズを指します。Apache Cassandra のパーティションとは異なり、Amazon Keyspaces のパーティションのサイズは事実上無制限です。パーティションキーとクラスタリング列には、メタデータ用の追加のストレージが必要です。このストレージは行の raw サイズに追加する必要があります。詳細については、「the section called “行サイズの推定”」を参照してください。

次のコードは、AWK を使用して CSV ファイルを分析し、行の平均サイズと最大サイズを出力します。

```
awk -F, 'BEGIN {samp=10000;max=-1;}{if(NR>1){len=length($0);t+=len;avg=t/
NR;max=(len>max ? len : max)}}NR==samp{exit}END{printf("{lines: %d, average: %d bytes,
 max: %d bytes}\n",NR,avg,max);}' keyspace.table.csv
```

このコードを実行すると、次の出力が表示されます。

```
using 10,000 samples:
{lines: 10000, avg: 123 bytes, max: 225 bytes}
```

このチュートリアルの次のステップの平均行サイズを使用して、テーブルの書き込みキャパシティをプロビジョニングします。

ステップ 3: テーブルのスループットキャパシティを設定する

このチュートリアルでは、設定された時間範囲内でデータがロードされるように cqlsh を調整する方法を示します。事前に実行する読み取りと書き込みの数がわかっているので、プロビジョンドキャパシティモードを使用します。データ転送が完了したら、アプリケーションのトラフィックパターンに合わせてテーブルのキャパシティモードを設定する必要があります。キャパシティ管理の詳細については、「サーバーレスリソースの管理」を参照してください。

プロビジョンドキャパシティモードでは、事前にテーブルにプロビジョニングする読み取りキャパシティと書き込みキャパシティの量を指定します。書き込みキャパシティは時間ユニットで課金され、書き込みキャパシティユニット (WCU) で計測されます。各 WCU は、1 秒あたり 1 KB のデータの書き込みをサポートするのに十分な書き込みキャパシティです。データをロードする際に、書き込みレートが、ターゲットテーブルで設定した WCU の上限 (パラメータ: `write_capacity_units`) を超えないようにしてください。

デフォルトでは、1 つのテーブルに最大 40,000 の WCU を、アカウント内のすべてのテーブルに最大 80,000 の WCU をプロビジョニングすることができます。追加のキャパシティが必要な場合は、Service Quotas コンソールでクォータの増加をリクエストできます。クォータの詳細については、「クォータ」を参照してください。

挿入に必要な WCU の平均数を計算する

1 秒あたり 1 KB のデータを挿入するには、1 WCU が必要です。CSV ファイルに 360,000 の行があり、1 時間ですべてのデータをロードする場合は、1 秒あたり 100 行 (360,000 行 / 60 分 / 60 秒 = 100 行/秒) を書き込む必要があります。各行に最大 1 KB のデータがある場合、1 秒あたり 100 行

を挿入するには、100 WCU をテーブルにプロビジョニングする必要があります。各行に 1.5 KB のデータがある場合、1 秒あたり 1 行を挿入するには 2 WCU が必要です。したがって、1 秒あたり 100 行を挿入するには、200 WCU をプロビジョニングする必要があります。

1 秒あたり 1 行の挿入が必要な WCU 数を調べるには、平均行サイズ (バイト) を 1024 で割り、端数を切り上げて最も近い整数にします。

例えば、平均行サイズが 3000 バイトの場合、1 秒あたり 1 行を挿入するには 3 WCU が必要です。

```
ROUNDUP(3000 / 1024) = ROUNDUP(2.93) = 3 WCUs
```

データのロード時間とキャパシティを計算する

これで、CSV ファイルの平均サイズと平均行数が分かったので、特定の時間内にデータをロードする場合に必要な WCU 数と、さまざまな WCU 設定を使用して CSV ファイルにすべてのデータをロードするのにかかるおおよその時間を計算できます。

例えば、ファイルの各行が 1 KB で、CSV ファイルに 1,000,000 行がある場合、1 時間でデータをロードするには、その時間に少なくとも 278 WCU をテーブルにプロビジョニングする必要があります。

```
1,000,000 rows * 1 KBs = 1,000,000 KBs
1,000,000 KBs / 3600 seconds =277.8 KBs / second = 278 WCUs
```

プロビジョンドキャパシティを設定する

テーブルの書き込みキャパシティは、そのテーブルの作成時、または ALTER TABLE CQL コマンドを使用して、設定することができます。以下は、ALTER TABLE CQL ステートメントを使用してテーブルのプロビジョンキャパシティ設定に変更を加えるための構文です。

```
ALTER TABLE mykeyspace.mytable WITH custom_properties={'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 100,
 'write_capacity_units': 278}} ;
```

完全な言語リファレンスについては、「the section called “ALTER TABLE”」を参照してください。

ステップ 4: **cqlsh COPY FROM** を設定する

このセクションでは、cqlsh COPY FROM のパラメータ値を決定する方法について説明します。cqlsh COPY FROM コマンドは、前の手順で準備した CSV ファイルを読み取り、CQL を使用して Amazon Keyspaces にデータを挿入します。このコマンドにより、行の分割と一連のワーカー

間での INSERT オペレーションの配分が行われます。各ワーカーは Amazon Keyspaces との接続を確立し、このチャンネルに沿って INSERT リクエストを送信します。

cqlsh COPY コマンドには、ワーカー間でワークを均等に分配するための内部ロジックがありません。ただし、ワークが均等に分配されるように手動で設定することはできます。まず、次の主要な cqlsh パラメータを確認します。

- DELIMITER — カンマ以外の区切り文字を使用した場合はこのパラメータを設定できます。デフォルトはカンマです。
- INGESTRATE — cqlsh COPY FROM により処理が試行される 1 秒あたりのターゲット行数。設定しない場合のデフォルト値は 100,000 です。
- NUMPROCESSES — COPY FROM タスクに対して cqlsh により作成される子ワーカープロセスの数。この設定の最大値は 16 で、デフォルトは num_cores - 1 です。num_cores は cqlsh が実行されているホストの処理コア数です。
- MAXBATCHSIZE - バッチサイズにより、1 つのバッチで挿入先テーブルに挿入される最大行数が決まります。設定されていない場合、cqlsh により挿入行数が 20 行のバッチが使用されます。
- CHUNKSIZE — 子ワーカーに渡すワーク単位のサイズ。デフォルトでは、5,000 に設定されます。
- MAXATTEMPTS — 失敗したワーカーチャンクの再試行の最高回数。最高試行回数に達すると、失敗したレコードが新しい CSV ファイルに書き込まれ、後ほど、失敗を調査した上で再実行できます。

ターゲット送信先テーブルにプロビジョニングした WCU の数に基づいて INGESTRATE を設定します。cqlsh COPY FROM コマンドの INGESTRATE は制限ではなくターゲット平均です。これは、設定した数を大きく上回る可能性がある (多くの場合そうなる) ことを意味します。このような超過を許可し、データロードリクエストを処理できるだけの十分なキャパシティを確保するには、INGESTRATE をテーブルの書き込みキャパシティの 90% に設定します。

```
INGESTRATE = WCUs * .90
```

次に、NUMPROCESSES パラメータを、システムのコア数より 1 少ない値に設定します。システムのコア数を調べるには、次のコードを実行します。

```
python -c "import multiprocessing; print(multiprocessing.cpu_count())"
```

このチュートリアルでは、以下の値を使用します。

```
NUMPROCESSES = 4
```

各プロセスでワーカーが作成され、各ワーカーで Amazon Keyspaces への接続が確立されます。Amazon Keyspaces は、接続ごとに 1 秒あたり最大で 3,000 件の CQL リクエストに対応できます。つまり、各ワーカーの 1 秒あたりのリクエスト処理数が 3,000 件未満であるか確認する必要があるということです。

INGESTRATE と同様に、ワーカーは設定した数値を大幅に上回ることが多く、クロックの秒数に制限されません。したがって、大幅な超過を考慮しておくために、各ワーカーの 1 秒あたりの目標リクエスト処理数が 2,500 件になるように cqlsh パラメータを設定します。ワーカーに分配されるワーク量を計算するには、次のガイドラインを使用します。

- INGESTRATE を NUMPROCESSES で割ります。
- INGESTRATE / NUMPROCESSES > 2,500 になった場合は、この式が真になるように INGESTRATE を下げます。

```
INGESTRATE / NUMPROCESSES <= 2,500
```

サンプルデータのアップロードを最適化するための設定を行う前に、cqlsh のデフォルト設定を再確認し、その使用がデータのアップロードプロセスにどのように影響するのか見てみましょう。cqlsh COPY FROM では CHUNKSIZE を使用して膨大なワークが作成されて (INSERT ステートメント) ワーカーに分配されるので、ワークは自動的には均等に分配されません。INGESTRATE 設定によっては、アイドル状態になるワーカーもあります。

ワーカー間でワークを均等に分配し、各ワーカーに対して 1 秒あたりの最適なリクエスト数を 2,500 件で維持するには、入力パラメータを変更して CHUNKSIZE、MAXBATCHSIZE、INGESTRATE に設定する必要があります。データロード中のネットワークトラフィックの使用率を最適化するには、MAXBATCHSIZE の値として最大値の 30 に近い値を選択します。CHUNKSIZE を 100 に、MAXBATCHSIZE を 25 に変更すると、10,000 行が 4 つのワーカーに均等に分配されます (10,000 / 2500 = 4)、

次のコード例はこのことを示しています。

```
INGESTRATE = 10,000
NUMPROCESSES = 4
CHUNKSIZE = 100
MAXBATCHSIZE. = 25
```

```
Work Distribution:
Connection 1 / Worker 1 : 2,500 Requests per second
Connection 2 / Worker 2 : 2,500 Requests per second
Connection 3 / Worker 3 : 2,500 Requests per second
Connection 4 / Worker 4 : 2,500 Requests per second
```

要約するために、cqlsh COPY FROM パラメータの設定時に次の数式を使用します。

- INGESTRATE = write_capacity_units * .90
- NUMPROCESSES = num_cores -1 (デフォルト)
- INGESTRATE / NUMPROCESSES = 2,500 (これは true ステートメントでなければなりません。)
- MAXBATCHSIZE = 30 (デフォルトは 20。Amazon Keyspaces では最大で 30 のバッチが受け入れられます。)
- CHUNKSIZE = (INGESTRATE / NUMPROCESSES) / MAXBATCHSIZE

これで NUMPROCESSES、INGESTRATE、CHUNKSIZE の計算が完了し、データをロードする準備が整いました。

ステップ 5: **cqlsh COPY FROM** コマンドを実行して CSV ファイルからターゲットテーブルにデータをアップロードする

cqlsh COPY FROM コマンドを使用して、以下のステップを実行します。

1. cqlsh を使用して Amazon Keyspaces に接続します。
2. 次のコードがあるキースペースを選択します。

   ```
   USE catalog;
   ```

3. 書き込み整合性を LOCAL_QUORUM に設定します。データの耐久性を確保するため、Amazon Keyspaces では他の書き込み整合性設定は使用できません。以下のコードを参照してください。

   ```
   CONSISTENCY LOCAL_QUORUM;
   ```

4. 次のコード例を使用して cqlsh COPY FROM 構文を作成します。

   ```
   COPY book_awards FROM './keyspace.table.csv' WITH HEADER=true
   AND INGESTRATE=calculated ingestrate
   ```

```
AND NUMPROCESSES=calculated numprocess
AND MAXBATCHSIZE=20
AND CHUNKSIZE=calculated chunksize;
```

5. 前のステップで準備したステートメントを実行します。cqlsh は、構成したすべての設定を エコーバックします。

   a. 設定が入力と一致していることを確認します。次の例を参照してください。

   ```
   Reading options from the command line: {'chunksize': '120', 'header': 'true',
    'ingestrate': '36000', 'numprocesses': '15', 'maxbatchsize': '20'}
   Using 15 child processes
   ```

   b. 次の例に示すように、転送された行数と現在の平均レートを確認します。

   ```
   Processed: 57834 rows; Rate: 6561 rows/s; Avg. rate: 31751 rows/s
   ```

   c. cqlsh によるデータのアップロードが完了したら、次の例に示すように、データロード統計のサマリー (読み取られたファイルの数、ランタイム、スキップされた行数) を確認します。

   ```
   15556824 rows imported from 1 files in 8 minutes and 8.321 seconds (0 skipped).
   ```

このチュートリアルの最後のステップでは、データを Amazon Keyspaces にアップロードしました。

> **Important**
>
> データを転送したので、アプリケーションの通常のトラフィックパターンに合わせてターゲットテーブルのキャパシティモード設定を調整します。プロビジョンドキャパシティは、変更するまでは、時間ユニットで課金されます。

トラブルシューティング

データのアップロードが完了したら、行がスキップされたかどうかを確認します。これを行うには、ソース CSV ファイルのソースディレクトリに移動し、次の名前のファイルを検索します。

```
import_yourcsvfilename.err.timestamp.csv
```

cqlsh により、この名前のファイルに、スキップされたデータ行が書き込まれます。ファイルがソースディレクトリに存在し、その中にデータが含まれている場合、これらの行は Amazon Keyspaces にアップロードされませんでした。これらの行のアップロードを再試行するには、まずアップロード中に発生したエラーを確認し、それに応じてデータを調整します。これらの行のアップロードを再試行するために、プロセスを再実行します。

一般的なエラー

行がロードされない最も一般的な理由は、容量エラーとパーサーエラーです。

Amazon Keyspaces にデータをアップロードする際の不正なリクエストエラー

次の例では、ソーステーブルにカウンター列が含まれているため、COPY cqlshコマンドからのバッチ呼び出しがログに記録されます。Amazon Keyspaces では、ログに記録されたバッチコールはサポートされていません。

```
Failed to import 10 rows: InvalidRequest - Error from server: code=2200 [Invalid query]
 message="Only UNLOGGED Batches are supported at this time.",  will retry later,
 attempt 22 of 25
```

このエラーを解決するには、DSBulk を使用してデータを移行します。詳細については、「the section called "DSBulk を使用したデータのロード"」を参照してください。

Amazon Keyspaces にデータをアップロードする際のパーサーエラー

次の例では、ParseError が原因で行がスキップされます。

```
Failed to import 1 rows: ParseError - Invalid ... -
```

このエラーを解決するには、インポートするデータが Amazon Keyspaces のテーブルスキーマと一致していることを確認する必要があります。インポートファイルで解析エラーが発生していないか確認してください。INSERT ステートメントを使用してエラーを切り離すことで、1 行のデータの使用を試すことができます。

Amazon Keyspaces にデータをアップロードする際の容量エラー

```
Failed to import 1 rows: WriteTimeout - Error from server: code=1100 [Coordinator node
 timed out waiting for replica nodes' responses]
 message="Operation timed out - received only 0 responses." info={'received_responses':
 0, 'required_responses': 2, 'write_type': 'SIMPLE', 'consistency':
 'LOCAL_QUORUM'}, will retry later, attempt 1 of 100
```

Amazon Keyspaces では、スループットキャパシティ不足により書き込みリクエストが失敗した場合に、ReadTimeout 例外と WriteTimeout 例外を使用してその失敗が示されます。キャパシティ不足の例外を診断するために、Amazon Keyspaces は Amazon CloudWatch で WriteThrottleEvents と ReadThrottledEvents のメトリクスを公開しています。詳細については、「the section called “CloudWatch によるモニタリング”」を参照してください。

Amazon Keyspaces にデータをアップロードする際の cqlsh エラー

cqlsh エラーのトラブルシューティングに役立てるために、失敗したコマンドに --debug フラグを付けて再実行します。

互換性のないバージョンの cqlsh を使用すると、次のエラーが表示されます。

```
AttributeError: 'NoneType' object has no attribute 'is_up'
Failed to import 3 rows: AttributeError - 'NoneType' object has no attribute 'is_up',
 given up after 1 attempts
```

次のコマンドを実行して、正しいバージョンの cqlsh がインストールされていることを確認します。

```
cqlsh --version
```

出力に関して次のような内容が表示されます。

```
cqlsh 5.0.1
```

Windows を使用している場合は、cqlsh のすべてのインスタンスを cqlsh.bat に置き換えます。例えば、Windows で cqlsh 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。

```
cqlsh.bat --version
```

サーバーから cqlsh クライアントに何らかの種類のエラーが 3 回連続で送信されると、Amazon Keyspaces への接続が失敗します。cqlsh クライアントで処理が失敗すると、次のメッセージが表示されます。

```
Failed to import 1 rows: NoHostAvailable - , will retry later, attempt 3 of 100
```

このエラーを解決するには、インポートするデータが Amazon Keyspaces のテーブルスキーマと一致していることを確認する必要があります。インポートファイルで解析エラーが発生していないか確認してください。INSERT ステートメントを使用してエラーを切り離すことで、1 行のデータの使用を試すことができます。

クライアントにより接続の再確立が自動的に試行されます。

チュートリアル: DSBulk を使用した Amazon Keyspaces へのデータのロード

このステップバイステップのチュートリアルでは、GitHub で入手できる DataStax Bulk Loader (DSBulk) を使用して、Apache Cassandra から Amazon Keyspaces にデータを移行する手順が説明されています。DSBulk は、学術用途やテスト目的で Amazon Keyspaces にデータセットをアップロードする場合に便利です。本稼働ワークロードの移行方法の詳細については、「the section called "オフライン移行"」を参照してください。このチュートリアルでは、次の手順を実行します。

前提条件 – 認証情報を使用して AWS アカウントを設定し、証明書の JKS トラストストアファイルを作成し、 を設定し`cqlsh`、DSBulk をダウンロードしてインストールし、 `application.conf` ファイルを設定します。

1. ソース CSV とターゲットテーブルの作成 – ソースデータとして CSV ファイルを準備し、Amazon Keyspaces でターゲットのキースペースとテーブルを作成します。
2. データの準備 – CSV ファイル内のデータをランダム化して分析し、行サイズの平均値と最大値を求めます。
3. スループットキャパシティの設定 – データサイズと必要なロード時間に基づいて必要な書き込みキャパシティユニット (WCU) を計算し、テーブルにプロビジョニングされるキャパシティを設定します。
4. DSBulk 設定の構成 – 認証、SSL/TLS、整合性レベル、接続プールサイズなどを設定した、DSBulk 設定ファイルを作成します。
5. DSBulk ロードコマンドの実行 – DSBulk ロードコマンドを実行して、CSV ファイルから Amazon Keyspaces テーブルにデータをアップロードし、進行状況を監視します。

トピック

- 前提条件: DSBulk でデータをアップロードする前に完了する手順
- ステップ 1: DSBulk を使用してデータアップロード用のソース CSV ファイルとターゲットテーブルを作成する
- ステップ 2: DSBulk を使用してアップロード対象のデータを準備する
- ステップ 3: ターゲットテーブルのスループットキャパシティを設定する
- ステップ 4: CSV ファイルからターゲットテーブルにデータをアップロードするように DSBulk を設定する

- ステップ 5: DSBulk load コマンドを実行して CSV ファイルからターゲットテーブルにデータをアップロードする

前提条件: DSBulk でデータをアップロードする前に完了する手順

このチュートリアルを開始する前に、次のタスクを完了しておく必要があります。

1. まだサインアップしていない場合は、「」の手順に従って AWS アカウントにサインアップしますthe section called “セットアップ AWS Identity and Access Management”。
2. the section called “ AWS 認証用の IAM 認証情報を作成する” のステップに従って認証情報を作成します。
3. JKS 信頼ストアファイルを作成します。

   a. 次のコマンドを使用して Starfield デジタル証明書をダウンロードし、`sf-class2-root.crt` をローカルまたはホームディレクトリ内に保存します。

   ```
   curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
   ```

   > **Note**
   >
   > Amazon デジタル証明書を使用して Amazon Keyspaces に接続することもできます。クライアントが Amazon Keyspaces に正常に接続されている場合は、引き続き Amazon Keyspaces に接続できます。Starfield 証明書は、古い認定権限を使用しているクライアントに対して追加の下位互換性を提供するものです。

   b. Starfield デジタル証明書を trustStore ファイルに変換します。

   ```
   openssl x509 -outform der -in sf-class2-root.crt -out temp_file.der
   keytool -import -alias cassandra -keystore cassandra_truststore.jks -file
    temp_file.der
   ```

   このステップでは、キーストアのパスワードを作成し、この証明書を信頼する必要があります。対話型コマンドは次のようになります。

   ```
   Enter keystore password:
   Re-enter new password:
   Owner: OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
    Inc.", C=US
   ```

```
Issuer: OU=Starfield Class 2 Certification Authority, O="Starfield
 Technologies, Inc.", C=US
Serial number: 0
Valid from: Tue Jun 29 17:39:16 UTC 2004 until: Thu Jun 29 17:39:16 UTC 2034
Certificate fingerprints:
  MD5:  32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
  SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
  SHA256:
 14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB
Signature algorithm name: SHA1withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
Extensions:
#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86   F4 5B 55 AC DC D7 10 C2  ._.......[U.....
0010: 0E A9 88 E7                                        ....
]
[OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
 Inc.", C=US]
SerialNumber: [    00]
]
#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen:2147483647
]
#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86   F4 5B 55 AC DC D7 10 C2  ._.......[U.....
0010: 0E A9 88 E7                                        ....
]
]
Trust this certificate? [no]:  y
```

4. Cassandra クエリ言語シェル (cqlsh) 接続をセットアップし、the section called “cqlsh の使用” のステップに従って Amazon Keyspaces に接続できることを確認します。
5. DSBulk をダウンロードしてインストールします。
   a. DSBulk をダウンロードするには、次のコードを使用します。

```
curl -OL https://downloads.datastax.com/dsbulk/dsbulk-1.8.0.tar.gz
```

b. 次に、tar ファイルを解凍し、以下の例に示されているように、DSBulk を PATH に追加します。

```
tar -zxvf dsbulk-1.8.0.tar.gz
# add the DSBulk directory to the path
export PATH=$PATH:./dsbulk-1.8.0/bin
```

c. DSBulk により使用される設定を保存するための application.conf ファイルを作成します。次の例を ./dsbulk_keyspaces.conf として保存できます。ローカルノード上にいない場合は、localhost を、ローカルの Cassandra クラスターのコンタクトポイント (DNS 名や IP アドレスなど) に置き換えます。ファイル名とパスは、後で dsbulk load コマンドで指定する必要があるためメモしておいてください。

```
datastax-java-driver {
  basic.contact-points = [ "localhost"]
  advanced.auth-provider {
        class = software.aws.mcs.auth.SigV4AuthProvider
        aws-region = us-east-1
  }
}
```

d. SigV4 サポートを有効にするには、次の例に示すように、GitHub からシェード jar ファイルをダウンロードし、DSBulk lib フォルダに配置します。

```
curl -O -L https://github.com/aws/aws-sigv4-auth-cassandra-java-driver-plugin/
releases/download/4.0.6-shaded-v2/aws-sigv4-auth-cassandra-java-driver-
plugin-4.0.6-shaded.jar
```

ステップ 1: DSBulk を使用してデータアップロード用のソース CSV ファイルとターゲットテーブルを作成する

このチュートリアルでは、keyspaces_sample_table.csv という名前のカンマ区切り値 (CSV) ファイルをデータ移行用のソースファイルとして使用します。提供されたサンプルファイルには、book_awards という名前のテーブルに関する数行のデータが含まれています。

1. ソースファイルを作成します。次のオプションのいずれかを選択します。

- 次のアーカイブファイル samplemigration.zip に含まれているサンプル CSV ファイル (keyspaces_sample_table.csv) をダウンロードします。アーカイブを解凍し、keyspaces_sample_table.csv へのパスをメモしておきます。
- Apache Cassandra データベースに保存されている独自のデータを CSV ファイルに入力するには、次の例に示すように、dsbulk unload を使用してソース CSV ファイルに入力します。

```
dsbulk unload -k mykeyspace -t mytable -f ./my_application.conf
 > keyspaces_sample_table.csv
```

作成する CSV ファイルが以下の要件を満たしていることを確認してください。

- 最初の行に列名が含まれています。
- ソース CSV ファイルの列名がターゲットテーブルの列名と一致してます。
- データがカンマで区切られてます。
- すべてのデータ値が有効な Amazon Keyspaces データ型です。「the section called “データ型”」を参照してください。

2. Amazon Keyspaces でターゲットのキースペースとテーブルを作成します。

a. cqlsh を使用して Amazon Keyspaces に接続し、次の例のサービスエンドポイント、ユーザー名、およびパスワードをそれぞれ独自の値に置き換えます。

```
cqlsh cassandra.us-east-2.amazonaws.com 9142 -u "111122223333" -
p "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY" --ssl
```

b. 次の例に示すように、catalog という名前の新しいキースペースを作成します。

```
CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

c. 新しいキースペースが利用可能な状態になったら、次のコードを使用してターゲットテーブル book_awards を作成します。非同期的なリソース作成と、リソースが利用可能かどうかを確認する方法については、「the section called “キースペースの作成ステータスの確認”」を参照してください。

```
CREATE TABLE catalog.book_awards (
    year int,
    award text,
    rank int,
```

```
    category text,
    book_title text,
    author text,
    publisher text,
    PRIMARY KEY ((year, award), category, rank)
    );
```

Apache Cassandra が元のデータソースである場合、ヘッダーが一致している Amazon Keyspaces ターゲットテーブルを作成する簡単な方法は、次のステートメントに示すように、ソーステーブルから CREATE TABLE ステートメントを生成する方法です。

```
cqlsh localhost 9042  -u "username" -p "password" --execute "DESCRIBE
 TABLE mykeyspace.mytable;"
```

次に、Amazon Keyspaces で、列名とデータ型が Cassandra ソーステーブルの説明と一致しているターゲットテーブルを作成します。

ステップ 2: DSBulk を使用してアップロード対象のデータを準備する

効率的な転送のためのソースデータの準備には、2 つのステップがあります。まず、データをランダム化します。2 番目のステップでは、データを分析して、適切な dsbulk パラメータ値と必要なテーブル設定を明確にします。

データをランダム化する

dsbulk コマンドは、CSV ファイルに表示される順序と同じ順序でデータの読み取りと書き込みを行います。dsbulk コマンドを使用してソースファイルを作成すると、データはキーソートされた順序で CSV に書き込まれます。内部的には、Amazon Keyspaces でパーティションキーを使用してデータが分割されます。Amazon Keyspaces には、同一のパーティションキーに対するロードバランスリクエストに役立つロジックが内蔵されていますが、順序をランダム化すると、データのロードが高速かつ効率的になります。これは、Amazon Keyspaces で異なるパーティションにデータが書き込まれたときに発生する組み込みのロードバランシングを利用できるためです。

パーティション間で書き込みを均等に分散させるには、ソースファイル内のデータをランダム化する必要があります。アプリケーションを書き込んでこれを実行することができます。また、Shuf などのオープンソースツールを使用することもできます。Shuf は、Linux ディストリビューション、macOS (コアユーティリティを homebrew にインストールする)、Windows (Windows

Subsystem for Linux (WSL) を使用する) で無料で使用できます。このステップで列名を含むヘッダー行がシャッフルされないようにするには、追加のステップが 1 つ必要です。

ヘッダーを維持した状態でソースファイルをランダム化するには、次のコードを入力します。

```
tail -n +2 keyspaces_sample_table.csv | shuf -o keyspace.table.csv && (head
 -1 keyspaces_sample_table.csv && cat keyspace.table.csv ) > keyspace.table.csv1 &&
 mv keyspace.table.csv1 keyspace.table.csv
```

Shuf により、`keyspace.table.csv` という新しい CSV ファイルにデータが書き換えられます。これで、不要になった `keyspaces_sample_table.csv` ファイルを削除できます。

データを分析する

データを分析して、平均行サイズと最大行サイズを決定します。

この作業を行う理由は次のとおりです。

- 転送されるデータの総量を見積もる場合に、平均行サイズが役立ちます。
- データのアップロードに必要な書き込みキャパシティをプロビジョニングする際には、平均行サイズが必要になります。
- 各行のサイズが 1 MB 未満 (Amazon Keyspaces の行サイズの上限) であることを確認できます。

> **Note**
>
> このクォータは、パーティションサイズではなく、行サイズを指します。Apache Cassandra のパーティションとは異なり、Amazon Keyspaces のパーティションのサイズは事実上無制限です。パーティションキーとクラスタリング列には、メタデータ用の追加のストレージが必要です。このストレージは行の raw サイズに追加する必要があります。詳細については、「the section called “行サイズの推定”」を参照してください。

次のコードは、AWK を使用して CSV ファイルを分析し、行の平均サイズと最大サイズを出力します。

```
awk -F, 'BEGIN {samp=10000;max=-1;}{if(NR>1){len=length($0);t+=len;avg=t/
NR;max=(len>max ? len : max)}}NR==samp{exit}END{printf("{lines: %d, average: %d bytes,
 max: %d bytes}\n",NR,avg,max);}' keyspace.table.csv
```

このコードを実行すると、次の出力が表示されます。

```
using 10,000 samples:
{lines: 10000, avg: 123 bytes, max: 225 bytes}
```

最大行サイズが 1 MB を超えないようにしてください。超えた場合は、行を分割するか、データを圧縮して、行サイズを 1 MB 未満にする必要があります。このチュートリアルの次のステップでは、平均行サイズを使用して、テーブルの書き込みキャパシティをプロビジョニングします。

ステップ 3: ターゲットテーブルのスループットキャパシティを設定する

このチュートリアルでは、設定された時間範囲内でデータがロードされるように DSBulk を調整する方法を示します。事前に実行する読み取りと書き込みの数がわかっているので、プロビジョンドキャパシティモードを使用します。データ転送が完了したら、アプリケーションのトラフィックパターンに合わせてテーブルのキャパシティモードを設定する必要があります。キャパシティ管理の詳細については、「サーバーレスリソースの管理」を参照してください。

プロビジョンドキャパシティモードでは、事前にテーブルにプロビジョニングする読み取りキャパシティと書き込みキャパシティの量を指定します。書き込みキャパシティは時間ユニットで課金され、書き込みキャパシティユニット (WCU) で計測されます。各 WCU は、1 秒あたり 1 KB のデータの書き込みをサポートするのに十分な書き込みキャパシティです。データをロードする際に、書き込みレートが、ターゲットテーブルで設定した WCU の上限 (パラメータ: `write_capacity_units`) を超えないようにしてください。

デフォルトでは、1 つのテーブルに最大 40,000 の WCU を、アカウント内のすべてのテーブルに最大 80,000 の WCU をプロビジョニングすることができます。追加のキャパシティが必要な場合は、Service Quotas コンソールでクォータの増加をリクエストできます。クォータの詳細については、「クォータ」を参照してください。

挿入に必要な WCU の平均数を計算する

1 秒あたり 1 KB のデータを挿入するには、1 WCU が必要です。CSV ファイルに 360,000 の行があり、1 時間ですべてのデータをロードする場合は、1 秒あたり 100 行 (360,000 行 / 60 分 / 60 秒 = 100 行/秒) を書き込む必要があります。各行に最大 1 KB のデータがある場合、1 秒あたり 100 行を挿入するには、100 WCU をテーブルにプロビジョニングする必要があります。各行に 1.5 KB のデータがある場合、1 秒あたり 1 行を挿入するには 2 WCU が必要です。したがって、1 秒あたり 100 行を挿入するには、200 WCU をプロビジョニングする必要があります。

1 秒あたり 1 行の挿入が必要な WCU 数を調べるには、平均行サイズ (バイト) を 1024 で割り、端数を切り上げて最も近い整数にします。

例えば、平均行サイズが 3000 バイトの場合、1 秒あたり 1 行を挿入するには 3 WCU が必要です。

```
ROUNDUP(3000 / 1024) = ROUNDUP(2.93) = 3 WCUs
```

データのロード時間とキャパシティを計算する

これで、CSV ファイルの平均サイズと平均行数が分かったので、特定の時間内にデータをロードする場合に必要な WCU 数と、さまざまな WCU 設定を使用して CSV ファイルにすべてのデータをロードするのにかかるおおよその時間を計算できます。

例えば、ファイルの各行が 1 KB で、CSV ファイルに 1,000,000 行がある場合、1 時間でデータをロードするには、その時間に少なくとも 278 WCU をテーブルにプロビジョニングする必要があります。

```
1,000,000 rows * 1 KBs = 1,000,000 KBs
1,000,000 KBs / 3600 seconds =277.8 KBs / second = 278 WCUs
```

プロビジョンドキャパシティを設定する

テーブルの書き込みキャパシティは、そのテーブルの作成時、または ALTER TABLE コマンドを使用して、設定することができます。以下は、ALTER TABLE コマンドを使用してテーブルのプロビジョンドキャパシティ設定に変更を加えるための構文です。

```
ALTER TABLE catalog.book_awards WITH custom_properties={'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 100, 'write_capacity_units':
 278}} ;
```

完全な言語リファレンスについては、「the section called “CREATE TABLE”」と「the section called “ALTER TABLE”」を参照してください。

ステップ 4: CSV ファイルからターゲットテーブルにデータをアップロードするように **DSBulk** を設定する

このセクションでは、データが Amazon Keyspaces にアップロードされるように DSBulk を設定する場合に必要なステップについて説明します。DSBulk を設定するには、設定ファイルを使用します。設定ファイルは、コマンドラインから直接指定します。

1. Amazon Keyspaces への移行用の DSBulk 設定ファイルを作成します。この例では、ファイル名 dsbulk_keyspaces.conf を使用します。DSBulk 設定ファイルで以下の設定を指定します。

a. *PlainTextAuthProvider* — PlainTextAuthProvider クラスを使用して認証プロバイダーを作成します。ServiceUserName と ServicePassword は、the section called "プログラムによるアクセスのための認証情報の作成" の手順に従ってサービス固有の認証情報を生成したときに取得したユーザー名とパスワードと一致している必要があります。

b. *local-datacenter* – の値を接続 AWS リージョン 先の local-datacenter に設定します。例えば、アプリケーションを cassandra.us-east-2.amazonaws.com に接続する場合は、ローカルデータセンターを us-east-2 に設定します。使用可能なすべての については AWS リージョン、「」を参照してくださいthe section called "サービスエンドポイント"。レプリカを避けるには、slow-replica-avoidance を false に設定します。

c. *SSLEngineFactory* — SSL/TLS を設定するには、class = DefaultSslEngineFactory を使用してクラスを指定する設定ファイル (1 行が含まれている) に、セクションを追加して、SSLEngineFactory を初期化します。cassandra_truststore.jks へのパスと、作成しておいたパスワードを提供します。

d. *consistency* — 整合性レベルを LOCAL_QUORUM に設定します。他の書き込み整合性レベルはサポートされません。詳細については「the section called "サポートされている Cassandra の整合性レベル"」を参照してください。

e. プールごとの接続数は Java ドライバーで設定できます。この例では、advanced.connection.pool.local.size を 3 に設定します。

次に、完全なサンプル設定ファイルを示します。

```
datastax-java-driver {
basic.contact-points = [ "cassandra.us-east-2.amazonaws.com:9142"]
advanced.auth-provider {
    class = PlainTextAuthProvider
    username = "ServiceUserName"
    password = "ServicePassword"
}

basic.load-balancing-policy {
    local-datacenter = "us-east-2"
    slow-replica-avoidance = false
}

basic.request {
    consistency = LOCAL_QUORUM
```

```
    default-idempotence = true
}
advanced.ssl-engine-factory {
    class = DefaultSslEngineFactory
    truststore-path = "./cassandra_truststore.jks"
    truststore-password = "my_password"
    hostname-validation = false
  }
advanced.connection.pool.local.size = 3
}
```

2. DSBulk load コマンドのパラメータを確認します。

   a. *executor.maxPerSecond* — ロードコマンドにより処理が試行される 1 秒あたりの最大行数。設定しない場合、この設定は -1 となり無効になります。

      ターゲット送信先テーブルにプロビジョニングした WCU の数に基づいて executor.maxPerSecond を設定します。load コマンドの executor.maxPerSecond は制限ではなくターゲット平均です。これは、設定した数を大きく上回る可能性がある (多くの場合そうなる) ことを意味します。このような超過を許可し、データロードリクエストを処理できるだけの十分なキャパシティを確保するには、executor.maxPerSecond をテーブルの書き込みキャパシティの 90% に設定します。

      ```
      executor.maxPerSecond = WCUs * .90
      ```

      このチュートリアルでは、executor.maxPerSecond を 5 に設定します。

      > **Note**
      >
      > DSBulk 1.6.0 以上を使用している場合は、代わりに dsbulk.engine.maxConcurrentQueries を使用できます。

   b. DSBulk load コマンドのためにこれらの追加パラメータを設定します。

      - *batch-mode* — このパラメータは、パーティションキー別にオペレーションをグループ化するようにシステムに指示を出します。バッチモードは無効にすることを推奨します。バッチモードは、ホットキーシナリオを引き起こし、WriteThrottleEvents の原因となる可能性があるためです。

- *driver.advanced.retry-policy-max-retries* — これにより、失敗したクエリが再試行される回数が決まります。設定しない場合のデフォルトは 10 になります。この値は必要に応じて調整できます。
- *driver.basic.request.timeout* — クエリが返されるまでのシステムの待機時間 (分)。設定しない場合のデフォルトは「5 分」です。この値は必要に応じて調整できます。

ステップ 5: DSBulk **load** コマンドを実行して CSV ファイルからターゲットテーブルにデータをアップロードする

このチュートリアルの最後のステップでは、データを Amazon Keyspaces にアップロードします。

DSBulk `load` コマンドを使用して、以下のステップを実行します。

1. 次のコードを実行して、CSV ファイルから Amazon Keyspaces テーブルにデータをアップロードします。前の手順で作成したアプリケーション設定ファイルへのパスを必ず更新してください。

```
dsbulk load -f ./dsbulk_keyspaces.conf  --connector.csv.url keyspace.table.csv
 -header true --batch.mode DISABLED --executor.maxPerSecond 5 --
driver.basic.request.timeout "5 minutes" --driver.advanced.retry-policy.max-
retries 10 -k catalog -t book_awards
```

2. 出力には、成功したオペレーションと失敗したオペレーションの詳細が記されているログファイルの場所が含まれます。このファイルは次のディレクトリに保存されています。

```
Operation directory: /home/user_name/logs/UNLOAD_20210308-202317-801911
```

3. ログファイルのエントリには、次の例のように、メトリックが含まれます。行数が csv ファイルの行数と一致しているか確認します。

```
total | failed | rows/s | p50ms | p99ms | p999ms
   200 |      0 |    200 | 21.63 | 21.89 |  21.89
```

> **Important**
>
> データを転送したので、アプリケーションの通常のトラフィックパターンに合わせてターゲットテーブルのキャパシティモード設定を調整します。プロビジョンドキャパシティは、

変更するまでは、時間ユニットで課金されます。詳細については、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。

Amazon Keyspaces (Apache Cassandra 向け) へのアクセス

Amazon Keyspaces にアクセスするには、 コンソールを使用する AWS CloudShellか、`cqlsh`クライアント、 AWS SDK を実行するか、Apache 2.0 ライセンスの Cassandra ドライバーを使用します。Amazon Keyspaces では、Apache Cassandra 3.11.2 と互換性のあるドライバーとクライアントがサポートされています。Amazon Keyspaces にアクセスする前に、セットアップを完了 AWS Identity and Access Management し、IAM ID に Amazon Keyspaces へのアクセス許可を付与する必要があります。

セットアップ AWS Identity and Access Management

にサインアップする AWS アカウント

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://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一部では、電話またはテキストメッセージを受信し、電話のキーパッドに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。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 のセキュリティを確保し AWS IAM Identity Center、 を有効にして管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドのルートユーザーとしてサインインするを参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの AWS アカウント「ルートユーザー (コンソール) の仮想 MFA デバイスを有効にする」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 AWS IAM Identity Center ユーザーガイド」の「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン「 ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの結合」を参照してください。

Amazon Keyspaces を設定する

Amazon Keyspaces リソースへのアクセスは、IAM を使用して管理します。IAM を使用すれば、IAM ユーザー、ロール、フェデレーティッドアイデンティティに、Amazon Keyspaces の特定のリソースへの読み書きアクセス許可を付与するポリシーをアタッチできます。

IAM アイデンティへのアクセス許可の付与を開始するには、Amazon Keyspaces 用の AWS マネージドポリシーのいずれかを使用できます。

- AmazonKeyspacesFullAccess – Amazon Keyspaces 内のすべてのリソースへのアクセス許可を付与し、すべての機能へのフルアクセスを認めるポリシー。
- AmazonKeyspacesReadOnlyAccess_v2 – Amazon Keyspaces への読み取り専用のアクセス許可を付与するポリシー。

マネージドポリシーで定義されているアクションの詳細については、「the section called “AWS マネージドポリシー”」を参照してください。

IAM アイデンティティが実行可能なアクションの範囲や、アイデンティティがアクセス可能なリソースを制限するには、`AmazonKeyspacesFullAccess` マネージドポリシーをテンプレートとして使用してカスタムポリシーを作成し、不要なアクセス許可をすべて削除できます。また、特定のキースペースやテーブルにアクセスを限定することもできます。Amazon Keyspaces でアクションを制限する方法や、特定のリソースにアクセスを限定する方法については、「the section called “Amazon Keyspaces で IAM が機能する仕組み”」を参照してください。

を作成し、IAM ID に Amazon Keyspaces へのアクセスを許可するポリシー AWS アカウント を作成した後で Amazon Keyspaces にアクセスするには、次のいずれかのセクションに進みます。

- コンソールの使用
- の使用 AWS CloudShell

コンソールを使用した Amazon Keyspaces へのアクセス

Amazon Keyspaces コンソールには https://console.aws.amazon.com/keyspaces/home からアクセスできます。 AWS Management Console アクセスの詳細については、「IAM ユーザーガイド」の「IAM ユーザーの へのアクセスの制御 AWS Management Console」を参照してください。

コンソールを使用して、Amazon Keyspaces で以下のことを行うことができます。

- キースペースとテーブルを作成、削除、管理する。
- テーブルの [モニタリング] タブで、テーブルの重要なメトリクスを監視する。
  - 請求可能なテーブルサイズ (バイト数)
  - キャパシティメトリクス
- CQL エディタを使用してクエリを実行する (データの挿入、更新、削除など)。
- アカウントのパーティショナー設定を変更する。
- ダッシュボードでアカウントのパフォーマンスとエラーのメトリクスを表示する。

Amazon Keyspaces のキースペースとテーブルを作成してサンプルアプリケーションデータで設定する方法については、「Amazon Keyspaces (Apache Cassandra 向け) の使用開始」を参照してください。

AWS CloudShell を使用して Amazon Keyspaces にアクセスする

AWS CloudShell はブラウザベースの事前認証済みシェルで、 から直接起動できます AWS Management Console。任意のシェル (Bash、PowerShell、または Z シェル) を使用して、 AWS サービスに対して AWS CLI コマンドを実行できます。`cqlsh` を使用して Amazon Keyspaces を操作するには、`cqlsh-expansion` をインストールする必要があります。`cqlsh-expansion` のインストール手順については、「the section called “`cqlsh-expansion` の使用”」を参照してください。

AWS CloudShell から を起動 AWS Management Consoleすると、コンソールへのサインインに使用した AWS 認証情報が新しいシェルセッションで自動的に利用可能になります。 AWS CloudShell ユーザーのこの事前認証により、 `cqlsh`または AWS CLI バージョン 2 (シェルのコンピューティング環境にプリインストール済み) を使用して Amazon Keyspaces などの AWS サービスとやり取りするときに、認証情報の設定をスキップできます。

の IAM アクセス許可の取得 AWS CloudShell

が提供するアクセス管理リソースを使用すると AWS Identity and Access Management、管理者は IAM ユーザーにアクセス許可を付与して、環境の機能にアクセスして AWS CloudShell 使用できます。

管理者がユーザーにアクセス権を付与する最も簡単な方法は、 AWS マネージドポリシーを使用することです。AWS マネージドポリシーは、 AWSが作成および管理するスタンドアロンポリシーです。CloudShell の次の AWS 管理ポリシーを IAM ID にアタッチできます。

- `AWSCloudShellFullAccess`: すべての機能へのフルアクセス AWS CloudShell で を使用するアクセス許可を付与します。

IAM ユーザーが実行できるアクションの範囲を制限する場合は AWS CloudShell、`AWSCloudShellFullAccess`管理ポリシーをテンプレートとして使用するカスタムポリシーを作成できます。CloudShell でユーザーが使用できるアクションの制限の詳細については、AWS CloudShell 「 ユーザーガイド」の「IAM ポリシーによる AWS CloudShell アクセスと使用状況の管理」を参照してください。

> **Note**
>
> IAM アイデンティティには、Amazon Keyspaces への呼び出しを行うアクセス許可を付与するポリシーも必要です。

AWS 管理ポリシーを使用して、IAM ID に Amazon Keyspaces へのアクセスを許可するか、管理ポリシーをテンプレートとして開始し、不要なアクセス許可を削除できます。また、特定のキースペースやテーブルにアクセスを限定して、カスタムのポリシーを作成することもできます。Amazon Keyspaces 用の次のマネージドポリシーを IAM アイデンティティにアタッチできます。

- AmazonKeyspacesFullAccess – Amazon Keyspaces の使用を許可し、すべての機能へのフルアクセスを認めるポリシー。

マネージドポリシーで定義されているアクションの詳細については、「the section called “AWS マネージドポリシー”」を参照してください。

Amazon Keyspaces でアクションを制限する方法や、特定のリソースにアクセスを限定する方法については、「the section called “Amazon Keyspaces で IAM が機能する仕組み”」を参照してください。

を使用した Amazon Keyspaces の操作 AWS CloudShell

AWS CloudShell から を起動すると AWS Management Console、 cqlshまたはコマンドラインインターフェイスを使用して Amazon Keyspaces の操作をすぐに開始できます。cqlsh-expansion をまだインストールしていない場合は、「the section called “cqlsh-expansion の使用”」で詳しい手順を確認してください。

> **Note**
>
> cqlsh-expansion で を使用する場合 AWS CloudShell、シェル内で既に認証されているため、呼び出しを行う前に認証情報を設定する必要はありません。

Amazon Keyspaces に接続して新しいキースペースを作成した後、次に、システムテーブルから読み取って、 を使用してキースペースが作成されたことを確認します。 AWS CloudShell

1. から AWS Management Console、ナビゲーションバーで使用できる以下のオプションを選択して CloudShell を起動できます。
   - CloudShell アイコンを選択します。
   - 検索ボックスに「cloudshell」を入力し始めてから [CloudShell] オプションを選択します。
2. 次のコマンドを使用して、Amazon Keyspaces への接続を確立できます。*cassandra.us-east-1.amazonaws.com* は、実際のリージョンに適したエンドポイントに置き換えてください。

   ```
   cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
   ```

   正常に接続されると次の例のような出力が表示されます。

   ```
   Connected to Amazon Keyspaces at cassandra.us-east-1.amazonaws.com:9142
   [cqlsh 6.1.0 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
   Use HELP for help.
   cqlsh current consistency level is ONE.
   cqlsh>
   ```

3. mykeyspace という名前の新しいキースペースを作成します。そのためには、次のコマンドを使用できます。

```
CREATE KEYSPACE mykeyspace WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

4. キースペースが作成されたことを確認するには、次のコマンドを使用してシステムテーブルから読み取ります。

```
SELECT * FROM system_schema_mcs.keyspaces WHERE keyspace_name = 'mykeyspace';
```

コールが成功すると、コマンドラインに次の出力に似たサービスからのレスポンスが表示されます。

```
 keyspace_name  | durable_writes | replication
----------------+----------------
+-------------------------------------------------------------------------------------------
 mykeyspace     |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}

(1 rows)
```

Amazon Keyspaces にプログラムによってアクセスするための認証情報を作成する

Amazon Keyspaces リソースへのプログラムアクセスに必要な認証情報を、ユーザーとアプリケーションに提供するには、次のいずれかを実行します。

- Cassandra が認証とアクセス管理に使用する従来のユーザー名とパスワードに似たサービス固有の認証情報を作成します。 AWS サービス固有の認証情報は、特定の AWS Identity and Access Management (IAM) ユーザーに関連付けられ、作成されたサービスにのみ使用できます。詳細については、「IAM ユーザーガイド」の「Using IAM with Amazon Keyspaces (for Apache Cassandra)」(IAM と Amazon Keyspaces (Apache Cassandra 向け) の併用) を参照してください。

> **Warning**
>
> IAM ユーザーは長期認証情報を保有するため、セキュリティ上のリスクが生じます。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を削除することをお勧めします。

- セキュリティを強化するために、すべての AWS サービスで使用される IAM ID を作成し、一時的な認証情報を使用することをお勧めします。Cassandra クライアントドライバー用の Amazon Keyspaces SigV4 認証プラグインを使用すると、ユーザー名とパスワードではなく IAM アクセスキーを使用して Amazon Keyspaces のコールの認証を行うことができます。Amazon Keyspaces SigV4 プラグインにより、IAM ユーザー、ロール、およびフェデレーティッドアイデンティティの認証を Amazon Keyspaces API リクエストで行えるようにするには、「AWS Signature Version 4 process (SigV4)」(署名バージョン 4 プロセス (SigV4)) を参照してください。

  SigV4 プラグインは次の場所からダウンロードできます。

  - Java: https://github.com/aws/aws-sigv4-auth-cassandra-java-driver-plugin。
  - Node.js: https://github.com/aws/aws-sigv4-auth-cassandra-nodejs-driver-plugin。
  - Python: https://github.com/aws/aws-sigv4-auth-cassandra-python-driver-plugin。
  - Go: https://github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin。

  SigV4 認証プラグインを使用して接続を確立する方法のサンプルコードについては、「the section called “Cassandra クライアントドライバーの使用”」を参照してください。

トピック

- Amazon Keyspaces にプログラムによってアクセスするためのサービス固有の認証情報を作成する
- Amazon Keyspaces の AWS 認証情報の作成と設定

Amazon Keyspaces にプログラムによってアクセスするためのサービス固有の認証情報を作成する

サービス固有の認証情報は、Cassandra が認証とアクセス管理に使用するような従来方式のユーザー名とパスワードに似ています。サービス固有の認証情報があれば、IAM ユーザーは特定の AWS サービスにアクセスできるようになります。これらの長期的な認証情報は、他の AWS サービスへの

アクセスには使用できません。これらは特定の IAM ユーザーに関連付けられているため、他の IAM ユーザーが使用することはできません。

> **Important**
>
> サービス固有の認証情報は、特定の IAM ユーザーに関連付けられた長期の資格情報で、認証情報の作成時に対象としていたサービスにしか使用できません。一時的な認証情報を使用してすべての AWS リソースにアクセスするためのアクセス許可を IAM ロールまたはフェデレーティッド ID に付与するには、AWS Amazon Keyspaces の SigV4 認証プラグインで認証を使用する必要があります。

次のいずれかの手順でサービス固有の認証情報を生成してください。

Console

コンソールを使用してサービス固有の認証情報を作成する

1. にサインイン AWS Management Console し、 で AWS Identity and Access Management コンソールを開きますhttps://console.aws.amazon.com/iam/home。
2. ナビゲーションペインで、[Users] (ユーザー) を選択し、次に、事前に作成した、Amazon Keyspaces のアクセス許可 (ポリシーがアタッチされている) を持っているユーザーを選択します。
3. [Security Credentials] (セキュリティ認証情報) を選択します。[Credentials for Amazon Keyspaces] (Amazon Keyspaces の認証情報) で、[Generate credentials] (認証情報を生成) を使用してサービス固有の認証情報を生成します。

   これで、サービス固有の認証情報が利用可能になりました。パスワードを表示またはダウンロードできるのはこの時点だけです。後で回復することはできません。ただし、パスワードはいつでもリセットできます。ユーザーおよびパスワードは後で必要になるため、安全な場所に保存します。

CLI

を使用してサービス固有の認証情報を作成する AWS CLI

サービス固有の認証情報を生成する前に、 AWS Command Line Interface () をダウンロード、インストール、設定する必要がありますAWS CLI。

1. AWS CLI で をダウンロードしますhttp://aws.amazon.com/cli。

   > **Note**
   >
   > は Windows、macOS、または Linux で AWS CLI 実行されます。

2. 「 AWS Command Line Interface ユーザーガイド」の AWS 「 CLI のインストール」および「 CLI AWS の設定」の手順に従います。
3. を使用して AWS CLI、次のコマンドを実行してユーザー のサービス固有の認証情報を生成しalice、Amazon Keyspaces にアクセスできるようにします。

```
aws iam create-service-specific-credential \
    --user-name alice \
    --service-name cassandra.amazonaws.com
```

出力は次のようになります。

```
{
    "ServiceSpecificCredential": {
        "CreateDate": "2019-10-09T16:12:04Z",
        "ServiceName": "cassandra.amazonaws.com",
        "ServiceUserName": "alice-at-111122223333",
        "ServicePassword": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "ServiceSpecificCredentialId": "ACCAYFI33SINPGJEBYESF",
        "UserName": "alice",
        "Status": "Active"
    }
}
```

出力の ServiceUserName と ServicePassword の値を書き留めておきます。これらの値は後で必要になるため、安全な場所に保存しておいてください。

> **Important**
>
> ServicePassword を利用できるのはこの時点だけです。

Amazon Keyspaces の AWS 認証情報の作成と設定

AWS CLI、 AWS SDK、または Cassandra クライアントドライバーと SigV4 プラグインを使用してプログラムで Amazon Keyspaces にアクセスするには、アクセスキーを持つ IAM ユーザーまたはロールが必要です。を AWS プログラムで使用する場合、 AWS がプログラムによる呼び出しで ID を検証できるように、 AWS アクセスキーを指定します。アクセスキーは、アクセスキー ID (例: AKIAIOSFODNN7EXAMPLE) とシークレットアクセスキー (例: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY) で構成されます。このトピックでは、このプロセスで必要になる手順を説明します。

セキュリティのベストプラクティスとして、アクセス許可が制限された IAM ユーザーを作成し、代わりに、特定のタスクの実行に必要なアクセス許可を IAM ロールに関連付けておくことをお勧めします。そうしておけば、IAM ユーザーが一時的に IAM ロールを引き受けて、必要なタスクを実行できます。例えば、アカウントの IAM ユーザーが Amazon Keyspaces コンソールを使用する場合、別のロールに切り替えて、コンソールでそのロールのアクセス許可を一時的に使用できます。ユーザーは、元のアクセス権限を返却し、そのロールに割り当てられたアクセス権限を取得します。ユーザーがそのロールを終了すると、元のアクセス権限に戻ります。ユーザーがロールを引き受ける際に使用する認証情報は一時的なものです。一方で、IAM ユーザーには長期的な認証情報が割り当てられるため、ロールを引き受ける代わりにアクセス許可を直接割り当てた場合、セキュリティリスクを伴います。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を削除することをお勧めします。ロールの詳細については、「IAM ユーザーガイド」の「ロールの一般的なシナリオ: ユーザー、アプリケーション、およびサービス」を参照してください。

トピック

- Cassandra クライアントドライバー用の AWS CLI、 AWS SDK、または Amazon Keyspaces SigV4 プラグインに必要な認証情報
- IAM ロールと SigV4 プラグインで Amazon Keyspaces に接続するための一時的な認証情報を作成する
- AWS アカウントの Amazon Keyspaces にプログラムでアクセスするための IAM ユーザーを作成する
- IAM ユーザーの新しいアクセスキーを作成する
- プログラムによるアクセス用のアクセスキーを保存する

Cassandra クライアントドライバー用の AWS CLI、 AWS SDK、または Amazon Keyspaces SigV4 プラグインに必要な認証情報

IAM ユーザーまたはロールの認証には、次の認証情報が必要です。

`AWS_ACCESS_KEY_ID`

IAM ユーザーまたはロールに関連付けられた AWS アクセスキーを指定します。

プログラムによる Amazon Keyspaces への接続には、アクセスキー `aws_access_key_id` が必要です。

`AWS_SECRET_ACCESS_KEY`

アクセスキーに関連付けられるシークレットキーを指定します。これは、基本的にアクセスキーの「パスワード」です。

プログラムによる Amazon Keyspaces への接続には、`aws_secret_access_key` が必要です。

`AWS_SESSION_TOKEN` – オプション

AWS Security Token Service オペレーションから直接取得した一時的なセキュリティ認証情報を使用している場合に必要なセッショントークン値を指定します。詳細については、「the section called “Amazon Keyspaces への接続用の一時的な認証情報の作成”」を参照してください。

IAM ユーザーに接続する場合、`aws_session_token` は不要です。

IAM ロールと SigV4 プラグインで Amazon Keyspaces に接続するための一時的な認証情報を作成する

Amazon Keyspaces にプログラムによってアクセスする場合に推奨される方法は、一時的な認証情報を使用して SigV4 プラグインで認証することです。多くの場合、期限のない長期のアクセスキー (IAM ユーザーのアクセスキーなど) は必要ありません。その代わり、IAM ロールを作成して一時的なセキュリティ認証情報を生成することができます。一時的なセキュリティ認証情報は、アクセスキー ID とシークレットアクセスキーで構成されていますが、認証情報がいつ無効になるかを示すセキュリティトークンも含んでいます。長期アクセスキーの代わりに IAM ロールを使用する方法の詳細については、「IAM ロールへの切り替え (AWS API)」を参照してください。

一時認証情報を使用するには、まず IAM ロールを作成しておく必要があります。

Amazon Keyspaces に読み取り専用アクセスを付与する IAM ロールを作成します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/:// www.com」で IAM コンソールを開きます。
2. ナビゲーションペインで [Roles (ロール)]、[Create role (ロールを作成)] の順に選択します。
3. [ロールを作成] ページの [信頼されたエンティティのタイプを選択] で、[AWS サービス)] を選択します。[Choose a use case (ユースケースを選択)] で [Amazon EC2] を選択し、[Next (次へ)] を選択します。
4. 「アクセス権限を追加」ページの「アクセス権限ポリシー」で、ポリシーリストから「Amazon Keyspaces 読み取り専用アクセス」を選択し、「次へ」を選択します。
5. 「名前、レビュー、作成」ページで、ロールの名前を入力し、「信頼できるエンティティを選択」 と「権限を追加」セクションを確認します。このページでは、ロールにオプションでタグを追加することもできます。終了したら、[Create role (ロールを作成)] を選択します。この名前は Amazon EC2 インスタンスを起動するときに必要になるため、覚えておいてください。

コードで一時的なセキュリティ認証情報を使用するには、 などの AWS Security Token Service API をプログラムで呼び出しAssumeRole、前のステップで作成した IAM ロールから、結果として得られる認証情報とセッショントークンを抽出します。次に、これらの値を後続の 呼び出しの認証情報として使用します AWS。以下の例で、一時的なセキュリティ認証情報を使用する方法に関する疑似コードを示します。

```
assumeRoleResult = AssumeRole(role-arn);
tempCredentials = new SessionAWSCredentials(
   assumeRoleResult.AccessKeyId,
   assumeRoleResult.SecretAccessKey,
   assumeRoleResult.SessionToken);
cassandraRequest = CreateAmazoncassandraClient(tempCredentials);
```

Python ドライバを使用して Amazon Keyspaces にアクセスするために一時的な認証情報を実装する例については、「???」を参照してください。

AssumeRole、GetFederationToken、およびその他の API オペレーションを呼び出す方法の詳細については、「AWS Security Token Service API リファリファレンス」を参照してください。結果から一時的なセキュリティ認証情報とセッショントークンを取得する方法の詳細については、お使いの SDK のドキュメントを参照してください。すべての AWS SDKs のドキュメントは、 SDKs とツールキットセクションのメインAWS ドキュメントページにあります。

AWS アカウントの Amazon Keyspaces にプログラムでアクセスするための IAM ユーザーを作成する

AWS CLI、 AWS SDK、または SigV4 プラグインを使用して Amazon Keyspaces にプログラムでアクセスするための認証情報を取得するには、まず IAM ユーザーまたはロールを作成する必要があります。IAM ユーザーを作成して、IAM ユーザーによる Amazon Keyspaces へのプログラムアクセスを設定する手順を以下に示します。

1. 、 AWS Management Console、 AWS CLI Tools for Windows PowerShell、または AWS API オペレーションを使用してユーザーを作成します。でユーザーを作成すると AWS Management Console、認証情報が自動的に作成されます。
2. プログラムによりユーザーを作成する場合は、追加のステップでそのユーザーのアクセスキー (アクセスキー ID とシークレットアクセスキー) を作成する必要があります。
3. Amazon Keyspaces へのアクセス許可をユーザーに付与します。

IAM ユーザーの作成に必要なアクセス許可の詳細については、「他の IAM リソースへのにアクセスするのに必要なアクセス許可」を参照してください。

Console

プログラムによってアクセスするための IAM ユーザーを作成する (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/:// www.com」で IAM コンソールを開きます。
2. ナビゲーションペインで [Users] (ユーザー)、[Add user] (ユーザーを追加する) の順に選択します。
3. 新しいユーザーのユーザー名を入力します。これは のサインイン名です AWS。

> **Note**
>
> ユーザー名は、最大 64 文字の英数字、プラス記号 (+)、等号 (=)、カンマ (,)、ピリオド (.)、アットマーク (@)、アンダースコア (_)、ハイフン (-) を組み合わせて指定します。名前はアカウント内で一意である必要があります。大文字と小文字は区別されません。例えば、TESTUSER というユーザーと testuser というユーザーを作成することはできません。

4. 新しいユーザーのアクセスキーを作成するには、[アクセスキー - プログラムによるアクセス] を選択します。[Final (最終)] ページに到達すると、アクセスキーの表示やダウンロードができます。

   [Next: Permissions (次へ: 許可)] を選択します。

5. [アクセス許可を設定] ページで、[既存のポリシーを直接添付] を選択して、新しいユーザーにアクセス許可を割り当てます。

   このオプションは、アカウントで利用可能な AWS 管理ポリシーとカスタマー管理ポリシーのリストを表示します。検索フィールドに `keyspaces` を入力して、Amazon Keyspaces に関連するポリシーのみを表示できます。

   Amazon Keyspaces の場合、利用可能なマネージドポリシーは `AmazonKeyspacesFullAccess` と `AmazonKeyspacesReadOnlyAccess` です。各ポリシーの詳細については、「the section called “AWS マネージドポリシー”」を参照してください。

   テストが目的の場合や、接続のチュートリアルに従う場合は、新しい IAM ユーザー用に `AmazonKeyspacesReadOnlyAccess` ポリシーを選択してください。注意:ベストプラクティスとして、最小権限の原則に従い、特定のリソースへのアクセスを制限し、必要なアクションのみを許可するカスタムポリシーを作成することをお勧めします。IAM ポリシーと Amazon Keyspaces サンプルポリシーの詳細については、「the section called “Amazon Keyspaces のアイデンティティベースポリシー”」を参照してください。カスタムのアクセス許可ポリシーを作成したら、ポリシーをロールにアタッチし、ユーザーが適切なロールを一時的に引き受けられるようにします。

   [Next: Tags] (次へ: タグ) を選択します。

6. [タグを追加 (オプション)] ページでは、ユーザーのタグを追加するか、[次へ:確認] を選択できます。

7. [レビュー] ページで、この時点までに行ったすべての選択を確認できます。続行する準備ができたら、[Create user (ユーザーの作成)] を選択します。

8. ユーザーのアクセスキー (アクセスキー ID とシークレットアクセスキー) を表示するには、パスワードとアクセスキーの横にある [Show (表示)] を選択します。アクセスキーを保存するには、[Download .csv (.csv のダウンロード)] を選択し、安全な場所にファイルを保存します。

> **Important**
>
> シークレットアクセスキーの表示またはダウンロードを実行できるのはこの時点だけです。ユーザーが SigV4 プラグインを使用できるようにするには、あらかじめこの情報を提供しておく必要があります。ユーザーの新しいアクセスキー ID とシークレットアクセスキーは、安全な場所に保存してください。このステップを行った後に、シークレットキーに再度アクセスすることはできません。

CLI

プログラムによってアクセスするための IAM ユーザーを作成する (AWS CLI)

1. 次の AWS CLI コードでユーザーを作成します。

   - aws iam create-user

2. ユーザーにプログラムアクセス権を付与します。これにはアクセスキーが必要です。アクセスキーは、次の方法で生成できます。

   - AWS CLI: aws iam create-access-key
   - New-IAMAccessKey Tools for Windows PowerShell
   - IAM API: CreateAccessKey

   > **Important**
   >
   > シークレットアクセスキーの表示またはダウンロードを実行できるのはこの時点だけです。ユーザーが SigV4 プラグインを使用できるようにするには、あらかじめこの情報を提供しておく必要があります。ユーザーの新しいアクセスキー ID とシークレットアクセスキーは、安全な場所に保存してください。このステップを行った後に、シークレットキーに再度アクセスすることはできません。

3. ユーザーのアクセス許可を定義する `AmazonKeyspacesReadOnlyAccess` ポリシーをユーザーにアタッチします。注意: ベストプラクティスとして、ユーザーのアクセス許可の管理は、ユーザーをグループに追加してグループにポリシーをアタッチすることで (ユーザーに直接アタッチするのではなく) 行うことをお勧めします。

   - AWS CLI: aws iam attach-user-policy

IAM ユーザーの新しいアクセスキーを作成する

IAM ユーザーをすでに作成している場合は、いつでも新しいアクセスキーを作成できます。アクセスキーの更新方法など、キー管理の詳細については、「IAM ユーザーのアクセスキーの管理」を参照してください。

IAM ユーザーのアクセスキーを作成するには (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/:// www.com」で IAM コンソールを開きます。
2. ナビゲーションペインで [Users (ユーザー)] を選択します。
3. アクセスキーを作成するユーザーの名前を選択します。
4. ユーザーの [Summary (概要)] ページで、[Security credentials (セキュリティ認証情報)] タブをクリックします。
5. [Access keys (アクセスキー)] セクションで、[Create access key (アクセスキーを作成)] を選択します。

   新しいアクセスキーのペアを表示するには、[Show (表示)] を選択します。認証情報は以下のようになります:

   - アクセスキーID: AKIAIOSFODNN7EXAMPLE
   - シークレットアクセスキー: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

   > **Note**
   >
   > このダイアログボックスを閉じた後で、シークレットアクセスキーに再度アクセスすることはできません。

6. キーペアをダウンロードするには、[Download .csv file.csv (.csv ファイルのダウンロード)] を選択します。このキーは安全な場所に保存してください。
7. .csv ファイルをダウンロードしたら、[Close (閉じる)] を選択します。

アクセスキーを作成すると、キーペアはデフォルトで有効になり、すぐにキーペアを使用できるようになります。

プログラムによるアクセス用のアクセスキーを保存する

ベストプラクティスとして、アクセスキーをコードに直接埋め込まないことをお勧めします。 AWS SDKs と AWS コマンドラインツールを使用すると、既知の場所にアクセスキーを配置できるため、コードに保持する必要はありません。次のいずれかの場所にアクセスキーを置きます。

- 環境変数 – マルチテナントシステムでは、システム環境変数ではなくユーザー環境変数を選択します。
- CLI 認証情報ファイル – コマンド `aws configure` を実行すると、`credentials` ファイルと `config` ファイルが更新されます。`credentials` ファイルは、Linux、macOS、Unix では `~/.aws/credentials` に、Windows では `C:\Users\USERNAME\.aws\credentials` にあります。このファイルには、`default` プロファイルと任意の名前付きプロファイルの認証情報の詳細が含まれています。
- CLI 設定ファイル - コマンド `aws configure` を実行すると、`credentials` ファイルと `config` ファイルが更新されます。`config` ファイルは、Linux、macOS、Unix では `~/.aws/config` に、Windows では `C:\Users\USERNAME\.aws\config` にあります。このファイルには、 デフォルトプロファイルとあらゆる名前付きプロファイルの設定が含まれています。

アクセスキーを環境変数として保存することは、「the section called “Java 4.x の認証プラグイン”」の前提条件です。これにはデフォルトが含まれていることに注意してください AWS リージョン。クライアントでは、デフォルトの認証情報プロバイダーチェーンを使用して認証情報が検索されます。環境変数として保存されたアクセスキーは、設定ファイルなどの他のすべての場所よりも優先されます。詳細については、「Configuration settings and precedence (設定と優先順位)」を参照してください。

次の例では、デフォルトのユーザーの環境変数を設定する方法を示します。

Linux, macOS, or Unix

```
$ export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
$ export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of security token>
$ export AWS_DEFAULT_REGION=aws-region
```

環境変数を設定すると、シェルセッションの終了時まで、または変数に別の値を設定するまで、使用する値が変更されます。変数をシェルのスタートアップスクリプトで設定することで、変数をこれからのセッションで永続的にすることができます。

Windows Command Prompt

```
C:\> setx AWS_ACCESS_KEY_ID AKIAIOSFODNN7EXAMPLE
C:\> setx AWS_SECRET_ACCESS_KEY wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
C:\> setx AWS_SESSION_TOKEN AQoDYXdzEJr...<remainder of security token>
C:\> setx AWS_DEFAULT_REGION aws-region
```

`set` を使用して環境変数を設定すると、現在のコマンドプロンプトセッションの終了時まで、または変数を別の値に設定するまで、使用する値が変更されます。`setx` を使用して環境変数を設定すると、現在のコマンドプロンプトセッションおよびコマンド実行後に作成するすべてのコマンドプロンプトセッションで使用する値が変更されます。これは、コマンド実行時にすでに実行されている他のコマンドシェルには影響を及ぼしません。

PowerShell

```
PS C:\> $Env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
PS C:\> $Env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
PS C:\> $Env:AWS_SESSION_TOKEN="AQoDYXdzEJr...<remainder of security token>"
PS C:\> $Env:AWS_DEFAULT_REGION="aws-region"
```

前の例に示すように PowerShell プロンプトで環境変数を設定した場合は、現在のセッションの期間だけ値が保存されます。PowerShell およびコマンドプロンプトセッション間で環境変数を永続的に設定するには、[コントロールパネル] の [システム] アプリケーションを使用して変数を保存します。または、変数を PowerShell プロファイルに追加すると、その変数を今後のすべての PowerShell セッションに設定できます。環境変数の保存やそれをセッション間で永続的に維持する詳細については、「PowerShell documentation」(PowerShell ドキュメント) を参照してください。

Amazon Keyspaces のサービスエンドポイント

トピック

- ポートとプロトコル
- グローバルエンドポイント
- AWS GovCloud (US) Region FIPS エンドポイント
- 中国リージョンのエンドポイント

ポートとプロトコル

Amazon Keyspaces にプログラムでアクセスするには、cqlshクライアントを実行するか、Apache 2.0 ライセンスの Cassandra ドライバーを使用するか、 AWS CLI と AWS SDK を使用します。

次の表は、さまざまなアクセスメカニズムのポートとプロトコルをまとめたものです。

プログラムによるアクセス	ポート	プロトコル
CQLSH	9142	TLS
Cassandra ドライバー	9142	TLS
AWS CLI	443	HTTPS
AWS SDK	443	HTTPS

TLS 接続の場合、Amazon Keyspaces は Starfield CA でサーバーに対して認証を行います。詳細については、「the section called “TLS の cqlsh 接続を手動で設定する方法”」または the section called “Cassandra クライアントドライバーの使用” 章のドライバーの「始める前に」セクションを参照してください。

グローバルエンドポイント

Amazon Keyspaces は以下にあります AWS リージョン。この表は、各リージョンで利用可能なサービスエンドポイントを示します。

リージョン名	リージョン	エンドポイント	プロトコル
米国東部 (オハイオ)	us-east-2	cassandra.us-east-2.amazonaws.com	HTTPS および TLS
米国東部 (バージニア北部)	us-east-1	cassandra.us-east-1.amazonaws.com cassandra-fips.us-east-1.amazonaws.com	HTTPS および TLS

リージョン名	リージョン	エンドポイント	プロトコル
			TLS
米国西部 (北カリフォルニア)	us-west-1	cassandra.us-west-1.amazonaws.com	HTTPS および TLS
米国西部 (オレゴン)	us-west-2	cassandra.us-west-2.amazonaws.com cassandra-fips.us-west-2.amazonaws.com	HTTPS および TLS TLS
アフリカ (ケープタウン)	af-south-1	cassandra.af-south-1.amazonaws.com	HTTPS および TLS
アジアパシフィック (香港)	ap-east-1	cassandra.ap-east-1.amazonaws.com	HTTPS および TLS
アジアパシフィック (ムンバイ)	ap-south-1	cassandra.ap-south-1.amazonaws.com	HTTPS および TLS
アジアパシフィック (ソウル)	ap-northeast-2	cassandra.ap-northeast-2.amazonaws.com	HTTPS および TLS
アジアパシフィック (シンガポール)	ap-southeast-1	cassandra.ap-southeast-1.amazonaws.com	HTTPS および TLS

リージョン名	リージョン	エンドポイント	プロトコル
アジアパシフィック (シドニー)	ap-southeast-2	cassandra.ap-southeast-2.amazonaws.com	HTTPS および TLS
アジアパシフィック (東京)	ap-northeast-1	cassandra.ap-northeast-1.amazonaws.com	HTTPS および TLS
カナダ (中部)	ca-central-1	cassandra.ca-central-1.amazonaws.com	HTTPS および TLS
欧州 (フランクフルト)	eu-central-1	cassandra.eu-central-1.amazonaws.com	HTTPS および TLS
欧州 (アイルランド)	eu-west-1	cassandra.eu-west-1.amazonaws.com	HTTPS および TLS
欧州 (ロンドン)	eu-west-2	cassandra.eu-west-2.amazonaws.com	HTTPS および TLS
欧州 (パリ)	eu-west-3	cassandra.eu-west-3.amazonaws.com	HTTPS および TLS
欧州 (ストックホルム)	eu-north-1	cassandra.eu-north-1.amazonaws.com	HTTPS および TLS
中東 (バーレーン)	me-south-1	cassandra.me-south-1.amazonaws.com	HTTPS および TLS

リージョン名	リージョン	エンドポイント	プロトコル	
南米 (サンパウロ)	sa-east-1	cassandra.sa-east-1.amazonaws.com	HTTPS および TLS	
AWS GovCloud (米国東部)	us-gov-east-1	cassandra.us-gov-east-1.amazonaws.com	HTTPS および TLS	
AWS GovCloud (米国西部)	us-gov-west-1	cassandra.us-gov-west-1.amazonaws.com	HTTPS および TLS	

AWS GovCloud (US) Region FIPS エンドポイント

AWS GovCloud (US) Regionで使用可能な FIPS エンドポイント。詳細については、『AWS GovCloud (US) ユーザーガイド』の「Amazon Keyspaces」を参照してください。

リージョン名	リージョン	FIPS エンドポイント	プロトコル
AWS GovCloud (米国東部)	us-gov-east-1	cassandra.us-gov-east-1.amazonaws.com	HTTPS および TLS
AWS GovCloud (米国西部)	us-gov-west-1	cassandra.us-gov-west-1.amazonaws.com	HTTPS および TLS

中国リージョンのエンドポイント

AWS 中国リージョンでは、次の Amazon Keyspaces エンドポイントを使用できます。

これらのエンドポイントにアクセスするには、中国リージョン固有のアカウント認証情報を別途取得する必要があります。詳細については、「中国でのサインアップ、アカウント、認証情報」を参照してください。

リージョン名	リージョン	エンドポイント	プロトコル
中国 (北京)	cn-north-1	cassandra.cn-north-1.amazonaws.com.cn	HTTPS および TLS
中国 (寧夏)	cn-northwest-1	cassandra.cn-northwest-1.amazonaws.com.cn	HTTPS および TLS

cqlsh を使用した Amazon Keyspaces への接続

cqlsh で Amazon Keyspaces に接続するには、cqlsh-expansion を使用します。このツールキットには、cqlsh や Apache Cassandra との完全な互換性を維持して、Amazon Keyspaces 用に事前設定した一般的な Apache Cassandra ツールなどのヘルパーが含まれています。cqlsh-expansion は、SigV4 認証プラグインを統合しているため、ユーザー名とパスワードを使用する代わりに、IAM アクセスキーで接続できます。Amazon Keyspaces はサーバーレスなので、cqlsh スクリプトをインストールするだけで接続でき、Apache Cassandraディストリビューション一式をインストールする必要はありません。この軽量インストールパッケージには、cqlsh-expansion と、Python をサポートするプラットフォームであればどのプラットフォームにもインストールできるクラシック cqlsh スクリプトが含まれています。

> **Note**
>
> Murmur3Partitioner は、Amazon Keyspaces と cqlsh-expansion に推奨されるパーティショナーです。cqlsh-expansion は Amazon Keyspaces DefaultPartitioner に対応していません。詳細については、「the section called “パーティショナーの操作”」を参照してください。

「cqlsh」の詳細については、「cqlsh: CQL シェル」を参照してください。

トピック

- cqlsh-expansion による Amazon Keyspaces までの接続
- TLS の cqlsh 接続を手動で設定する方法

cqlsh-expansion による Amazon Keyspaces までの接続

cqlsh-expansion のインストールと設定

1. cqlsh-expansion Python パッケージをインストールするには、pip コマンドを実行します。これにより、pip install と依存関係のリストを保存したファイルとともに、cqlsh-expansion スクリプトがマシンにインストールされます。--user flag は、プラットフォームの Python ユーザーインストールディレクトリを使用するように pip に指示します。Unix ベースのシステムでは、それが ~/.local/ ディレクトリになるはずです。

   cqlsh-expansion をインストールするには Python 3 が必要です。使用している Python のバージョンを確認するには、Python --version を使用してください。インストールするには、次のコマンドを実行します。

   ```
   python3 -m pip install --user cqlsh-expansion
   ```

   出力は次のようになります。

   ```
   Collecting cqlsh-expansion
     Downloading cqlsh_expansion-0.9.6-py3-none-any.whl (153 kB)
        ######################################## 153.7/153.7 KB 3.3 MB/s eta 0:00:00
   Collecting cassandra-driver
     Downloading cassandra_driver-3.28.0-cp310-cp310-
   manylinux_2_17_x86_64.manylinux2014_x86_64.whl (19.1 MB)
        ######################################## 19.1/19.1 MB 44.5 MB/s eta 0:00:00
   Requirement already satisfied: six>=1.12.0 in /usr/lib/python3/dist-packages (from
    cqlsh-expansion) (1.16.0)
   Collecting boto3
     Downloading boto3-1.29.2-py3-none-any.whl (135 kB)
        ######################################## 135.8/135.8 KB 17.2 MB/s eta 0:00:00
   Collecting cassandra-sigv4>=4.0.2
     Downloading cassandra_sigv4-4.0.2-py2.py3-none-any.whl (9.8 kB)
   Collecting botocore<1.33.0,>=1.32.2
   ```

```
  Downloading botocore-1.32.2-py3-none-any.whl (11.4 MB)
     ######################################## 11.4/11.4 MB 60.9 MB/s eta 0:00:00
Collecting s3transfer<0.8.0,>=0.7.0
  Downloading s3transfer-0.7.0-py3-none-any.whl (79 kB)
     ######################################## 79.8/79.8 KB 13.1 MB/s eta 0:00:00
Collecting jmespath<2.0.0,>=0.7.1
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Collecting geomet<0.3,>=0.1
  Downloading geomet-0.2.1.post1-py3-none-any.whl (18 kB)
Collecting python-dateutil<3.0.0,>=2.1
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
     ######################################## 247.7/247.7 KB 33.1 MB/s eta 0:00:00
Requirement already satisfied: urllib3<2.1,>=1.25.4 in /usr/lib/python3/dist-
packages (from botocore<1.33.0,>=1.32.2->boto3->cqlsh-expansion) (1.26.5)
Requirement already satisfied: click in /usr/lib/python3/dist-packages (from
 geomet<0.3,>=0.1->cassandra-driver->cqlsh-expansion) (8.0.3)
Installing collected packages: python-dateutil, jmespath, geomet, cassandra-driver,
 botocore, s3transfer, boto3, cassandra-sigv4, cqlsh-expansion
  WARNING: The script geomet is installed in '/home/ubuntu/.local/bin' which is not
 on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this
 warning, use --no-warn-script-location.
  WARNING: The scripts cqlsh, cqlsh-expansion and cqlsh-expansion.init are
 installed in '/home/ubuntu/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this
 warning, use --no-warn-script-location.
Successfully installed boto3-1.29.2 botocore-1.32.2 cassandra-driver-3.28.0
 cassandra-sigv4-4.0.2 cqlsh-expansion-0.9.6 geomet-0.2.1.post1 jmespath-1.0.1
 python-dateutil-2.8.2 s3transfer-0.7.0
```

インストールディレクトリが PATH にない場合は、オペレーティングシステムの指示に従って追加する必要があります。以下は、Ubuntu Linux の場合の一例です。

```
export PATH="$PATH:/home/ubuntu/.local/bin"
```

パッケージがインストールされたことを確認するためには、次のコマンドを実行します。

```
cqlsh-expansion --version
```

出力は次のようになります。

```
cqlsh 6.1.0
```

2. cqlsh-expansion を設定するには、インストール後のスクリプトを実行して次の手順を自動的に完了させることができます。

   1. .cassandra ディレクトリがなければ、ユーザーのホームディレクトリに作成します。
   2. 事前設定済みの cqlshrc 設定ファイルを .cassandra ディレクトリにコピーします。
   3. Starfield デジタル証明書を .cassandra ディレクトリにコピーします。Amazon Keyspaces では、この証明書で、Transport Layer Security (TLS) との安全な接続を設定します。転送時の暗号化では、Amazon Keyspaces との間で送受信するときにデータを暗号化することによって、データ保護のレイヤーを追加します。

   このスクリプトを最初に確認するため、post_install.py にある Github リポジトリからスクリプトにアクセスします。

   スクリプトを使用するには、次のコマンドを実行します。

   ```
   cqlsh-expansion.init
   ```

   > **Note**
   >
   > ポストインストールスクリプトによって作成されるディレクトリとファイルは、pip uninstall で cqlsh-expansion をアンインストールしても削除されないため、手動で削除する必要があります。

cqlsh-expansion を使用した Amazon Keyspaces への接続

1. を設定し AWS リージョン 、ユーザー環境変数として追加します。

   デフォルトリージョンを Unix ベースシステムの環境変数として追加するには、次のコマンドを実行します。この例では、米国東部 (バージニア北部) を使用します

   ```
   export AWS_DEFAULT_REGION=us-east-1
   ```

   他のプラットフォーム用など、環境変数の設定方法の詳細については、「環境変数を設定する方法」を参照してください。

2. サービスエンドポイントを検索します。

   リージョンに適したサービスエンドポイントを選択します。Amazon Keyspaces で使用可能なエンドポイントを確認するには、「the section called “サービスエンドポイント”」を参照してください。この例では、`cassandra.us-east-1.amazonaws.com` エンドポイントを使用します。

3. 認証方法を設定します。

   IAM アクセスキー (IAM ユーザー、ロール、フェデレーテッドアイデンティティ) を使用した接続が、セキュリティ強化のための推奨方法です。

   IAM アクセスキーで続する前に、以下の手順を完了してください。

   a. IAM ユーザーを作成するか、ベストプラクティスに従って IAM ユーザーが引き受けることができる IAM ロールを作成します。IAM アクセスキーの作成方法の詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

   b. Amazon Keyspaces に少なくとも読み取り専用アクセス権限を与えるロール (または IAM ユーザー) を作成します。IAM ユーザーまたはロールが Amazon Keyspaces に接続するために必要な権限の詳細については、「the section called “Amazon Keyspaces テーブルへのアクセス”」を参照してください。

   c. 次の例のように、IAM ユーザーのアクセスキーをユーザーの環境変数に追加します。

   ```
   export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
   export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   ```

   他のプラットフォームなど、環境変数の設定方法の詳細については、「環境変数を設定する方法」を参照してください。

   > **Note**
   >
   > Amazon EC2 インスタンスから接続する場合は、そのインスタンスから Amazon Keyspaces へのトラフィックを許可するアウトバウンドルールをセキュリティグループで設定する必要もあります。EC2 アウトバウンドルールの表示方法と編集方法の詳細については、「Amazon EC2 ユーザーガイド」の「セキュリティグループへのルールの追加」を参照してください。

4. `cqlsh-expansion` と SigV4 認証で Amazon Keyspaces に接続

cqlsh-expansion を使用して Amazon Keyspaces に接続するには、次のコマンドを使用します。必ずサービスエンドポイントを、リージョンに適したエンドポイントに置き換えてください。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

正常に接続されると次の例のような出力が表示されます。

```
Connected to Amazon Keyspaces at cassandra.us-east-1.amazonaws.com:9142
[cqlsh 6.1.0 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh current consistency level is ONE.
cqlsh>
```

接続エラーが発生した場合は、「the section called “cqlsh 接続エラー”」でトラブルシューティング情報をご確認ください。

- サービス固有の認証情報で Amazon Keyspaces に接続します。

  Cassandra が認証に使用する従来のユーザー名とパスワードの組み合わせで接続するには、the section called “サービス固有の認証情報の作成” にあるように、まず Amazon Keyspaces 用のサービス固有の認証情報を作成する必要があります。また、Amazon Keyspaces にアクセスするための権限をそのユーザーに与える必要があります。詳細については、「the section called “Amazon Keyspaces テーブルへのアクセス”」を参照してください。

  ユーザー用にサービス固有の認証情報と権限を作成したら、cqlshrc ファイルを更新します。通常、このファイルはユーザーディレクトリパス ~/.cassandra/ にあります。cqlshrc ファイル内の Cassandra [authentication] セクションに移動し、以下の例のように「;」文字を使用して [auth_provider] の下にある SigV4 モジュールとクラスをコメントアウトします。

  ```
  [auth_provider]

  ; module = cassandra_sigv4.auth

  ; classname = SigV4AuthProvider
  ```

cqlshrc ファイルを更新すると、次のコマンドで Amazon Keyspaces に接続できます。サービス固有の認証情報で Amazon Keyspaces に接続できます。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 -u myUserName -
p myPassword --ssl
```

クリーンアップ

- cqlsh-expansion パッケージを削除するには、pip uninstall コマンドを使用します。

  ```
  pip3 uninstall cqlsh-expansion
  ```

  pip3 uninstall コマンドでは、インストール後のスクリプトによって作成されたディレクトリと関連ファイルは削除されません。ポストインストールスクリプトによって作成されたフォルダとファイルを削除するには、.cassandra ディレクトリを削除します。

TLS の cqlsh 接続を手動で設定する方法

Amazon Keyspaces では、Transport Layer Security (TLS) を使用した安全な接続しか許容されません。cqlsh-expansion ユーティリティを使用できます。これで、証明書を自動的にダウンロードし、事前設定済みの cqlshrc 設定ファイルをインストールします。詳細については、このページの「the section called “cqlsh-expansion の使用”」を参照してください。

証明書をダウンロードして手動で接続を設定する場合は、以下の手順に従います。

1. 次のコマンドを使用して Starfield デジタル証明書をダウンロードし、sf-class2-root.crt をローカルまたはホームディレクトリ内に保存します。

   ```
   curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
   ```

   > **Note**
   >
   > Amazon デジタル証明書を使用して Amazon Keyspaces に接続することもできます。クライアントが Amazon Keyspaces に正常に接続されている場合は、引き続き Amazon Keyspaces に接続できます。Starfield 証明書は、古い認定権限を使用しているクライアントに対して追加の下位互換性を提供するものです。

2. Cassandra ホームディレクトリにある cqlshrc 設定ファイル (${HOME}/.cassandra/cqlshrc など) を開いて、次の行を追加します。

```
[connection]
port = 9142
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
validate = true
certfile =  path_to_file/sf-class2-root.crt
```

を使用して Amazon Keyspaces AWS CLI に接続する

AWS Command Line Interface （ AWS CLI) を使用して、コマンドラインから複数の AWS サービスを制御したり、スクリプトを使用して自動化したりできます。Amazon Keyspaces では、テーブルの作成などのデータ定義言語 (DDL) オペレーション AWS CLI に を使用できます。さらに、Infrastructure as Code (IaC) サービスと、 AWS CloudFormation や Terraform などのツールを使用できます。

Amazon Keyspaces AWS CLI で を使用する前に、アクセスキー ID とシークレットアクセスキーを取得する必要があります。詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

で Amazon Keyspaces で使用できるすべてのコマンドの完全なリストについては AWS CLI、AWS CLI 「 コマンドリファレンス」を参照してください。

トピック

- AWS CLIのダウンロードと設定
- Amazon Keyspaces AWS CLI での の使用

AWS CLIのダウンロードと設定

AWS CLI は で入手できますhttps://aws.amazon.com/cli。Windows、macOS、または Linux 上で実行できます。をダウンロードしたら AWS CLI、以下の手順に従ってインストールして設定します。

1. AWS Command Line Interface ユーザーガイドに移動します。
2. 「AWS CLIのインストールとAWS CLIの設定」の手順に従ってください。

Amazon Keyspaces AWS CLI での の使用

このコマンドラインは、Amazon Keyspaces オペレーション名の後にそのオペレーション用のパラメータが続く形式になっています。は、パラメータ値の短縮構文と JSON AWS CLI をサポートしています。次の Amazon Keyspaces の例では、 AWS CLI 短縮構文を使用しています。詳細については、「 CLI での短縮構文の使用 AWS」を参照してください。

以下のコマンドは、catalog という名前のキースペースを作成します。

```
aws keyspaces create-keyspace --keyspace-name 'catalog'
```

このコマンドは、リソースの Amazon リソースネーム (ARN) を出力に返します。

```
{
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/catalog/"
}
```

キースペースカタログが存在することは、以下のコマンドで確認できます。

```
aws keyspaces get-keyspace --keyspace-name 'catalog'
```

このコマンドの出力で、以下の値が返ります。

```
{
    "keyspaceName": "catalog",
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/catalog/"
}
```

以下のコマンドで book_wards という名前のテーブルが作成されます。テーブルのパーティションキーは year 列と award 列で構成され、クラスタリングキーは category 列と rank 列で構成され、どちらのクラスタリング列も昇順でソートされます。(読みやすくするために、このセクションの長いコマンドは、複数の行に分かれています)。

```
aws keyspaces create-table --keyspace-name 'catalog' --table-name 'book_awards'
            --schema-definition 'allColumns=[{name=year,type=int},
{name=award,type=text},{name=rank,type=int},
            {name=category,type=text}, {name=author,type=text},
{name=book_title,type=text},{name=publisher,type=text}],
            partitionKeys=[{name=year},
{name=award}],clusteringKeys=[{name=category,orderBy=ASC},{name=rank,orderBy=ASC}]'
```

このコマンドで、次の出力が返ります。

```
{
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/catalog/table/
book_awards"
}
```

テーブルのメタデータとプロパティは、以下のコマンドで確認します。

```
aws keyspaces get-table --keyspace-name 'catalog' --table-name 'book_awards'
```

このコマンドで、以下の出力が返ります。

```
{
    "keyspaceName": "catalog",
    "tableName": "book_awards",
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/catalog/table/
book_awards",
    "creationTimestamp": 1645564368.628,
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "year",
                "type": "int"
            },
            {
                "name": "award",
                "type": "text"
            },
            {
                "name": "category",
                "type": "text"
            },
            {
                "name": "rank",
                "type": "int"
            },
            {
                "name": "author",
                "type": "text"
            },
            {
```

```
                "name": "book_title",
                "type": "text"
            },
            {
                "name": "publisher",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "year"
            },
            {
                "name": "award"
            }
        ],
        "clusteringKeys": [
            {
                "name": "category",
                "orderBy": "ASC"
            },
            {
                "name": "rank",
                "orderBy": "ASC"
            }
        ],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": 1645564368.628
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "ttl": {
        "status": "ENABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
```

```
    }
}
```

複雑なスキーマがあるテーブルを作成する場合、JSON ファイルからテーブルのスキーマ定義を読み込むと便利です。次に例を示します。schema_definition.zip からスキーマ定義のサンプル JSON ファイルをダウンロードし、ファイルへのパスを書き留めて schema_definition.json を抽出します。この例では、スキーマ定義 JSON ファイルは現在のディレクトリにあります。さまざまなファイルパスオプションについては、「ファイルからパラメーターを読み込む方法」を参照してください。

```
aws keyspaces create-table --keyspace-name 'catalog'
            --table-name 'book_awards' --schema-definition 'file://
schema_definition.json'
```

次の例は、myTable という名前でオプションを追加した単純なテーブルを作成する方法です。読みやすくするために、コマンドは別々の行に分かれているので注意してください。このコマンドでは、テーブルの作成方法と次の操作を示します。

- テーブルのキャパシティモードを設定する
- テーブルでポイントインタイムリカバリを有効にする
- テーブルのデフォルト有効期限 (TTL) 値を 1 年に設定する
- テーブルに 2 つのタグを追加する

```
aws keyspaces create-table --keyspace-name 'catalog' --table-name 'myTable'
            --schema-definition 'allColumns=[{name=id,type=int},{name=name,type=text},
{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --capacity-specification
 'throughputMode=PROVISIONED,readCapacityUnits=5,writeCapacityUnits=5'
            --point-in-time-recovery 'status=ENABLED'
            --default-time-to-live '31536000'
            --tags 'key=env,value=test' 'key=dpt,value=sec'
```

この例では、暗号化にカスタマーマネージドキーを使用し、列と行の有効期限を設定できるように TTL を有効にした新しいテーブルを作成する方法を説明します。このサンプルを実行するには、カスタマーマネージド AWS KMS キーのリソース ARN を独自のキーに置き換え、Amazon Keyspaces がそのキーにアクセスできることを確認する必要があります。

```
aws keyspaces create-table --keyspace-name 'catalog' --table-name 'myTable'
```

```
            --schema-definition 'allColumns=[{name=id,type=int},{name=name,type=text},
{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --encryption-specification
 'type=CUSTOMER_MANAGED_KMS_KEY,kmsKeyIdentifier=arn:aws:kms:us-
east-1:111222333444:key/11111111-2222-3333-4444-555555555555'
            --ttl 'status=ENABLED'
```

API を使用して Amazon Keyspaces に接続する

AWS SDK と AWS Command Line Interface （ AWS CLI) を使用して、Amazon Keyspaces とインタラクティブに連携できます。API で、キースペースやテーブルの作成など、データ言語定義 (DDL) の操作を同期なしで実行します。さらに、Infrastructure as Code (IaC) サービスと、 AWS CloudFormation や Terraform などのツールを使用できます。

Amazon Keyspaces AWS CLI で を使用する前に、アクセスキー ID とシークレットアクセスキーを取得する必要があります。詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

API の Amazon キースペースで使用できるすべてのオペレーションの完全なリストについては、「Amazon Keyspaces API リファレンス」を参照してください。

Cassandra クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス

Amazon Keyspaces への接続には、サードパーティー製の多数のオープンソース Cassandra ドライバーを使用できます。Amazon Keyspaces は、Apache Cassandra バージョン 3.11.2 に対応している Cassandra ドライバーと互換性があります。以下に、Amazon Keyspaces での使用が推奨されるテスト済みのドライバーと最新バージョンを紹介します。

- `Java v3.3`
- `Java v4.17`
- `Python Cassandra-driver 3.29.1`
- `Node.js cassandra driver -v 4.7.2`
- `GO using GOCQL v1.6`
- `.NET CassandraCSharpDriver -v 3.20.1`

Cassandra ドライバーの詳細については、「Apache Cassandra Client drivers(Apache Cassandra クライアントドライバー)」を参照してください。

> **Note**
>
> 円滑に使用を開始するために、一般的なドライバーを使用して Amazon Keyspaces への接続を確立するエンドツーエンドのコード例を確認してダウンロードすることができます。GitHub で「Amazon Keyspaces examplesmazon Keyspaces の例」を参照してください。

この章のチュートリアルには、Amazon Keyspaces への接続が正常に確立されたことを確認するための簡易 CQL クエリが含まれています。Amazon Keyspaces エンドポイントへの接続後にキースペースとテーブルを操作する方法については、「CQL 言語リファレンス」を参照してください。Amazon VPC エンドポイントから Amazon Keyspaces に接続する方法を示すステップバイステップチュートリアルについては、「the section called “VPC エンドポイントによる接続”」を参照してください。

トピック

- Cassandra Java クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス
- Cassandra Python クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス
- Cassandra Node.js クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス
- Cassandra .NET Core クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス
- Cassandra Go クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス
- Cassandra Perl クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス

Cassandra Java クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス

このセクションでは、Java クライアントドライバーを使用して Amazon Keyspaces に接続する方法について説明します。

> **Note**
>
> Java 17 と DataStax Java ドライバー 4.17 は、現時点ではベータサポートのみです。詳細については、「https://docs.datastax.com/en/developer/java-driver/4.17/upgrade_guide/」を参照してください。

Amazon Keyspaces リソースへのプログラムアクセスに必要な認証情報を、ユーザーとアプリケーションに提供するには、次のいずれかを実行します。

- 特定の AWS Identity and Access Management (IAM) ユーザーに関連付けられたサービス固有の認証情報を作成します。
- セキュリティを強化するために、すべての AWS サービスで使用される IAM ID の IAM アクセスキーを作成することをお勧めします。Cassandra クライアントドライバー用の Amazon Keyspaces SigV4 認証プラグインを使用すると、ユーザー名とパスワードではなく IAM アクセスキーを使用して Amazon Keyspaces のコールの認証を行うことができます。詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

> **Note**
>
> Spring Boot で Amazon Keyspaces を使用する方法の例については、「https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/spring」を参照してください。

トピック

- [開始する前に]
- サービス固有の認証情報を使用して、Apache Cassandra 用の DataStax Java ドライバーで Amazon Keyspaces に接続するためのステップバイステップチュートリアル
- Apache Cassandra 用の 4.x DataStax Java ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続するためのステップバイステップのチュートリアル
- Apache Cassandra 用 3.x DataStax Java ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続する

[開始する前に]

Amazon Keyspaces への接続を開始する前に、以下のタスクを行う必要があります。

1. Amazon Keyspaces では、クライアントとの安全な接続を確保するために Transport Layer Security (TLS) を使用する必要があります。

   a. 次のコマンドを使用して Starfield デジタル証明書をダウンロードし、`sf-class2-root.crt` をローカルまたはホームディレクトリ内に保存します。

   ```
   curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
   ```

   > **Note**
   >
   > Amazon デジタル証明書を使用して Amazon Keyspaces に接続することもできます。クライアントが Amazon Keyspaces に正常に接続されている場合は、引き続き Amazon Keyspaces に接続できます。Starfield 証明書は、古い認定権限を使用しているクライアントに対して追加の下位互換性を提供するものです。

   b. Starfield デジタル証明書を trustStore ファイルに変換します。

   ```
   openssl x509 -outform der -in sf-class2-root.crt -out temp_file.der
   keytool -import -alias cassandra -keystore cassandra_truststore.jks -file
    temp_file.der
   ```

   このステップでは、キーストアのパスワードを作成し、この証明書を信頼する必要があります。対話型コマンドは次のようになります。

   ```
   Enter keystore password:
   Re-enter new password:
   Owner: OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
    Inc.", C=US
   Issuer: OU=Starfield Class 2 Certification Authority, O="Starfield
    Technologies, Inc.", C=US
   Serial number: 0
   Valid from: Tue Jun 29 17:39:16 UTC 2004 until: Thu Jun 29 17:39:16 UTC 2034
   Certificate fingerprints:
     MD5:  32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
     SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
   ```

```
  SHA256:
 14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB
Signature algorithm name: SHA1withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
Extensions:
#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86   F4 5B 55 AC DC D7 10 C2  ._.......[U.....
0010: 0E A9 88 E7                                        ....
]
[OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
 Inc.", C=US]
SerialNumber: [    00]
]
#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen:2147483647
]
#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86   F4 5B 55 AC DC D7 10 C2  ._.......[U.....
0010: 0E A9 88 E7                                        ....
]
]
Trust this certificate? [no]:  y
```

2. JVM 引数に TrustStore ファイルをアタッチします。

```
-Djavax.net.ssl.trustStore=path_to_file/cassandra_truststore.jks
-Djavax.net.ssl.trustStorePassword=my_password
```

サービス固有の認証情報を使用して、Apache Cassandra 用の DataStax Java ドライバーで Amazon Keyspaces に接続するためのステップバイステップチュートリアル

以下のステップバイステップチュートリアルでは、サービス固有の認証情報を使用して Cassandra 用 Java ドライバーで Amazon Keyspaces に接続する手順を説明します。具体的には、Apache Cassandra 用の DataStax Java ドライバーのバージョン 4.0 を使用します。

トピック

- ステップ 1: 前提条件
- ステップ 2: ドライバーを設定する
- ステップ 3: サンプルアプリケーションを実行する

ステップ 1: 前提条件

このチュートリアルに従って作業を行うには、サービス固有の認証情報を生成し、Apache Cassandra 用の DataStax Java ドライバーを Java プロジェクトに追加しておく必要があります。

- 「the section called “サービス固有の認証情報の作成”」の手順を完了することで、Amazon Keyspaces IAM ユーザー向けにサービス固有の認証情報を生成します。認証に IAM アクセスキーを使用する場合は、「the section called “Java 4.x の認証プラグイン”」を参照してください。
- Apache Cassandra 用の DataStax Java ドライバーを Java プロジェクトに追加します。Apache Cassandra 3.11.2 に対応しているドライバーバージョンを使用していることを確認します。詳細については、「DataStax Java driver for Apache Cassandra documentation(Apache Cassandra 用 DataStax Java ドライバーのドキュメント)」を参照してください。

ステップ 2: ドライバーを設定する

アプリケーション用の設定ファイルを作成すれば、DataStax Java Cassandra ドライバーの設定を指定できます。この設定ファイルは、デフォルト設定をオーバーライドし、ポート 9142 を使用して Amazon Keyspaces サービスエンドポイントに接続するようにドライバーに指示を与えます。利用可能なサービスエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

設定ファイルを作成し、アプリケーションのリソースフォルダ (例: `src/main/resources/application.conf`) に保存します。`application.conf` を開き、次の設定を追加します。

1. 認証プロバイダ — `PlainTextAuthProvider` クラスを使用して認証プロバイダを作成します。*`ServiceUserName`* と *`ServicePassword`* は、Amazon Keyspaces にプログラムによってアクセスするためのサービス固有の認証情報を作成する の手順に従ってサービス固有の認証情報を生成したときに取得したユーザー名とパスワードと一致します。

> **Note**
>
> 短期の認証情報を使用するには、ドライバー設定ファイルの認証情報をハードコードする代わりに、Apache Cassandra 用 DataStax Java ドライバーの認証プラグインを使用します。詳細を把握するには、「the section called “Java 4.x の認証プラグイン”」の指示に従ってください。

2. ローカルデータセンター — `local-datacenter` の値を、接続先のリージョンに設定します。例えば、アプリケーションを `cassandra.us-east-2.amazonaws.com` に接続する場合は、ローカルデータセンターを `us-east-2` に設定します。すべての利用可能な AWS リージョンについては「???」を参照してください。。負荷分散の対象となるノードの数を減らすように `slow-replica-avoidance = false` を設定します。
3. SSL/TLS — `class = DefaultSslEngineFactory` があるクラスを指定する 1 行を含む設定ファイルにセクションを追加することで、SSlengineFactory を初期化します。trustStore ファイルへのパスと、以前に作成したパスワードを提供します。Amazon Keyspaces はピアの `hostname-validation` をサポートしていないため、このオプションを false に設定してください。

```
datastax-java-driver {

    basic.contact-points = [ "cassandra.us-east-2.amazonaws.com:9142"]
    advanced.auth-provider{
        class = PlainTextAuthProvider
        username = "ServiceUserName"
        password = "ServicePassword"
    }
    basic.load-balancing-policy {
        local-datacenter = "us-east-2"
        slow-replica-avoidance = false
    }

    advanced.ssl-engine-factory {
        class = DefaultSslEngineFactory
        truststore-path = "./src/main/resources/cassandra_truststore.jks"
        truststore-password = "my_password"
        hostname-validation = false
      }
}
```

> **Note**
>
> trustStore へのパスを設定ファイル追加する代わりに、trustStore パスをアプリケーションコード直接追加することも、trustStore へのパスを JVM 引数に追加することもできます。

ステップ 3: サンプルアプリケーションを実行する

このコード例は、先ほど作成した設定ファイルを使用して Amazon Keyspaces への接続プールを作成する単純なコマンドラインアプリケーションを示しています。これは、単純なクエリを実行することで、接続が確立されたことを確認するものです。

```
package <your package>;
// add the following imports to your project
import com.datastax.oss.driver.api.core.CqlSession;
import com.datastax.oss.driver.api.core.config.DriverConfigLoader;
import com.datastax.oss.driver.api.core.cql.ResultSet;
import com.datastax.oss.driver.api.core.cql.Row;

public class App
{

    public static void main( String[] args )
    {
        //Use DriverConfigLoader to load your configuration file
        DriverConfigLoader loader =
 DriverConfigLoader.fromClasspath("application.conf");
        try (CqlSession session = CqlSession.builder()
                .withConfigLoader(loader)
                .build()) {

            ResultSet rs = session.execute("select * from system_schema.keyspaces");
            Row row = rs.one();
            System.out.println(row.getString("keyspace_name"));
        }
    }
}
```

> **Note**
>
> try ブロックを使用して接続を確立し、その接続が常に閉じていることを確認します。try ブロックを使用しない場合は、リソースの漏洩を防ぐために必ず接続を閉じてください。

Apache Cassandra 用の 4.x DataStax Java ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続するためのステップバイステップのチュートリアル

次のセクションでは、Apache Cassandra 用のオープンソース 4.x DataStax Java ドライバーに SigV4 認証プラグインを使用して、Amazon Keyspaces (Apache Cassandra 向け) にアクセスする方法について説明します。このプラグインは GitHub リポジトリから入手できます。

この SigV4 認証プラグインを使用すると、Amazon Keyspaces に接続するときに、ユーザーまたはロールの IAM 認証情報を使用できます。このプラグインは、ユーザー名とパスワードを要求する代わりに、アクセスキーを使用して API リクエストに署名します。詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

ステップ 1: 前提条件

このチュートリアルを実行するには、次のタスクを完了する必要があります。

- まだ完了していない場合は、the section called “ AWS 認証用の IAM 認証情報を作成する” のステップに従って IAM ユーザーまたはロールの認証情報を作成します。このチュートリアルでは、アクセスキーが環境変数として保存されることを前提としています。詳細については、「the section called “ アクセスキーを管理する”」を参照してください。
- Apache Cassandra 用の DataStax Java ドライバーを Java プロジェクトに追加します。Apache Cassandra 3.11.2 に対応しているドライバーバージョンを使用していることを確認します。詳細については、「DataStax Java Driver for Apache Cassandra documentation (Apache Cassandra 用 DataStax Java ドライバーのドキュメント)」を参照してください。
- 認証プラグインをアプリケーションに追加します。認証プラグインは、Apache Cassandra 用 DataStax Java ドライバーのバージョン 4.x に対応しています。Apache Maven を使用している場合、または Maven の依存関係を使用できるビルドシステムを使用している場合は、次の依存関係を pom.xml ファイルに追加します。

> **Important**
>
> GitHub リポジトリに示されているように、プラグインのバージョンを最新バージョンに置き換えます。

```
<dependency>
        <groupId>software.aws.mcs</groupId>
        <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin</artifactId>
        <version>4.0.9</version>
</dependency>
```

ステップ 2: ドライバーを設定する

アプリケーション用の設定ファイルを作成すれば、DataStax Java Cassandra ドライバーの設定を指定できます。この設定ファイルは、デフォルト設定をオーバーライドし、ポート 9142 を使用して Amazon Keyspaces サービスエンドポイントに接続するようにドライバーに指示を与えます。利用可能なサービスエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

設定ファイルを作成し、アプリケーションのリソースフォルダ (例: `src/main/resources/application.conf`) に保存します。`application.conf` を開き、次の設定を追加します。

1. 認証プロバイダ — `advanced.auth-provider.class` を `software.aws.mcs.auth.SigV4AuthProvider` の新しいインスタンスに設定します。SigV4AuthProvider は、SigV4 認証を実行するためのプラグインによって提供される認証ハンドラです。
2. ローカルデータセンター — `local-datacenter` の値を、接続先のリージョンに設定します。例えば、アプリケーションを `cassandra.us-east-2.amazonaws.com` に接続する場合は、ローカルデータセンターを `us-east-2` に設定します。使用可能なすべての については AWS リージョン、「」を参照してください???。利用可能なすべてのノードで負荷を分散させるには、`slow-replica-avoidance = false` を設定します。
3. Idempotence – アプリケーションのデフォルトの `idempotence` を `true` に設定して、読み取り/書き込み/準備/実行のリクエストが失敗した場合に常に再試行するようにドライバーを設定します。これは、分散型アプリケーションではベストプラクティスであり、失敗したリクエストを再試行することで一時的な障害に対処できます。

4. SSL/TLS — `class = DefaultSslEngineFactory` があるクラスを指定する 1 行を含む設定ファイルにセクションを追加することで、SSlengineFactory を初期化します。trustStore ファイルへのパスと、以前に作成したパスワードを提供します。Amazon Keyspaces はピアの `hostname-validation` をサポートしていないため、このオプションを false に設定してください。
5. 接続数 – `local.size = 3` を設定して、エンドポイントごとに 3 つ以上のローカル接続を作成します。これは、アプリケーションがオーバーヘッドやトラフィックのバーストに対処するうえで役立つベストプラクティスです。予想されるトラフィックパターンに基づいて、アプリケーションが必要とするローカル接続の数をエンドポイントごとに計算する方法については、「the section called “接続を設定する方法”」を参照してください。
6. 再試行ポリシー – Cassandra ドライバーに付属の `DefaultRetryPolicy` ではなく、Amazon Keyspaces の再試行ポリシー `AmazonKeyspacesExponentialRetryPolicy` を実装します。これにより、`AmazonKeyspacesExponentialRetryPolicy` の再試行回数をニーズに応じて設定できます。デフォルトでは、`AmazonKeyspacesExponentialRetryPolicy` の再試行回数は 3 に設定されています。詳細については、「the section called “再試行ポリシーを設定する方法”」を参照してください。
7. プリペアドステートメント – ネットワーク使用量を最適化するため、`prepare-on-all-nodes` を false に設定します。

```
datastax-java-driver {
    basic {
        contact-points = [ "cassandra.us-east-2.amazonaws.com:9142"]
        request {
            timeout = 2 seconds
            consistency = LOCAL_QUORUM
            page-size = 1024
            default-idempotence = true
        }
        load-balancing-policy {
            local-datacenter = "us-east-2"
            class = DefaultLoadBalancingPolicy
            slow-replica-avoidance = false
        }
    }
    advanced {
        auth-provider {
            class = software.aws.mcs.auth.SigV4AuthProvider
            aws-region = us-east-2
```

```
        }
        ssl-engine-factory {
            class = DefaultSslEngineFactory
            truststore-path = "./src/main/resources/cassandra_truststore.jks"
            truststore-password = "my_password"
            hostname-validation = false
        }
        connection {
      connect-timeout = 5 seconds
      max-requests-per-connection = 512
      pool {
                local.size = 3
      }
        }
       retry-policy {
           class =  com.aws.ssa.keyspaces.retry.AmazonKeyspacesExponentialRetryPolicy
     max-attempts = 3
     min-wait = 10 mills
     max-wait = 100 mills
        }
        prepared-statements {
     prepare-on-all-nodes = false
        }
    }
}
```

> **Note**
>
> trustStore へのパスを設定ファイル追加する代わりに、trustStore パスをアプリケーションコード直接追加することも、trustStore へのパスを JVM 引数に追加することもできます。

ステップ 3: アプリケーションを実行する

このコード例は、先ほど作成した設定ファイルを使用して Amazon Keyspaces への接続プールを作成する単純なコマンドラインアプリケーションを示しています。これは、単純なクエリを実行することで、接続が確立されたことを確認するものです。

```
package <your package>;
// add the following imports to your project
import com.datastax.oss.driver.api.core.CqlSession;
import com.datastax.oss.driver.api.core.config.DriverConfigLoader;
```

```
import com.datastax.oss.driver.api.core.cql.ResultSet;
import com.datastax.oss.driver.api.core.cql.Row;

public class App
{

    public static void main( String[] args )
    {
        //Use DriverConfigLoader to load your configuration file
        DriverConfigLoader loader =
 DriverConfigLoader.fromClasspath("application.conf");
        try (CqlSession session = CqlSession.builder()
                .withConfigLoader(loader)
                .build()) {

            ResultSet rs = session.execute("select * from system_schema.keyspaces");
            Row row = rs.one();
            System.out.println(row.getString("keyspace_name"));
        }
    }
}
```

> **Note**
>
> try ブロックを使用して接続を確立し、その接続が常に閉じていることを確認します。try ブロックを使用しない場合は、リソースの漏洩を防ぐために必ず接続を閉じてください。

Apache Cassandra 用 3.x DataStax Java ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続する

次のセクションでは、Apache Cassandra 用 3.x オープンソース DataStax Java ドライバーの SigV4 認証プラグインを使用して Amazon Keyspaces にアクセスする方法について説明します。このプラグインは GitHub リポジトリから入手できます。

SigV4 認証プラグインを使用すると、Amazon Keyspaces に接続するときに、ユーザーとロールの IAM 認証情報を使用できます。このプラグインは、ユーザー名とパスワードを要求する代わりに、アクセスキーを使用して API リクエストに署名します。詳細については、「the section called “AWS 認証用の IAM 認証情報を作成する”」を参照してください。

ステップ 1: 前提条件

このコードサンプルを実行するには、まず以下のタスクを完了する必要があります。

- 「the section called “ AWS 認証用の IAM 認証情報を作成する”」の手順に従って、IAM ユーザーまたはロールの認証情報を作成します。このチュートリアルでは、アクセスキーが環境変数として保存されることを前提としています。詳細については、「the section called “ アクセスキーを管理する”」を参照してください。
- the section called “[開始する前に]” のステップに従って Starfield デジタル証明書をダウンロードし、それを trustStore ファイルに変換して、JVM 引数の trustStore ファイルをアプリケーションにアタッチします。
- Apache Cassandra 用の DataStax Java ドライバーを Java プロジェクトに追加します。Apache Cassandra 3.11.2 に対応しているドライバーバージョンを使用していることを確認します。詳細については、「DataStax Java Driver for Apache Cassandra documentation (Apache Cassandra 用 DataStax Java ドライバーのドキュメント)」を参照してください。
- 認証プラグインをアプリケーションに追加します。認証プラグインは、Apache Cassandra 用 DataStax Java ドライバーのバージョン 3.x に対応しています。Apache Maven を使用している場合、または Maven の依存関係を使用できるビルドシステムを使用している場合は、次の依存関係を `pom.xml` ファイルに追加します。GitHub リポジトリに示されているように、プラグインのバージョンを最新バージョンに置き換えます。

```
<dependency>
        <groupId>software.aws.mcs</groupId>
        <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin_3</artifactId>
        <version>3.0.3</version>
</dependency>
```

ステップ 2: アプリケーションを実行する

このコード例は、Amazon Keyspaces への接続プールを作成する単純なコマンドラインアプリケーションを示しています。これは、単純なクエリを実行することで、接続が確立されたことを確認するものです。

```
package <your package>;
// add the following imports to your project

import software.aws.mcs.auth.SigV4AuthProvider;
import com.datastax.driver.core.Cluster;
```

```
import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Row;
import com.datastax.driver.core.Session;

public class App
{

    public static void main( String[] args )
    {
        String endPoint = "cassandra.us-east-2.amazonaws.com";
        int portNumber = 9142;
        Session session = Cluster.builder()
                                 .addContactPoint(endPoint)
                                 .withPort(portNumber)
                                 .withAuthProvider(new SigV4AuthProvider("us-east-2"))

                                 .withSSL()
                                 .build()
                                 .connect();

        ResultSet rs = session.execute("select * from system_schema.keyspaces");
        Row row = rs.one();
        System.out.println(row.getString("keyspace_name"));
    }
}
```

使用に関する注意事項:

利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

Java ドライバーを Amazon Keyspaces で使用する場合に役立つ Java ドライバーポリシー (https://github.com/aws-samples/amazon-keyspaces-java-driver-helpers) およびベストプラクティスについては、次のリポジトリを参照してください。

Cassandra Python クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス

このセクションでは、Python クライアントドライバーを使用して Amazon Keyspaces に接続する方法について説明します。Amazon Keyspaces リソースへのプログラムアクセスに必要な認証情報を、ユーザーとアプリケーションに提供するには、次のいずれかを実行します。

- 特定の AWS Identity and Access Management (IAM) ユーザーに関連付けられたサービス固有の認証情報を作成します。
- セキュリティを強化するために、すべての AWS サービスで使用される IAM ユーザーまたはロールの IAM アクセスキーを作成することをお勧めします。Cassandra クライアントドライバー用の Amazon Keyspaces SigV4 認証プラグインを使用すると、ユーザー名とパスワードではなく IAM アクセスキーを使用して Amazon Keyspaces のコールの認証を行うことができます。詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

トピック

- [開始する前に]
- Apache Cassandra 用の Python ドライバーとサービス固有の認証情報を使用して Amazon Keyspaces に接続する
- Apache Cassandra 用 DataStax Java ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続する

[開始する前に]

開始する前に、次のタスクを完了する必要があります。

Amazon Keyspaces では、クライアントとの安全な接続を確保するために Transport Layer Security (TLS) を使用する必要があります。TLS を使用して Amazon Keyspaces に接続するには、Amazon デジタル証明書をダウンロードし、TLS を使用するように Python ドライバーを設定する必要があります。

次のコマンドを使用して Starfield デジタル証明書をダウンロードし、`sf-class2-root.crt` をローカルまたはホームディレクトリ内に保存します。

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

> **Note**
>
> Amazon デジタル証明書を使用して Amazon Keyspaces に接続することもできます。クライアントが Amazon Keyspaces に正常に接続されている場合は、引き続き Amazon Keyspaces に接続できます。Starfield 証明書は、古い認定権限を使用しているクライアントに対して追加の下位互換性を提供するものです。

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

Apache Cassandra 用の Python ドライバーとサービス固有の認証情報を使用して Amazon Keyspaces に接続する

次のコード例では、Python クライアントドライバーとサービス固有の認証情報を使用して Amazon Keyspaces に接続する方法を示します。

```
from cassandra.cluster import Cluster
from ssl import SSLContext, PROTOCOL_TLSv1_2 , CERT_REQUIRED
from cassandra.auth import PlainTextAuthProvider

ssl_context = SSLContext(PROTOCOL_TLSv1_2 )
ssl_context.load_verify_locations('path_to_file/sf-class2-root.crt')
ssl_context.verify_mode = CERT_REQUIRED
auth_provider = PlainTextAuthProvider(username='ServiceUserName',
 password='ServicePassword')
cluster = Cluster(['cassandra.us-east-2.amazonaws.com'], ssl_context=ssl_context,
 auth_provider=auth_provider, port=9142)
session = cluster.connect()
r = session.execute('select * from system_schema.keyspaces')
print(r.current_rows)
```

使用に関する注意事項:

1. "*path_to_file*/sf-class2-root.crt" を、最初のステップで保存した証明書へのパスに置き換えてください。
2. *ServiceUserName* と *ServicePassword* が、Amazon Keyspaces にプログラムによってアクセスするためのサービス固有の認証情報を作成する の手順に従ってサービス固有の認証情報を生成したときに取得したユーザー名とパスワードと一致していることを確認してください。
3. 利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

Apache Cassandra 用 DataStax Java ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続する

次のセクションでは、Apache Cassandra 用オープンソース DataStax Python ドライバーの SigV4 認証プラグインを使用して、Amazon Keyspaces (Apache Cassandra 向け) にアクセスする方法を示します。

まだ完了していない場合は、the section called “ AWS 認証用の IAM 認証情報を作成する” のステップに従って IAM ロールの認証情報の作成を開始します。このチュートリアルでは、IAM ロールを必要とする一時認証情報を使用します。一時認証情報の詳細については、「the section called “Amazon Keyspaces への接続用の一時的な認証情報の作成”」を参照してください。

次に、Python SigV4 認証プラグインを GitHub リポジトリから環境に追加します。

```
pip install cassandra-sigv4
```

次のコード例では、Cassandra 用のオープンソース DataStax Python ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続する方法を示します。プラグインは AWS SDK for Python (Boto3) によって異なります。boto3.session を使用して一時認証情報を取得します。

```
from cassandra.cluster import Cluster
from ssl import SSLContext, PROTOCOL_TLSv1_2 , CERT_REQUIRED
from cassandra.auth import PlainTextAuthProvider
import boto3
from cassandra_sigv4.auth import SigV4AuthProvider

ssl_context = SSLContext(PROTOCOL_TLSv1_2)
ssl_context.load_verify_locations('path_to_file/sf-class2-root.crt')
ssl_context.verify_mode = CERT_REQUIRED

# use this if you want to use Boto to set the session parameters.
boto_session = boto3.Session(aws_access_key_id="AKIAIOSFODNN7EXAMPLE",
                             aws_secret_access_key="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY",
                             aws_session_token="AQoDYXdzEJr...<remainder of token>",
                             region_name="us-east-2")
auth_provider = SigV4AuthProvider(boto_session)

# Use this instead of the above line if you want to use the Default Credentials and not
 bother with a session.
# auth_provider = SigV4AuthProvider()
```

```
cluster = Cluster(['cassandra.us-east-2.amazonaws.com'], ssl_context=ssl_context,
 auth_provider=auth_provider,
                  port=9142)
session = cluster.connect()
r = session.execute('select * from system_schema.keyspaces')
print(r.current_rows)
```

使用に関する注意事項:

1. "*path_to_file*/sf-class2-root.crt" を、最初のステップで保存した証明書へのパスに置き換えてください。
2. *aws_access_key_id*、*aws_secret_access_key*、*aws_session_token* が、boto3.session を使用して入手した Access Key、Secret Access Key、Session Token に一致していることを確認します。詳細については、AWS SDK for Python (Boto3) の「Credentials」(認証情報) を参照してください。
3. 利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

Cassandra Node.js クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス

このセクションでは、Node.js クライアントドライバーを使用して Amazon Keyspaces に接続する方法を説明します。Amazon Keyspaces リソースへのプログラムアクセスに必要な認証情報を、ユーザーとアプリケーションに提供するには、次のいずれかを実行します。

- 特定の AWS Identity and Access Management (IAM) ユーザーに関連付けられたサービス固有の認証情報を作成します。
- セキュリティを強化するために、すべての AWS サービスで使用される IAM ユーザーまたはロールの IAM アクセスキーを作成することをお勧めします。Cassandra クライアントドライバー用の Amazon Keyspaces SigV4 認証プラグインを使用すると、ユーザー名とパスワードではなく IAM アクセスキーを使用して Amazon Keyspaces のコールの認証を行うことができます。詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

トピック

- [開始する前に]

- Apache Cassandra 用の Node.js DataStax ドライバーとサービス固有の認証情報を使用して Amazon Keyspaces に接続する
- Apache Cassandra 用 DataStax Node.js ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続する

[開始する前に]

開始する前に、次のタスクを完了する必要があります。

Amazon Keyspaces では、クライアントとの安全な接続を確保するために Transport Layer Security (TLS) を使用する必要があります。TLS を使用して Amazon Keyspaces に接続するには、Amazon デジタル証明書をダウンロードし、TLS を使用するように Python ドライバーを設定する必要があります。

次のコマンドを使用して Starfield デジタル証明書をダウンロードし、`sf-class2-root.crt` をローカルまたはホームディレクトリ内に保存します。

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

> **Note**
>
> Amazon デジタル証明書を使用して Amazon Keyspaces に接続することもできます。クライアントが Amazon Keyspaces に正常に接続されている場合は、引き続き Amazon Keyspaces に接続できます。Starfield 証明書は、古い認定権限を使用しているクライアントに対して追加の下位互換性を提供するものです。


```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```


Apache Cassandra 用の Node.js DataStax ドライバーとサービス固有の認証情報を使用して Amazon Keyspaces に接続する

TLS に Starfield デジタル証明書を使用し、認証にサービス固有の認証情報を使用するようにドライバーを設定します。例:

```
const cassandra = require('cassandra-driver');
const fs = require('fs');
```

```
const auth = new cassandra.auth.PlainTextAuthProvider('ServiceUserName',
 'ServicePassword');
const sslOptions1 = {
         ca: [
                    fs.readFileSync('path_to_file/sf-class2-root.crt', 'utf-8')],
                    host: 'cassandra.us-west-2.amazonaws.com',
                    rejectUnauthorized: true
        };
const client = new cassandra.Client({
                   contactPoints: ['cassandra.us-west-2.amazonaws.com'],
                   localDataCenter: 'us-west-2',
                   authProvider: auth,
                   sslOptions: sslOptions1,
                   protocolOptions: { port: 9142 }
        });
const query = 'SELECT * FROM system_schema.keyspaces';

client.execute(query)
                   .then( result => console.log('Row from Keyspaces %s',
 result.rows[0]))
                   .catch( e=> console.log(`${e}`));
```

使用に関する注意事項:

1. "*path_to_file*/sf-class2-root.crt" を、最初のステップで保存した証明書へのパスに置き換えてください。
2. *ServiceUserName* と *ServicePassword* が、Amazon Keyspaces にプログラムによってアクセスするためのサービス固有の認証情報を作成する の手順に従ってサービス固有の認証情報を生成したときに取得したユーザー名とパスワードと一致していることを確認してください。
3. 利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

Apache Cassandra 用 DataStax Node.js ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続する

次のセクションでは、Apache Cassandra 用オープンソース DataStax Node.js ドライバーの SigV4 認証プラグインを使用して、Amazon Keyspaces (Apache Cassandra 向け) にアクセスする方法を示します。

まだ完了していない場合は、the section called “ AWS 認証用の IAM 認証情報を作成する” のステップに従って IAM ユーザーまたはロールの認証情報を作成します。

Node.js SigV4 認証プラグインを GitHub リポジトリからアプリケーションに追加します。プラグインは、Cassandra 用の DataStax Node.js ドライバーのバージョン 4.x をサポートし、 AWS SDK for Node.js に依存します。認証情報の取得には `AWSCredentialsProvider` が使用されます。

```
$ npm install aws-sigv4-auth-cassandra-plugin --save
```

次のコード例で、`SigV4AuthProvider` のリージョン固有のインスタンスを認証プロバイダーとして設定する方法について説明します。

```
const cassandra = require('cassandra-driver');
const fs = require('fs');
const sigV4 = require('aws-sigv4-auth-cassandra-plugin');

const auth = new sigV4.SigV4AuthProvider({
    region: 'us-west-2',
    accessKeyId:'AKIAIOSFODNN7EXAMPLE',
    secretAccessKey: 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY'});

const sslOptions1 = {
  ca: [
      fs.readFileSync('path_to_filecassandra/sf-class2-root.crt', 'utf-8')],
  host: 'cassandra.us-west-2.amazonaws.com',
  rejectUnauthorized: true
};


const client = new cassandra.Client({
  contactPoints: ['cassandra.us-west-2.amazonaws.com'],
  localDataCenter: 'us-west-2',
  authProvider: auth,
  sslOptions: sslOptions1,
  protocolOptions: { port: 9142 }
});


const query = 'SELECT * FROM system_schema.keyspaces';

client.execute(query).then(
    result => console.log('Row from Keyspaces %s', result.rows[0]))
```

```
    .catch( e=> console.log(`${e}`));
```

使用に関する注意事項:

1. "*path_to_file*/sf-class2-root.crt" を、最初のステップで保存した証明書へのパスに置き換えてください。
2. *accessKeyId* と *secretAccessKey* が、AWSCredentialsProvider を使用して取得したアクセスキーとシークレットアクセスキーに一致していることを確認してください。詳細については、「AWS SDK for JavaScript in Node.js」(Node.js での JavaScript の SDK) の「Setting Credentials in Node.js」(Node.js での認証情報の設定) を参照してください。
3. アクセスキーをコード外に保存するには、the section called “ アクセスキーを管理する” のベストプラクティスを参照してください。
4. 利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

Cassandra .NET Core クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス

このセクションでは、.NET Core クライアントドライバーを使用して Amazon Keyspaces に接続する方法について説明します。セットアップ手順が環境やオペレーティングシステムによって異なるため、状況によっては手順の変更が必要になることがあります。Amazon Keyspaces では、クライアントとの安全な接続を確保するために Transport Layer Security (TLS) を使用する必要があります。TLS を使用して Amazon Keyspaces に接続するには、Starfield デジタル証明書をダウンロードし、TLS が使用されるようにドライバーを設定する必要があります。

1. Starfield 証明書をダウンロードしてローカルディレクトリに保存します。その際にパスを書き留めておいてください。以下の例では PowerShell を使用します。

```
$client = new-object System.Net.WebClient
$client.DownloadFile("https://certs.secureserver.net/repository/sf-class2-
root.crt","path_to_file\sf-class2-root.crt")
```

2. NuGet コンソールを使用して、NuGet から CassandraCSharpDriver をインストールします。

```
PM> Install-Package CassandraCSharpDriver
```

3. 次の例では、.NET Core C# コンソールプロジェクトを使用して Amazon Keyspaces に接続し、クエリを実行します。

```
using Cassandra;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Security;
using System.Runtime.ConstrainedExecution;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace CSharpKeyspacesExample
{
    class Program
    {
        public Program(){}

        static void Main(string[] args)
        {
            X509Certificate2Collection certCollection = new
 X509Certificate2Collection();
            X509Certificate2 amazoncert = new X509Certificate2(@"path_to_file\sf-
class2-root.crt");
            var userName = "ServiceUserName";
            var pwd = "ServicePassword";
            certCollection.Add(amazoncert);

            var awsEndpoint =  "cassandra.us-east-2.amazonaws.com" ;

            var cluster = Cluster.Builder()
                     .AddContactPoints(awsEndpoint)
                     .WithPort(9142)
                     .WithAuthProvider(new PlainTextAuthProvider(userName, pwd))
                     .WithSSL(new
 SSLOptions().SetCertificateCollection(certCollection))
                     .Build();

            var session = cluster.Connect();
            var rs = session.Execute("SELECT * FROM system_schema.tables;");
            foreach (var row in rs)
            {
```

```
                var name = row.GetValue<String>("keyspace_name");
                Console.WriteLine(name);
            }
        }
    }
}
```

使用に関する注意事項:

a. "*path_to_file*/sf-class2-root.crt" を、最初のステップで保存した証明書へのパスに置き換えてください。

b. *ServiceUserName* と *ServicePassword* が、Amazon Keyspaces にプログラムによってアクセスするためのサービス固有の認証情報を作成する の手順に従ってサービス固有の認証情報を生成したときに取得したユーザー名とパスワードと一致していることを確認してください。

c. 利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

Cassandra Go クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス

このセクションでは、Go Cassandra クライアントドライバーを使用して Amazon Keyspaces に接続する方法を説明します。Amazon Keyspaces リソースへのプログラムアクセスに必要な認証情報を、ユーザーとアプリケーションに提供するには、次のいずれかを実行します。

- 特定の AWS Identity and Access Management (IAM) ユーザーに関連付けられたサービス固有の認証情報を作成します。
- セキュリティを強化するために、すべての AWS サービスで使用される IAM プリンシパルの IAM アクセスキーを作成することをお勧めします。Cassandra クライアントドライバー用の Amazon Keyspaces SigV4 認証プラグインを使用すると、ユーザー名とパスワードではなく IAM アクセスキーを使用して Amazon Keyspaces のコールの認証を行うことができます。詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

トピック

- [開始する前に]
- Apache Cassandra 用の Gocql ドライバーとサービス固有の認証情報を使用して Amazon Keyspaces に接続する

- Apache Cassandra 用の Go ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続する

[開始する前に]

開始する前に、次のタスクを完了する必要があります。

Amazon Keyspaces では、クライアントとの安全な接続を確保するために Transport Layer Security (TLS) を使用する必要があります。TLS を使用して Amazon Keyspaces に接続するには、Amazon デジタル証明書をダウンロードし、TLS を使用するように Go ドライバーを設定する必要があります。

次のコマンドを使用して Starfield デジタル証明書をダウンロードし、`sf-class2-root.crt` をローカルまたはホームディレクトリ内に保存します。

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

> **Note**
>
> Amazon デジタル証明書を使用して Amazon Keyspaces に接続することもできます。クライアントが Amazon Keyspaces に正常に接続されている場合は、引き続き Amazon Keyspaces に接続できます。Starfield 証明書は、古い認定権限を使用しているクライアントに対して追加の下位互換性を提供するものです。


```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```


Apache Cassandra 用の Gocql ドライバーとサービス固有の認証情報を使用して Amazon Keyspaces に接続する

1. アプリケーション用の新しいディレクトリを作成します。

```
mkdir ./gocqlexample
```

2. 新しいディレクトリに移動します。

```
cd gocqlexample
```

3. アプリケーション用のファイルを作成します。

```
touch cqlapp.go
```

4. Go ドライバーをダウンロードします。

```
go get github.com/gocql/gocql
```

5. 次のサンプルコードを cqlapp.go ファイルに追加します。

```
package main

import (
     "fmt"
     "github.com/gocql/gocql"
     "log"
)

func main() {

    // add the Amazon Keyspaces service endpoint
    cluster := gocql.NewCluster("cassandra.us-east-2.amazonaws.com")
    cluster.Port=9142
    // add your service specific credentials
    cluster.Authenticator = gocql.PasswordAuthenticator{
            Username: "ServiceUserName",
            Password: "ServicePassword"}
    // provide the path to the sf-class2-root.crt
    cluster.SslOpts = &gocql.SslOptions{
            CaPath: "path_to_file/sf-class2-root.crt",
            EnableHostVerification: false,
     }

     // Override default Consistency to LocalQuorum
     cluster.Consistency = gocql.LocalQuorum
     cluster.DisableInitialHostLookup = false

     session, err := cluster.CreateSession()
     if err != nil {
            fmt.Println("err>", err)
     }
     defer session.Close()
```

```
    // run a sample query from the system keyspace
    var text string
    iter := session.Query("SELECT keyspace_name FROM system_schema.tables;").Iter()
    for iter.Scan(&text) {
        fmt.Println("keyspace_name:", text)
    }
    if err := iter.Close(); err != nil {
        log.Fatal(err)
    }
    session.Close()
}
```

使用に関する注意事項:

a. "*path_to_file*/sf-class2-root.crt" を、最初のステップで保存した証明書へのパスに置き換えてください。

b. *ServiceUserName* と *ServicePassword* が、Amazon Keyspaces にプログラムによってアクセスするためのサービス固有の認証情報を作成する の手順に従ってサービス固有の認証情報を生成したときに取得したユーザー名とパスワードと一致していることを確認してください。

c. 利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

6. プログラムを構築します。

```
go build cqlapp.go
```

7. プログラムを実行します。

```
./cqlapp
```

Apache Cassandra 用の Go ドライバーと SigV4 認証プラグインを使用して Amazon Keyspaces に接続する

次のコードサンプルで、Apache Cassandra 用オープンソース Go ドライバーの SigV4 認証プラグインを使用して、Amazon Keyspaces (Apache Cassandra 向け) にアクセスする方法を示します。

IAM プリンシパル用の認証情報をまだ作成していない場合は、「the section called “ AWS 認証用の IAM 認証情報を作成する”」の手順に従って作成します。Lambda または Amazon EC2 インスタンスで実行されているアプリケーションは、そのインスタンスの認証情報を自動的に使用します。このチュートリアルをローカルで実行するには、認証情報をローカル環境変数として保存します。

Go SigV4 認証プラグインを GitHub リポジトリからアプリケーションに追加します。プラグインは、Cassandra 用のオープンソース Go ドライバーのバージョン 1.2.x をサポートし、 AWS SDK for Go に依存します。

```
$ go mod init
$ go get github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin
```

このコードサンプルでは、Amazon Keyspaces エンドポイントは、Cluster クラスで表されています。クラスターの認証システムプロパティに対して AwsAuthenticator を使用して、認証情報を取得します。

```
package main

import (
        "fmt"
        "github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin/sigv4"
        "github.com/gocql/gocql"
        "log"
)

func main() {
    // configuring the cluster options
    cluster := gocql.NewCluster("cassandra.us-west-2.amazonaws.com")
    cluster.Port=9142

    // the authenticator uses the default credential chain to find AWS credentials
    cluster.Authenticator = sigv4.NewAwsAuthenticator()

    cluster.SslOpts = &gocql.SslOptions{
            CaPath: "path_to_file/sf-class2-root.crt",
            EnableHostVerification: false,
    }
    cluster.Consistency = gocql.LocalQuorum
    cluster.DisableInitialHostLookup = false

    session, err := cluster.CreateSession()
    if err != nil {
     fmt.Println("err>", err)
     return
    }
    defer session.Close()
```

```
    // doing the query
    var text string
    iter := session.Query("SELECT keyspace_name FROM system_schema.tables;").Iter()
    for iter.Scan(&text) {
     fmt.Println("keyspace_name:", text)
    }
    if err := iter.Close(); err != nil {
     log.Fatal(err)
    }
}
```

使用に関する注意事項:

1. "*path_to_file*/sf-class2-root.crt" を、最初のステップで保存した証明書へのパスに置き換えてください。
2. このサンプルをローカルで実行するには、次の変数を環境変数として定義する必要があります。
   - AWS_ACCESS_KEY_ID
   - AWS_SECRET_ACCESS_KEY
   - AWS_DEFAULT_REGION
3. アクセスキーをコード外に保存するには、the section called “ アクセスキーを管理する” のベストプラクティスを参照してください。
4. 利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

Cassandra Perl クライアントドライバーを使用した Amazon Keyspaces へのプログラムアクセス

このセクションでは、Perl クライアントドライバーを使用して Amazon Keyspaces に接続する方法について説明します。このコードサンプルでは、Perl 5 を使用しました。Amazon Keyspaces では、クライアントとの安全な接続を確保するために Transport Layer Security (TLS) を使用する必要があります。

> **Important**
>
> セキュアな接続を構築するために、コードサンプルでは、Starfield デジタル証明書を使用してサーバー認証を行った上で、TLS 接続を確立します。Perl ドライバーではサーバーの Amazon SSL 証明書が検証されないため、Amazon Keyspaces に接続していることを確認で

きません。Amazon Keyspaces への接続時にドライバーにより TLS が使用されるように設定するという 2 番目のステップは引き続き必要で、このステップにより、クライアントとサーバーの間で転送されるデータが確実に暗号化されます。

1. https://metacpan.org/pod/DBD::Cassandra から Cassandra DBI ドライバーをダウンロードして Perl 環境にインストールします。厳密には、環境によって手順が多少異なります。一般的な例を以下に示します。

```
cpanm DBD::Cassandra
```

2. アプリケーション用のファイルを作成します。

```
touch cqlapp.pl
```

3. 次のサンプルコードを cqlapp.pl ファイルに追加します。

```
use DBI;
my $user = "ServiceUserName";
my $password = "ServicePassword";
my $db = DBI->connect("dbi:Cassandra:host=cassandra.us-
east-2.amazonaws.com;port=9142;tls=1;",
$user, $password);

my $rows = $db->selectall_arrayref("select * from system_schema.keyspaces");
print "Found the following Keyspaces...\n";
for my $row (@$rows) {
      print join(" ",@$row['keyspace_name']),"\n";
}

$db->disconnect;
```

Important

ServiceUserName と *ServicePassword* が、Amazon Keyspaces にプログラムによってアクセスするためのサービス固有の認証情報を作成する の手順に従ってサービス固有の認証情報を生成したときに取得したユーザー名とパスワードと一致していることを確認してください。

> **Note**
>
> 利用可能なエンドポイントのリストについては、「the section called “サービスエンドポイント”」を参照してください。

4. アプリケーションを実行します。

```
perl cqlapp.pl
```

VPC エンドポイントを使用して Amazon Keyspaces へのクロスアカウントアクセスを設定する

個別の を作成して使用 AWS アカウント することで、リソースを分離し、開発や本番稼働などのさまざまな環境で使用できます。このトピックでは、 Amazon Virtual Private Cloudのインターフェイス VPC エンドポイントによる Amazon Keyspaces へのクロスアカウントアクセスについて説明します。IAM のクロスアカウントアクセスの設定の詳細については、『IAM ユーザーガイド』の「開発アカウントとプロダクションアカウントを分けるシナリオの例」を参照してください。

Amazon Keyspaces と VPC エンドポイントの詳細については、「the section called “インターフェイス VPC エンドポイントの使用”」を参照してください。

トピック

- 共有 VPC の VPC エンドポイントを使用して Amazon Keyspaces へのクロスアカウントアクセスを設定する
- 共有 VPC を利用せずに Amazon Keyspaces へのクロスアカウントアクセスを設定する

共有 VPC の VPC エンドポイントを使用して Amazon Keyspaces へのクロスアカウントアクセスを設定する

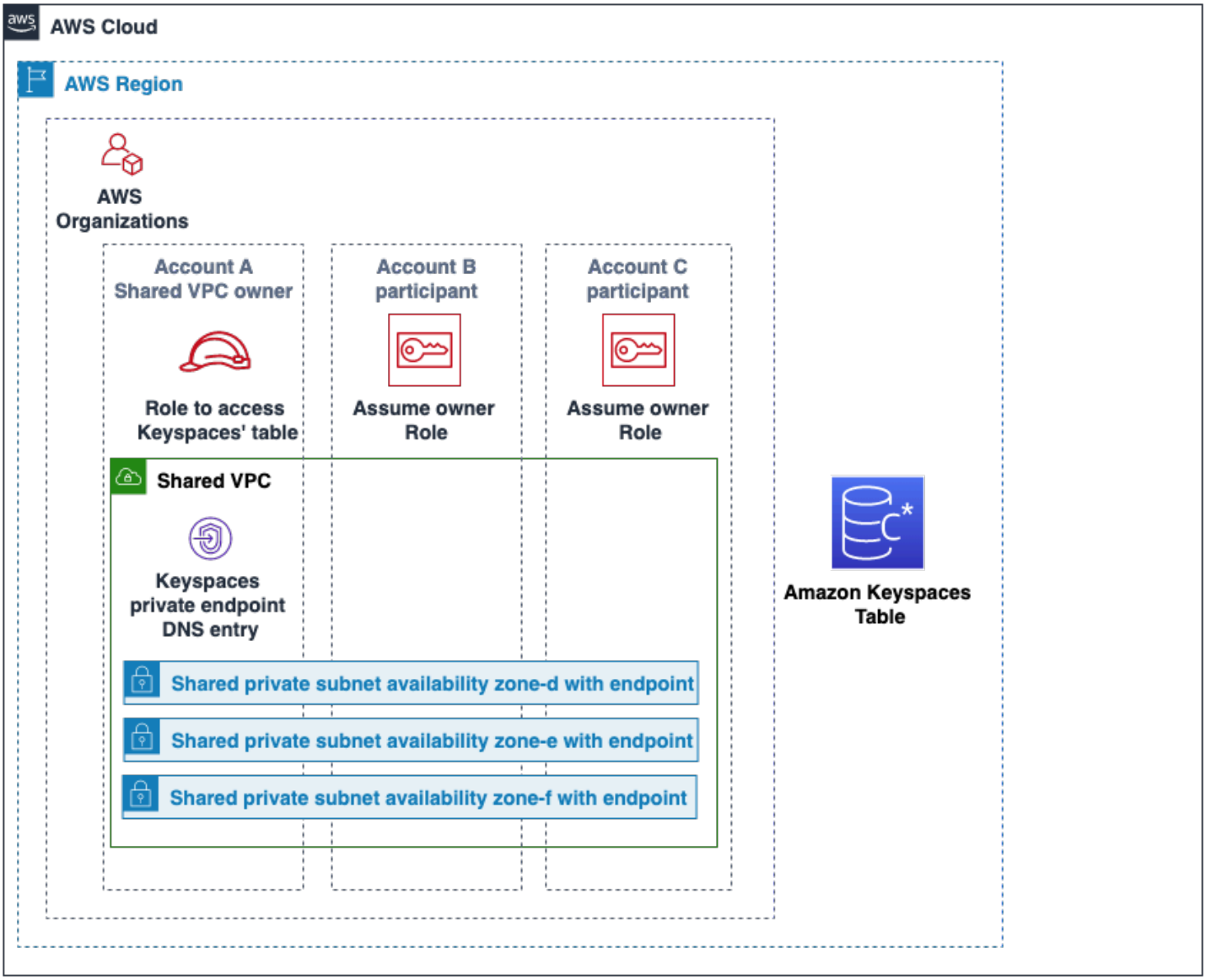
異なる を作成して AWS アカウント 、アプリケーションからリソースを分離できます。たとえば、Amazon Keyspaces テーブル用に 1 つのアカウントを作成し、開発環境のアプリケーション用に別のアカウントを作成し、本番環境のアプリケーション用に別のアカウントを作成するといったことができます。このトピックでは、共有 VPC のインターフェイス VPC エンドポイントを使用した

Amazon Keyspaces のクロスアカウントアクセスを設定するために必要な設定手順を順を追って説明します。

Amazon Keyspaces の VPC エンドポイントを設定する方法の詳細な手順については、「the section called “ステップ 3: Amazon Keyspaces 用の VPC エンドポイントを作成する”」を参照してください。

この例では、共有 VPC で次の 3 つのアカウントを使用します。

- `Account A` — このアカウントには、VPC エンドポイント、VPC サブネット、Amazon Keyspaces テーブルなどのインフラストラクチャが含まれています。
- `Account B` — このアカウントには、`Account A` の Amazon Keyspaces テーブルに接続する必要がある開発環境のアプリケーションが含まれています。
- `Account C` — このアカウントには、`Account A` の Amazon Keyspaces テーブルに接続する必要がある本番環境のアプリケーションが含まれています。



Account A は、 Account Bがアクセスする必要があるリソース (Amazon Keyspaces テーブル) を含むアカウントであり、 Account A は信頼するアカウントです。 Account Bと Account Cは、 のリソース (Amazon Keyspaces テーブル) にアクセスする必要があるプリンシパルを持つアカウントでありAccount A、 Account B と Account Cは信頼されたアカウントです。信頼するアカウントは IAM ロールを共有して、信頼されたアカウントに権限を与えます。次の手順では、Account A に必要な設定ステップを簡単に説明します。

Account A の設定

1. AWS Resource Access Manager を使用してサブネットのリソース共有を作成し、プライベートサブネットを Account Bおよび と共有しますAccount C。

Account B と Account C は、共有されているサブネット内のリソースを表示し、作成できます。

2. を使用した Amazon Keyspaces プライベート VPC エンドポイントを作成します AWS PrivateLink。これにより、Amazon Keyspaces サービスエンドポイントの共有サブネットと DNS エントリに複数のエンドポイントが作成されます。
3. Amazon Keyspaces キースペースとテーブルを作成します。
4. 次のポリシー例に示すように、Amazon Keyspaces テーブルへのAccount Aフルアクセス、Amazon Keyspaces システムテーブルへの読み取りアクセス、Amazon EC2 VPC リソースを記述できる IAM ロールを に作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountAccess",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcEndpoints",
                "cassandra:*"
            ],
            "Resource": "*"
        }
    ]
}
```

5. Account B と がロールを信頼Account Cされたアカウントとして引き受けAccount Aられるように、 で IAM ロールの信頼ポリシーを設定します。以下の例ではこれを示しています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountB:role/Job-role-B",
                "AWS": "arn:aws:iam::AccountC:role/Job-role-C"
            },
            "Action": "sts:AssumeRole",
            "Condition": {}
```

```
        }
    ]
}
```

クロスアカウント IAM ポリシーの詳細については、『IAM ユーザーガイド』の「クロスアカウントポリシー」を参照してください。

Account B と Account C における設定

1. Account B と Account C で新しいロールを作成し、Account A で作成した共有ロールをプリンシパルが引き受けることを許可する以下のポリシーをアタッチします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

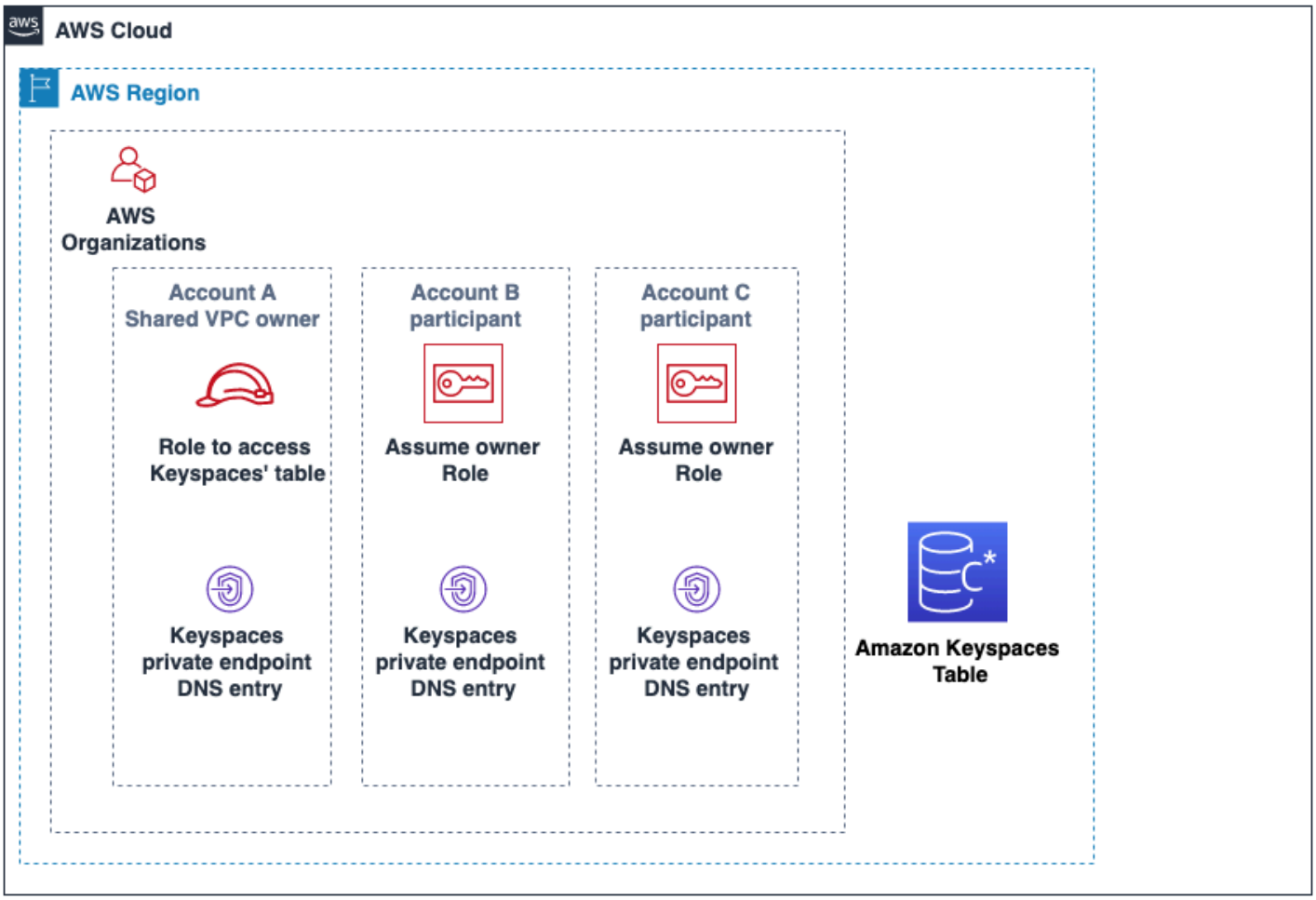
プリンシパルが共有ロールを引き受けることを許可するには、 AWS Security Token Service () の AssumeRole API を使用しますAWS STS。詳細については、「IAM ユーザーガイド」の「自分が所有する別の の IAM ユーザーへのアクセスを提供する AWS アカウント」を参照してください。

2. Account B と Account C では、SIGV4 認証プラグインを利用するアプリケーションを作成できます。これにより、アプリケーションは共有ロールを引き受け、共有 VPC の VPC エンドポイントを介して Account A にある Amazon Keyspaces テーブルに接続できます。SIGV4 認証プラグインの詳細については、「the section called “プログラムによるアクセスのための認証情報の作成”」を参照してください。別の AWS アカウントでロールを引き受けるようにアプリケーションを設定する方法の詳細については、 SDK およびツールリファレンスガイドの「認証とアクセス」を参照してください。 AWS SDKs

共有 VPC を利用せずに Amazon Keyspaces へのクロスアカウントアクセスを設定する

Amazon Keyspaces テーブルとプライベート VPC エンドポイントを別のアカウントが所有していて、VPC を共有していない場合でも、アプリケーションは VPC エンドポイントを使用してクロスアカウントに接続できます。アカウントは VPC エンドポイントを共有していないため、`Account A`、`Account B` 、`Account C` には独自の VPC エンドポイントが必要です。Cassandra クライアントドライバーは、Amazon Keyspaces はマルチノードクラスターではなく単一ノードとして認識します。接続すると、クライアントドライバーは DNS サーバーにアクセスし、DNS サーバーはアカウントの VPC で使用可能なエンドポイントの 1 つを返します。

パブリックエンドポイントを使用するか、各アカウントにプライベート VPC エンドポイントをデプロイすることで、共有 VPC エンドポイントを使用せずに、異なるアカウントの Amazon Keyspaces テーブルにアクセスすることもできます。共有 VPC を使用しない場合、各アカウントには独自の VPC エンドポイントが必要です。この例では、`Account A`、`Account B`、`Account C` が `Account A` のテーブルにアクセスするには独自の VPC エンドポイントが必要です。この設定で VPC エンドポイントを使用する場合、Cassandra クライアントドライバーに、Amazon Keyspaces は、マルチノードクラスターではなく単一ノードクラスターとして認識されます。接続すると、クライアントドライバーは DNS サーバーにアクセスし、DNS サーバーはアカウントの VPC で使用可能なエンドポイントの 1 つを返します。ただし、クライアントドライバーは `system.peers` テーブルにアクセスして追加のエンドポイントを検出することはできません。利用できるホストの数が少ないため、ドライバーによる接続数は少なくなります。これを調整するには、ドライバーの接続プール設定を 3 倍に増やします。



Account A は、 Account Bがアクセスする必要があるリソース (Amazon Keyspaces テーブル) を含むアカウントであり、 Account Aは信頼するアカウントです。 Account Bと Account Cは、 のリソース (Amazon Keyspaces テーブル) へのアクセスを必要とするプリンシパルを持つアカウントでありAccount A、 Account B と Account Cは信頼されたアカウントです。信頼するアカウントは IAM ロールを共有して、信頼されたアカウントに権限を与えます。次の手順では、Account A に必要な設定ステップを簡単に説明します。

Account A の設定

1. で Amazon Keyspaces キースペースとテーブルを作成しますAccount A。
2. Amazon Keyspaces テーブルへのフルアクセスと Amazon Keyspaces システムテーブルへの読み取りアクセス権Account Aを持つ IAM ロールを に作成します。

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

```
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select",
            "cassandra:Modify"
         ],
         "Resource":[
            "arn:aws:cassandra:region:Account-A:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:region:Account-A:/keyspace/system*"
         ]
      }
   ]
}
```

3. と のプリンシパルが信頼Account BAccount Cされたアカウントとしてロールを引き受けAccount Aられるように、 で IAM ロールの信頼ポリシーを設定します。以下の例ではこれを示しています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountB:role/Cross-Account-Role-B",
                "AWS": "arn:aws:iam::AccountC:role/Cross-Account-Role-C"
            },
            "Action": "sts:AssumeRole",
            "Condition": {}
        }
    ]
}
```

クロスアカウント IAM ポリシーの詳細については、『IAM ユーザーガイド』の「クロスアカウントポリシー」を参照してください。

4. で VPC エンドポイントを設定しAccount A、VPC エンドポイントAccount Aを使用してAccount Bと のロールが でロールを引き受けることを許可するアクセス許可をエンドポイントAccount Cにアタッチします。これらのアクセス許可は、アタッチされている VPC エンドポイントに対して有効です。VPC エンドポイントポリシーの詳細については、「the section

called “Amazon Keyspaces のインターフェイス VPC エンドポイントへのアクセスの制御”」を参照してください。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "Allow-access-from-specific-IAM-roles",
         "Effect": "Allow",
         "Principal": "*",
         "Action": "cassandra:*",
         "Resource": "*",
         "Condition": {
            "ArnEquals": {
               "aws:PrincipalArn": "arn:aws:iam::AccountB:role/Cross-Account-Role-
B",
               "aws:PrincipalArn": "arn:aws:iam::AccountC:role/Cross-Account-Role-
C"
            }
         }
      }
   ]
}
```

Account B と Account C における設定

1. Account B と Account C で新しいロールを作成し、Account A で作成した共有ロールをプリンシパルが引き受けることを許可する以下のポリシーをアタッチします。

```
{
    "Version": "2012-10-17",
    "Statement": {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::Account-A:role/keyspaces_access"
        }
}
```

プリンシパルが共有ロールを引き受けることを許可するには、 AWS Security Token Service () の AssumeRole API を使用しますAWS STS。詳細については、「IAM ユーザーガイド」の「自

分が所有する別の の IAM ユーザーへのアクセスを提供する AWS アカウント」を参照してください。

2. Account B および ではAccount C、SIGV4 認証プラグインを使用するアプリケーションを作成できます。これにより、アプリケーションは共有ロールを引き受けて、 にある Amazon Keyspaces テーブルに接続できますAccount A。SIGV4 認証プラグインの詳細については、「the section called “プログラムによるアクセスのための認証情報の作成”」を参照してください。別の AWS アカウントでロールを引き受けるようにアプリケーションを設定する方法の詳細については、 SDK およびツールリファレンスガイドの「認証とアクセス」を参照してください。 AWS SDKs

Amazon Keyspaces (Apache Cassandra 向け) の使用開始

このチュートリアルでは、Apache Cassandra と Amazon Keyspaces を初めて使用する方を対象に、Amazon Keyspaces を正しく使用するために必要なプログラムとツールのインストール手順を案内します。Cassandra クエリ言語 (CQL)、 AWS Management Console、または AWS Command Line Interface () を使用してキースペースとテーブルを作成する方法について説明しますAWS CLI。その後、Cassandra クエリ言語 (CQL) を使用して、Amazon Keyspaces テーブル内のデータに対して作成/読み取り/更新/削除 (CRUD) オペレーションを実行します。

このチュートリアルは、以下のステップで構成されています。

- 前提条件 – チュートリアルを開始する前に、 AWS セットアップ手順に従って にサインアップ AWS し、Amazon Keyspaces にアクセスできる IAM ユーザーを作成します。次に、 `cqhsh-expansion`と をセットアップします AWS CloudShell。または AWS CLI 、 を使用して Amazon Keyspaces にリソースを作成することもできます。
- ステップ 1: キースペースとテーブルを作成する – 「catalog」という名前のキースペースを作成し、その中に「book_awards」という名前のテーブルを作成します。、CQL、または を使用して、テーブルの列、データ型 AWS Management Console、パーティションキー、およびクラスタリング列を指定します AWS CLI。
- ステップ 2: CRUD オペレーションを実行する – CloudShell の `cqlsh-expansion` を使用して、「book_awards」テーブルへのデータの挿入、読み取り、更新、削除を実行します。SELECT、INSERT、UPDATE、DELETE などのさまざまな CQL ステートメントの使用方法を学び、データのフィルタリングと変更の練習をします。
- ステップ 3: リソースをクリーンアップする – 使用していないリソースに対して課金されないように、コンソール、CQL、または AWS CLIを使用して「book_awards」テーブルと「catalog」キースペースを削除する方法を説明します。

さまざまな Apache Cassandra クライアントドライバーを使用して Amazon Keyspaces にプログラムで接続するチュートリアルについては、「the section called “Cassandra クライアントドライバーの使用”」を参照してください。異なる AWS SDKsAWS SDKs」を参照してください。

トピック

- チュートリアルの前提条件と考慮事項
- Amazon Keyspaces でキースペースを作成する
- Amazon Keyspaces でキースペースの作成ステータスを確認する

- Amazon Keyspaces でテーブルを作成する
- Amazon Keyspaces でテーブルの作成ステータスを確認する
- Amazon Keyspaces で CQL を使用してデータを作成/読み取り/更新/削除する (CRUD)
- Amazon Keyspaces でテーブルを削除する
- Amazon Keyspaces でキースペースを削除する

チュートリアルの前提条件と考慮事項

Amazon Keyspaces の使用を開始する前に、 AWS 「」のセットアップ手順に従ってくださいAmazon Keyspaces (Apache Cassandra 向け) へのアクセス。これらのステップには、Amazon Keyspaces へのアクセス権を持つ AWS Identity and Access Management (IAM) ユーザーへの AWS サインアップと作成が含まれます。

チュートリアルのステップをすべて完了するには、cqlsh をインストールする必要があります。「cqlsh を使用した Amazon Keyspaces への接続」の設定手順に従ってください。

cqlsh または を使用して Amazon Keyspaces にアクセスするには AWS CLI、 を使用することをお勧めします AWS CloudShell。CloudShell はブラウザーベースの事前認証済みシェルで、 AWS Management Consoleから直接起動できます。任意のシェル (Bash、PowerShell、または Z シェルAWS CLI) を使用して、Amazon Keyspaces に対して AWS Command Line Interface () コマンドを実行できます。cqlsh を使用するには、cqlsh-expansion をインストールする必要があります。cqlsh-expansion のインストール手順については、「the section called “cqlsh-expansion の使用”」を参照してください。CloudShell の詳細については、「the section called “の使用 AWS CloudShell”」を参照してください。

を使用して Amazon Keyspaces でリソース AWS CLI を作成、表示、削除するには、「」のセットアップ手順に従ってくださいthe section called “AWS CLIのダウンロードと設定”。

前提条件のステップが完了したら、Amazon Keyspaces でキースペースを作成する に進みます。

Amazon Keyspaces でキースペースを作成する

このセクションでは、コンソール、cqlsh、または AWS CLIを使用してキースペースを作成します。

> **Note**
>
> このチュートリアルを開始する前に、次の前提条件をすべて設定してください。

キースペースは、1 つ以上のアプリケーションに関係している関連テーブルをグループ化するものです。キースペースには 1 つ以上のテーブルが含まれており、キースペースに含まれるすべてのテーブルのレプリケーション戦略がキースペースによって定義されます。キースペースの詳細については、次のトピックを参照してください。

- CQL 言語リファレンスのデータ定義言語 (DDL) ステートメント: Keyspaces
- Amazon Keyspaces (Apache Cassandra 向け) のクォータ

このチュートリアルでは、単一リージョンのキースペースを作成します。キースペースのレプリケーション戦略は `SingleRegionStrategy` です。`SingleRegionStrategy` を使用する場合、Amazon Keyspaces は 1 つの AWS リージョン内で 3 つのアベイラビリティーゾーンにわたってデータをレプリケートします。マルチリージョンのキースペースの作成方法については、「the section called “マルチリージョンキースペースの作成”」を参照してください。

コンソールを使用する

コンソールを使用してキースペースを作成するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Keyspaces (キー空間)] を選択します。
3. [Create keyspace (キースペースの作成)] を選択します。
4. [Keyspace name (キースペース名)] ボックスに、キースペースの名前として **`catalog`** を入力します。

   名前の制約:

   - 名前を空にすることはできません。
   - 使用できる文字: 英数字と下線 (_) です。
   - 最大長は 48 文字です。
5. [AWS リージョン] で、[単一リージョンレプリケーション] がキースペースのレプリケーション戦略になっていることを確認します。

6. キースペースを作成するには、[Create keyspace (キースペースの作成)] を選択します。
7. 以下の作業により、キースペース `catalog` が作成されたことを確認します。
   a. ナビゲーションペインで、[Keyspaces (キー空間)] を選択します。
   b. キースペースのリストで該当するキースペース `catalog` を見つけます。

CQL の使用

以下の手順では、CQL を使用してキースペースを作成します。

CQL を使用してキースペースを作成するには

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

このコマンドの出力は次のようになります。

```
Connected to Amazon Keyspaces at cassandra.us-east-1.amazonaws.com:9142
[cqlsh 6.1.0 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh current consistency level is ONE.
```

2. 次の CQL コマンドを使用して、キースペースを作成します。

```
CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

`SingleRegionStrategy` は 3 つのレプリケーション係数を使用し、そのリージョンの 3 つの AWS アベイラビリティーゾーンにデータをレプリケートします。

> **Note**
>
> Amazon Keyspaces ではすべての入力がデフォルトで小文字に設定されます。ただし、引用符で囲まれた入力はこれに該当しません。

3. キースペースが作成されていることを確認します。

```
SELECT * from system_schema.keyspaces;
```

このコマンドの出力は、次のようになります。

```
cqlsh> SELECT * from system_schema.keyspaces;

 keyspace_name           | durable_writes | replication
-------------------------+----------------
+-------------------------------------------------------------------------------------
           system_schema |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}
      system_schema_mcs |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}
                 system |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}
 system_multiregion_info |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}
                catalog |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}

(5 rows)
```

の使用 AWS CLI

以下の手順では、 AWS CLIを使用してキースペースを作成します。

を使用してキースペースを作成するには AWS CLI

1. 環境がセットアップされていることを確認するには、CloudShell で次のコマンドを実行します。

```
aws keyspaces help
```

2. 次の AWS CLI ステートメントを使用してキースペースを作成します。

```
aws keyspaces create-keyspace --keyspace-name 'catalog'
```

3. 次の AWS CLI ステートメントを使用してキースペースが作成されたことを確認します。

```
aws keyspaces get-keyspace --keyspace-name 'catalog'
```

このコマンドの出力は、次の例のようになります。

```
{
    "keyspaceName": "catalog",
    "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/catalog/",
    "replicationStrategy": "SINGLE_REGION"
}
```

Amazon Keyspaces でキースペースの作成ステータスを確認する

Amazon Keyspaces では、キースペースの作成や削除などのデータ定義言語 (DDL) オペレーションを非同期に実行します。

で新しいキースペースの作成ステータスをモニタリングできます。これは AWS Management Console、キースペースが保留中またはアクティブであることを示します。system_schema_mcs キースペースを使用して、新しいキースペースの作成ステータスをプログラムにより監視することもできます。キースペースは、使用できる状態になると、system_schema_mcs keyspaces テーブルに表示されます。

新しいキースペースが使用可能な状態になるタイミングをチェックするための推奨設計パターンとは、Amazon Keyspaces の system_schema_mcs keyspaces テーブル (system_schema_mcs.*) のポーリングです。キースペースの DDL ステートメントのリストについては、「CQL language reference」(CQL 言語リファレンス) の「the section called “Keyspaces”」のセクションを参照してください。

次のクエリは、キースペースが正常に作成されたかどうかを示します。

```
SELECT * FROM system_schema_mcs.keyspaces WHERE keyspace_name = 'mykeyspace';
```

キースペースが正常に作成された場合、クエリの出力は次のようになります。

```
keyspace_name | durable_writes  | replication
--------------+-----------------+--------------
   mykeyspace | true            |{...} 1 item
```

Amazon Keyspaces でテーブルを作成する

このセクションでは、コンソール、cqlsh、または AWS CLIを使用してテーブルを作成します。

テーブルとは、データが整理されて保存されている場所です。テーブルのプライマリキーによって、テーブル内のデータの分割方法が決まります。プライマリキーは、必須のパーティションキー 1 つと、1 つ以上のオプションのクラスタリング列で構成されています。プライマリキーを構成する複合値は、テーブルのすべてのデータで一意でなければなりません。テーブルの詳細については、次のトピックを参照してください。

- パーティションキーの設計: the section called “パーティションキーの設計”
- テーブルの操作: the section called “テーブルの作成ステータスの確認”
- CQL 言語リファレンスの DDL ステートメント: テーブル
- テーブルリソース管理: サーバーレスリソースの管理
- テーブルのリソース使用率のモニタリング: the section called “CloudWatch によるモニタリング”
- Amazon Keyspaces (Apache Cassandra 向け) のクォータ

テーブルを作成する際には、以下を指定できます。

- テーブルの名前。
- テーブル内の各列の名前とデータ型。
- テーブルのプライマリキー。
  - [Partition key (パーティションキー)] - 必須
  - [Clustering columns (クラスタリング列)] — オプション

次の手順で、指定した列、データ型、パーティションキー、クラスタリング列を含むテーブルを作成します。

コンソールを使用する

次の手順では、これらの列とデータ型が含まれているテーブル book_awards を作成します。

```
year           int
award          text
```

```
rank           int
category       text
book_title     text
author         text
publisher      text
```

コンソールを使用してテーブルを作成するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Keyspaces (キー空間)] を選択します。
3. このテーブルを作成するキースペースとして `catalog` を選択します。
4. [Create table (テーブルの作成)] を選択します。
5. [Table name (テーブル名)] ボックスに、テーブルの名前として「**`book_awards`**」を入力します。

   名前の制約:

   - 名前を空にすることはできません。
   - 使用できる文字: 英数字と下線 (_) です。
   - 最大長は 48 文字です。
6. [Columns (列)] セクションで、このテーブルを追加する各列に対して以下のステップを繰り返します。

   次の列とデータ型を追加します。

   ```
   year           int
   award          text
   rank           int
   category       text
   book_title     text
   author         text
   publisher      text
   ```

   a. [Name (名前)] — 列の名前を入力します。

      名前の制約:

      - 名前を空にすることはできません。

- 使用できる文字: 英数字と下線 (_) です。
- 最大長は 48 文字です。

b. [Type (型)] — データ型のリストで、この列のデータ型を選択します。

c. 列を追加するには、[列を追加] を選択します。

7. [パーティションキー] で、パーティションキーとして [award] と [year] を選択します。各テーブルにはパーティションキーが 1 つ必要です。パーティションキーは、1 つ以上の列で構成できます。

8. [クラスタリング列] として category と rank を追加します。クラスタリング列はオプションであり、この列によって各パーティション内のソート順序が決まります。

   a. クラスタリング列を追加するには、[Add clustering column (クラスタリング列の追加)] を選択します。

   b. [列] リストで、[category] を選択します。[Order (順序)] リストで、この列の値を昇順に並べ替えるために [ASC] を選択します。(降順の場合は [DESC] を選択してください。)

   c. 次に、[クラスタリング列を追加] を選択し、[rank] を選択します。

9. [Table settings (テーブル設定)] セクションで、[Default settings (デフォルト設定)] を選択します。

10. [Create table (テーブルの作成)] を選択します。

11. テーブルが作成されていることを確認します。

    a. ナビゲーションペインで、[Tables (テーブル)] を選択します。

    b. そのテーブルがテーブルの一覧に表示されていることを確認します。

    c. テーブルの名前を選択します。

    d. すべての列とデータ型が正しいことを確認します。

> **Note**
>
> これらの列の順序が、テーブルに追加した順序と同じではない場合があります。

CQL の使用

この手順では、CQL を使用して、次の列とデータ型を含むテーブルを作成します。year 列と award 列がパーティションキーであり、category と rank がクラスタリング列です。これらを合わせて、テーブルのプライマリキーを構成しています。

```
year           int
award          text
rank           int
category       text
book_title     text
author         text
publisher      text
```

CQL を使用してテーブルを作成するには

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

   ```
   cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
   ```

   このコマンドの出力は次のようになります。

   ```
   Connected to Amazon Keyspaces at cassandra.us-east-1.amazonaws.com:9142
   [cqlsh 6.1.0 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
   Use HELP for help.
   cqlsh current consistency level is ONE.
   ```

2. キースペースプロンプト (cqlsh:keyspace_name>) で、コマンドウィンドウに次のコードを入力してテーブルを作成します。

   ```
   CREATE TABLE catalog.book_awards (
      year int,
      award text,
      rank int,
      category text,
      book_title text,
      author text,
      publisher text,
      PRIMARY KEY ((year, award), category, rank)
      );
   ```

> **Note**
>
> ASC は、デフォルトのクラスタリング順序です。DESC を指定して降順にすることもできます。

year と award はパーティションキー列です。そして、category と rank はクラスタリング列で、昇順 (ASC) で並べられています。これらの列を合わせて、テーブルのプライマリキーを構成しています。

3. テーブルが作成されていることを確認します。

```
SELECT * from system_schema.tables WHERE keyspace_name='catalog.book_awards' ;
```

出力は次のようになります。

```
 keyspace_name | table_name | bloom_filter_fp_chance | caching | cdc | comment
 | compaction | compression | crc_check_chance | dclocal_read_repair_chance
 | default_time_to_live | extensions | flags | gc_grace_seconds | id |
 max_index_interval | memtable_flush_period_in_ms | min_index_interval |
 read_repair_chance | speculative_retry
---------------+------------+------------------------+---------+-----+---------
+------------+-------------+------------------+---------------------------
+----------------------+------------+-------+------------------+----
+--------------------+-----------------------------+--------------------
+--------------------+------------------

(0 rows)
```

4. テーブルの構造を確認します。

```
SELECT * FROM system_schema.columns WHERE keyspace_name = 'catalog' AND table_name
 = 'book_awards';
```

このステートメントの出力は、次の例のようになります。

```
 keyspace_name | table_name  | column_name | clustering_order | column_name_bytes
    | kind          | position | type
---------------+-------------+-------------+------------------
+------------------------+---------------+----------+------
```

```
      catalog | book_awards |        year |             none |
 0x79656172 | partition_key |        0 |  int
      catalog | book_awards |       award |             none |
 0x6177617264 | partition_key |        1 | text
      catalog | book_awards |    category |              asc |
 0x63617465676f7279 |    clustering |        0 | text
      catalog | book_awards |        rank |              asc |
 0x72616e6b |    clustering |        1 |  int
      catalog | book_awards |      author |             none |
 0x617574686f72 |       regular |       -1 | text
      catalog | book_awards |  book_title |             none |
 0x626f6f6b5f7469746c65 |       regular |       -1 | text
      catalog | book_awards |   publisher |             none |
 0x7075626c6973686572 |       regular |       -1 | text

(7 rows)
```

すべての列とデータ型が想定どおりであるか確認します。列の順序は CREATE ステートメントの順序と異なる場合があります。

の使用 AWS CLI

この手順では、 AWS CLIを使用して、次の列とデータ型を含むテーブルを作成します。year 列と award 列がパーティションキーであり、category と rank がクラスタリング列です。

```
year           int
award          text
rank           int
category       text
book_title     text
author         text
publisher      text
```

を使用してテーブルを作成するには AWS CLI

以下のコマンドで book_wards という名前のテーブルが作成されます。テーブルのパーティションキーは year 列と award 列で構成され、クラスタリングキーは category 列と rank 列で構成されていて、クラスタリング列はどちらも昇順で並べられています (読みやすさに配慮して、このセクションでは、テーブル作成コマンドの schema-definition を複数の行に分けて表示しています)。

1. 次のステートメントを使用して、テーブルを作成できます。

```
aws keyspaces create-table --keyspace-name 'catalog' \
                      --table-name 'book_awards' \
                      --schema-definition 'allColumns=[{name=year,type=int},
{name=award,type=text},{name=rank,type=int},
            {name=category,type=text}, {name=author,type=text},
{name=book_title,type=text},{name=publisher,type=text}],
            partitionKeys=[{name=year},
{name=award}],clusteringKeys=[{name=category,orderBy=ASC},{name=rank,orderBy=ASC}]'
```

   このコマンドで、次の出力が返ります。

```
{
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/catalog/
table/book_awards"
}
```

2. テーブルのメタデータとプロパティは、以下のコマンドで確認します。

```
aws keyspaces get-table --keyspace-name 'catalog' --table-name 'book_awards'
```

   このコマンドで、以下の出力が返ります。

```
{
    "keyspaceName": "catalog",
    "tableName": "book_awards",
    "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/catalog/
table/book_awards",
    "creationTimestamp": "2024-07-11T15:12:55.571000+00:00",
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "year",
                "type": "int"
            },
            {
                "name": "award",
                "type": "text"
            },
            {
```

```
                "name": "category",
                "type": "text"
            },
            {
                "name": "rank",
                "type": "int"
            },
            {
                "name": "author",
                "type": "text"
            },
            {
                "name": "book_title",
                "type": "text"
            },
            {
                "name": "publisher",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "year"
            },
            {
                "name": "award"
            }
        ],
        "clusteringKeys": [
            {
                "name": "category",
                "orderBy": "ASC"
            },
            {
                "name": "rank",
                "orderBy": "ASC"
            }
        ],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2024-07-11T15:12:55.571000+00:00"
    },
```

```
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    },
    "replicaSpecifications": []
}
```

テーブル内のデータに対して CRUD (create (作成)、read (読み取り)、update (更新)、delete (削除)) オペレーションを実行するには、the section called “CRUD オペレーション” に進みます。

Amazon Keyspaces でテーブルの作成ステータスを確認する

Amazon Keyspaces では、テーブルの非同期的な作成や削除など、データ定義言語 (DDL) オペレーションを同期なしで実行します。で新しいテーブルの作成ステータスをモニタリングできます。これは AWS Management Console、テーブルが保留中またはアクティブであることを示します。システムスキーマテーブルを使用して、新しいテーブルの作成ステータスをプログラムにより監視することもできます。

テーブルは、使用可能な状態になると、システムスキーマでアクティブとして表示されます。新しいテーブルが使用可能な状態になるタイミングをチェックするための推奨設計パターンとは、Amazon Keyspaces のシステムスキーマテーブル (`system_schema_mcs.*`) のポーリングです。テーブルの DDL ステートメントのリストについては、「CQL language reference」(CQL 言語リファレンス) の「the section called “テーブル”」セクションを参照してください。

次のクエリはテーブルのステータスを示しています。

```
SELECT keyspace_name, table_name, status FROM system_schema_mcs.tables WHERE
 keyspace_name = 'mykeyspace' AND table_name = 'mytable';
```

まだ作成中で保留されているテーブルの場合、クエリの出力は次のようになります。

```
keyspace_name | table_name | status
--------------+------------+--------
```

```
   mykeyspace |    mytable | CREATING
```

テーブルが正常に作成されてアクティブになると、クエリの出力は次のようになります。

```
keyspace_name | table_name | status
--------------+------------+--------
   mykeyspace |    mytable | ACTIVE
```

Amazon Keyspaces で CQL を使用してデータを作成/読み取り/更新/削除する (CRUD)

チュートリアルのこのステップでは、CQL データ操作言語 (DML) ステートメントを使用して、Amazon Keyspaces テーブルのデータを挿入/読み取り/更新/削除する方法について説明します。Amazon Keyspaces では、CQL 言語でのみ DML ステートメントを作成できます。このチュートリアルでは、 AWS Management Consoleの AWS CloudShell で `cqlsh-expansion` を使用して、DML ステートメントの実行を練習します。

- データの挿入 – `INSERT` ステートメントを使用して、1 つのテーブルに単一および複数のレコードを挿入する方法を説明します。CSV ファイルからデータをアップロードし、正常に挿入されたことを `SELECT` クエリで確認する方法を学びます。
- データの読み取り – `SELECT` ステートメントのさまざまなバリエーションを試し、テーブルからデータを取得します。取り扱うトピックは、すべてのデータの選択、特定の列の選択、`WHERE` 句を使用した条件に基づく行のフィルタリング、単純条件と複合条件の理解などです。
- データの更新 – `UPDATE` ステートメントを使用して、テーブル内の既存のデータを変更する方法を学びます。単一列と複数列を実際に更新してみて、プライマリキー列の更新にまつわる制約について理解しましょう。
- データの削除 – 最終セクションでは、`DELETE` ステートメントを使用して、テーブルからデータを削除する方法を説明します。特定のセルや行全体の削除方法を学び、データを削除した場合とテーブル全体またはキースペース全体を削除した場合の違いを理解しましょう。

このチュートリアル全体を通じて、例やヒントを紹介します。また、さまざまなシナリオに合わせて自分で CQL クエリを書く練習をすることもできます。

トピック

- Amazon Keyspaces テーブルにデータを挿入してロードする

- Amazon Keyspaces で CQL SELECT ステートメントを使用してテーブルからデータを読み取る
- CQL を使用して Amazon Keyspaces テーブルのデータを更新する
- CQL DELETE ステートメントを使用してテーブルからデータを削除する

Amazon Keyspaces テーブルにデータを挿入してロードする

book_awards テーブルでデータを作成するには、INSERT ステートメントを使用して 1 行を追加します。

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

   ```
   cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
   ```

   このコマンドの出力は次のようになります。

   ```
   Connected to Amazon Keyspaces at cassandra.us-east-1.amazonaws.com:9142
   [cqlsh 6.1.0 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
   Use HELP for help.
   cqlsh current consistency level is ONE.
   ```

2. cqlsh を使用してAmazon Keyspaces テーブルにデータを書き込む前に、現在の cqlsh セッションの書き込み整合性を LOCAL_QUORUM に設定する必要があります。サポートされる整合性レベルの詳細については、「the section called “書き込みの整合性レベル”」を参照してください。AWS Management Consoleで CQL エディタを使用している場合は、このステップは不要であることに注意してください。

   ```
   CONSISTENCY LOCAL_QUORUM;
   ```

3. 単一のレコードを挿入するには、CQL エディタで次のコマンドを実行します。

   ```
   INSERT INTO catalog.book_awards (award, year, category, rank, author, book_title,
    publisher)
   VALUES ('Wolf', 2023, 'Fiction',3,'Shirley Rodriguez','Mountain', 'AnyPublisher') ;
   ```

4. 次のコマンドを実行して、データがテーブルに正しく追加されていることを確認します。

   ```
   SELECT * FROM catalog.book_awards;
   ```

ステートメントの出力は次のようになります。

```
 year | award | category | rank | author            | book_title | publisher
------+-------+----------+------+-------------------+------------+-------------
 2023 |  Wolf |  Fiction |    3 | Shirley Rodriguez |   Mountain | AnyPublisher

(1 rows)
```

cqlsh を使用してファイルから複数のレコードを挿入するには

1. サンプル CSV ファイル (keyspaces_sample_table.csv) を収録したアーカイブファイル samplemigration.zip をダウンロードします。アーカイブを解凍し、keyspaces_sample_table.csv へのパスをメモしておきます。

award	year	category	rank	author	book_title	publisher
Kwesi Manu Prize	2020	Fiction	1	Akua Mansa	Where did you go?	SomePublisher
Kwesi Manu Prize	2020	Fiction	2	John Stiles	Yesterday	Example Books
Kwesi Manu Prize	2020	Fiction	3	Nikki Wolf	Moving to the Chateau	AnyPublisher
Wolf	2020	Non-Fiction	1	Wang Xiulan	History of Ideas	Example Books
Wolf	2020	Non-Fiction	2	Ana Carolina Silva	Science Today	SomePublisher
Wolf	2020	Non-Fiction	3	Shirley Rodriguez	The Future of Sea Ice	AnyPublisher
Richard Roe	2020	Fiction	1	Alejandro Rosalez	Long Summer	SomePublisher
Richard Roe	2020	Fiction	2	Arnav Desai	The Key	Example Books
Richard Roe	2020	Fiction	3	Mateo Jackson	Inside the Whale	AnyPublisher

2. AWS CloudShell で AWS Management Console を開き、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

3. cqlsh プロンプト (cqlsh>) で、キースペースを指定します。

```
USE catalog ;
```

4. 書き込み整合性を LOCAL_QUORUM に設定します。サポートされる整合性レベルの詳細については、「the section called “書き込みの整合性レベル”」を参照してください。

```
CONSISTENCY LOCAL_QUORUM;
```

5. 画面の右上にある「アクション AWS CloudShell 」を選択し、「ファイルのアップロード」を選択して、以前にダウンロードした csv ファイルをアップロードします。ファイルのパスを書き留めておいてください。

6. キースペースのプロンプト (cqlsh:catalog>) で、次のステートメントを実行します。

```
COPY book_awards (award, year, category, rank, author, book_title, publisher) FROM
 '/home/cloudshell-user/keyspaces_sample_table.csv' WITH header=TRUE ;
```

ステートメントの出力は次のようになります。

```
cqlsh:catalog> COPY book_awards (award, year, category, rank, author,
 book_title, publisher)                      FROM '/home/cloudshell-user/
keyspaces_sample_table.csv' WITH delimiter=',' AND header=TRUE ;
cqlsh current consistency level is LOCAL_QUORUM.
Reading options from /home/cloudshell-user/.cassandra/cqlshrc:[copy]:
 {'numprocesses': '16', 'maxattempts': '1000'}
Reading options from /home/cloudshell-user/.cassandra/cqlshrc:[copy-from]:
 {'ingestrate': '1500', 'maxparseerrors': '1000', 'maxinserterrors': '-1',
 'maxbatchsize': '10', 'minbatchsize': '1', 'chunksize': '30'}
Reading options from the command line: {'delimiter': ',', 'header': 'TRUE'}
Using 16 child processes

Starting copy of catalog.book_awards with columns [award, year, category, rank,
 author, book_title, publisher].
OSError: handle is closed      0 rows/s; Avg. rate:       0 rows/s
Processed: 9 rows; Rate:       0 rows/s; Avg. rate:       0 rows/s
9 rows imported from 1 files in 0 day, 0 hour, 0 minute, and 26.706 seconds (0
 skipped).
```

7. 次のクエリを実行して、データがテーブルに正しく追加されていることを確認します。

```
SELECT * FROM book_awards ;
```

次のような出力が表示されます。

```
 year | award            | category    | rank | author             | book_title
        | publisher
------+------------------+-------------+------+--------------------
+----------------------+---------------
 2020 |             Wolf | Non-Fiction |    1 |        Wang Xiulan |      History
 of Ideas | Example Books
 2020 |             Wolf | Non-Fiction |    2 | Ana Carolina Silva |
 Science Today | SomePublisher
 2020 |             Wolf | Non-Fiction |    3 |  Shirley Rodriguez | The Future of
 Sea Ice |  AnyPublisher
```

```
 2020 | Kwesi Manu Prize |     Fiction |    1 |         Akua Mansa |     Where did
 you go? | SomePublisher
 2020 | Kwesi Manu Prize |     Fiction |    2 |        John Stiles |
 Yesterday | Example Books
 2020 | Kwesi Manu Prize |     Fiction |    3 |         Nikki Wolf | Moving to the
 Chateau |  AnyPublisher
 2020 |      Richard Roe |     Fiction |    1 |  Alejandro Rosalez |           Long
 Summer | SomePublisher
 2020 |      Richard Roe |     Fiction |    2 |        Arnav Desai |
 The Key | Example Books
 2020 |      Richard Roe |     Fiction |    3 |      Mateo Jackson |      Inside
 the Whale |  AnyPublisher

(9 rows)
```

cqlsh COPY を使用して csv ファイルから Amazon Keyspaces テーブルにデータをアップロードする方法の詳細については、「the section called “cqlsh を使用したデータのロード”」を参照してください。

Amazon Keyspaces で CQL **SELECT** ステートメントを使用してテーブルからデータを読み取る

「Amazon Keyspaces テーブルにデータを挿入してロードする」セクションでは、SELECT ステートメントを使用してテーブルにデータを正しく追加できたことを確認しました。このセクションでは、SELECT を使用して特定の列を表示し、特定の条件を満たす行のみを表示する方法について詳しく説明します。

SELECT ステートメントの一般的な形式は次のとおりです。

```
SELECT column_list FROM table_name [WHERE condition [ALLOW FILTERING]] ;
```

トピック

- テーブル内のすべてのデータを選択する
- 列のサブセットを選択する
- 行のサブセットを選択する

テーブル内のすべてのデータを選択する

最も単純な形式の SELECT ステートメントにより、テーブル内のすべてのデータが返されます。

> **Important**
>
> このコマンドを実行するとテーブル内のすべてのデータが返されるため、通常、本番環境でこのコマンドを実行することはベストプラクティスではありません。

テーブルのすべてのデータを選択するには

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

2. 次のクエリを実行します。

```
SELECT * FROM catalog.book_awards ;
```

column_list にワイルドカード文字 (*) を使用するとすべての列が選択されます。ステートメントの出力は、次の例のようになります。

```
 year | award            | category    | rank | author             | book_title
        | publisher
------+------------------+-------------+------+--------------------
+-----------------------+---------------
 2020 |             Wolf | Non-Fiction |    1 |        Wang Xiulan |      History
 of Ideas |  AnyPublisher
 2020 |             Wolf | Non-Fiction |    2 | Ana Carolina Silva |
 Science Today | SomePublisher
 2020 |             Wolf | Non-Fiction |    3 |  Shirley Rodriguez | The Future of
 Sea Ice |  AnyPublisher
 2020 | Kwesi Manu Prize |     Fiction |    1 |         Akua Mansa |     Where did
 you go? | SomePublisher
 2020 | Kwesi Manu Prize |     Fiction |    2 |        John Stiles |
 Yesterday | Example Books
 2020 | Kwesi Manu Prize |     Fiction |    3 |         Nikki Wolf | Moving to the
 Chateau |  AnyPublisher
```

```
 2020 |      Richard Roe |     Fiction |    1 |  Alejandro Rosalez |           Long
Summer | SomePublisher
2020 |      Richard Roe |     Fiction |    2 |        Arnav Desai |
The Key | Example Books
2020 |      Richard Roe |     Fiction |    3 |      Mateo Jackson |      Inside
the Whale |  AnyPublisher
```

列のサブセットを選択する

列のサブセットをクエリするには

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

2. award 列、category 列、year 列のみを検索するには、次のクエリを実行します。

```
SELECT award, category, year FROM catalog.book_awards ;
```

出力には、指定した列だけが、SELECT ステートメントに指定された順序で表示されます。

```
 award            | category    | year
------------------+-------------+------
             Wolf | Non-Fiction | 2020
             Wolf | Non-Fiction | 2020
             Wolf | Non-Fiction | 2020
 Kwesi Manu Prize |     Fiction | 2020
 Kwesi Manu Prize |     Fiction | 2020
 Kwesi Manu Prize |     Fiction | 2020
      Richard Roe |     Fiction | 2020
      Richard Roe |     Fiction | 2020
      Richard Roe |     Fiction | 2020
```

行のサブセットを選択する

大きなデータセットをクエリする場合、特定の条件を満たすレコードのみが必要になる可能性があります。これを行うには、WHERE 句を SELECT ステートメントの最後に追加します。

行のサブセットをクエリするには

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

2. 特定の年 (year) の賞 (award) に関するレコードのみを取得するには、次のクエリを実行します。

```
SELECT * FROM catalog.book_awards WHERE year=2020 AND award='Wolf' ;
```

上記の SELECT ステートメントは、次の出力を返します。

```
 year | award | category    | rank | author             | book_title            |
 publisher
------+-------+-------------+------+--------------------+-----------------------
+---------------
 2020 |  Wolf | Non-Fiction |    1 |        Wang Xiulan |      History of Ideas |
 AnyPublisher
 2020 |  Wolf | Non-Fiction |    2 | Ana Carolina Silva |         Science Today |
 SomePublisher
 2020 |  Wolf | Non-Fiction |    3 |  Shirley Rodriguez | The Future of Sea Ice |
 AnyPublisher
```

WHERE 句について

WHERE 句は、データをフィルタリングして指定の基準を満たすデータのみを返すために使用されます。指定の基準は単純条件または複合条件になります。

WHERE 句で条件を使用する方法

- 単純条件 — 単一の列。

```
WHERE column_name=value
```

次の条件のいずれかを満たしていれば WHERE 句の単純条件を使用できます。

- この列は、テーブルの唯一のパーティションキー列です。
- WHERE 句の条件の後に ALLOW FILTERING を追加します。

ALLOW FILTERING を使用すると、特に、複数に分割された大規模なテーブルでは、パフォーマンスの一貫性が失われる可能性がありますので注意が必要です。

- 複合条件 — 複数の単純条件を AND によって結合したもの。

```
WHERE column_name1=value1 AND column_name2=value2 AND column_name3=value3...
```

次の条件のいずれかを満たしていれば WHERE 句の複合条件を使用できます。

- WHERE 句では、テーブルのパーティションキーの列を使用できますが、それらの列を全部含めるか、一部だけを含める必要があります。WHERE 句で一部の列だけを使用する場合は、パーティションキーの先頭列を筆頭に、左から右に連続するパーティションキー列を含めなければなりません。例えば、パーティションキー列が year、month、award である場合、次の列を WHERE 句で使用できます。
  - year
  - year と month の両方
  - year、month、award のすべて
- 次の例で示されているように、WHERE 句の複合条件の後に ALLOW FILTERING を追加します。

```
SELECT * FROM my_table WHERE col1=5 AND col2='Bob' ALLOW FILTERING ;
```

ALLOW FILTERING を使用すると、特に、複数に分割された大規模なテーブルでは、パフォーマンスの一貫性が失われる可能性がありますので注意が必要です。

試してみましょう

独自の CQL クエリを作成して、book_awards テーブルで以下の検索を実行してみましょう。

- 2020 年の Wolf 賞 (award) の受賞作品を検索し、本のタイトル (book title) と著者 (author) をランク (rank) 順に表示します。
- 2020 年のすべての賞の 1 位を表示し、その本のタイトルと賞の名前を表示します。

CQL を使用して Amazon Keyspaces テーブルのデータを更新する

book_awards テーブルのデータを更新するには、UPDATE ステートメントを使用します。

UPDATE ステートメントの一般的な形式は次のとおりです。

```
UPDATE table_name SET column_name=new_value WHERE primary_key=value ;
```

Tip

- 次の例に示すとおり、column_names と値のカンマ区切りリストを使用して、複数の列を更新することができます。

  ```
  UPDATE my_table SET col1='new_value_1', col2='new_value2' WHERE col3='1' ;
  ```

- プライマリキーが複数の列で構成されている場合、すべてのプライマリキー列とその値が WHERE 句に含まれていなければなりません。
- プライマリキーの列を更新すると、レコードのプライマリキーが変更されるため、更新はできません。

単一のセルを更新するには

book_awards テーブルを使用して、2020 年のノンフィクション Wolf 賞受賞作品の出版社 (publisher) の名前を変更します。

```
UPDATE book_awards SET publisher='new Books' WHERE year = 2020 AND award='Wolf' AND
 category='Non-Fiction' AND rank=1;
```

出版社が new Books になったことを確認します。

```
SELECT * FROM book_awards WHERE year = 2020 AND award='Wolf' AND category='Non-Fiction'
 AND rank=1;
```

このステートメントは、次の出力を返します。

```
 year | award | category    | rank | author      | book_title       | publisher
------+-------+-------------+------+-------------+------------------+-----------
 2020 |  Wolf | Non-Fiction |    1 | Wang Xiulan | History of Ideas | new Books
```

試してみましょう

上級: 2020 年のフィクション「Kwezi Manu Prize」賞の受賞作品の名前が変更されました。該当するレコードを更新し、名前を 'Akua Mansa-House' に変更してください。

CQL **DELETE** ステートメントを使用してテーブルからデータを削除する

book_awards テーブルのデータを削除するには、DELETE ステートメントを使用します。

行またはパーティションからデータを削除できます。削除は取り消せないため、データを削除するときは注意してください。

テーブルから 1 つまたはすべての行を削除しても、テーブルは削除されません。したがって、データの再入力が可能です。テーブルを削除すると、テーブルとその中のすべてのデータが削除されます。テーブルを再使用するには、テーブルを再作成してデータを追加する必要があります。キースペースを削除すると、キースペースとその中のすべてのテーブルが削除されます。削除したキースペースとテーブルを使用するには、それらを再作成してデータを入力する必要があります。Amazon Keyspaces のポイントインタイムリカバリ (PITR) を使用して、削除されたテーブルを復元できます。詳細については、「the section called “ポイントインタイムリカバリによるバックアップと復元”」を参照してください。PITR が有効になっている削除済みテーブルを復元する方法については、「the section called “削除済みテーブルの復元”」を参照してください。

セルを削除する

行から列を削除すると、指定したセルからデータが削除されます。SELECT ステートメントを使用してその列を表示すると、そのデータは *null* として表示されます。ただし、null 値はその場所には保存されません。

1 つ以上の特定の列を削除するための一般的な構文は次のとおりです。

```
DELETE column_name1[, column_name2...] FROM table_name WHERE condition ;
```

book_awards テーブルでは、2020 年の「Richard Roe」賞の第 1 位を受賞した本のタイトルが「Long Summer」であることがわかります。この作品の受賞が取り消されたため、このセルから該当データを削除する必要があるとしましょう。

特定のセルを削除するには

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

   ```
   cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
   ```

2. 次の DELETE クエリを実行します。

```
DELETE book_title FROM catalog.book_awards WHERE year=2020 AND award='Richard Roe'
 AND category='Fiction' AND rank=1;
```

3. 削除リクエストが想定どおりに行われたことを確認します。

```
SELECT * FROM catalog.book_awards WHERE year=2020 AND award='Richard Roe' AND
 category='Fiction' AND rank=1;
```

このステートメントの出力は次のようになります。

```
 year | award       | category | rank | author            | book_title | publisher
------+-------------+----------+------+-------------------+------------
+---------------
 2020 | Richard Roe |  Fiction |    1 | Alejandro Rosalez |       null |
 SomePublisher
```

行を削除する

データ削除のリクエストに応えるといった理由で、行全体を削除しなければならない場合も考えられます。行の削除に使用する一般的な構文は次のとおりです。

```
DELETE FROM table_name WHERE condition ;
```

行を削除するには

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

2. 次の DELETE クエリを実行します。

```
DELETE FROM catalog.book_awards WHERE year=2020 AND award='Richard Roe' AND
 category='Fiction' AND rank=1;
```

3. 削除が想定どおりに行われたことを確認します。

```
SELECT * FROM catalog.book_awards WHERE year=2020 AND award='Richard Roe' AND
 category='Fiction' AND rank=1;
```

行が削除された後は、このステートメントの出力が次のようになります。

```
 year | award | category | rank | author | book_title | publisher
------+-------+----------+------+--------+------------+-----------

(0 rows)
```

Amazon Keyspaces の Time to Live を使用して、期限切れのデータをテーブルから自動的に削除できます。詳細については、「the section called “TTL によるデータの期限切れ”」を参照してください。

Amazon Keyspaces でテーブルを削除する

不要になったテーブルやデータに対して課金されないように、使用していないテーブルはすべて削除してください。テーブルを削除すると、テーブルとそのデータが削除され、それらの料金が発生しなくなります。ただし、キースペースは残ります。キー空間を削除すると、キー空間とそのテーブルが削除され、それらの料金が発生しなくなります。

コンソール、CQL、または AWS CLIを使用して、テーブルを削除することができます。テーブルを削除すると、テーブルとそのデータがすべて削除されます。

コンソールを使用する

以下の手順では、 AWS Management Consoleを使用してテーブルとそのデータをすべて削除します。

コンソールを使用してテーブルを削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Tables (テーブル)] を選択します。
3. 削除する各テーブルの名前の左にあるチェックボックスを選択します。
4. [Delete (削除)] を選択します。

5. [Delete table (テーブルの削除)] 画面で、ボックスに **Delete** を入力します。[Delete table (テーブルの削除)] を選択します。
6. テーブルが削除されたことを確認するには、ナビゲーションペインで [Tables (テーブル)] を選択して、book_awards テーブルが表示されなくなったことを確認します。

CQL の使用

以下の手順では、CQL を使用してテーブルとそのデータをすべて削除します。

CQL を使用してテーブルを削除するには

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

2. 次のステートメントを入力して、テーブルを削除します。

```
DROP TABLE IF EXISTS catalog.book_awards ;
```

3. テーブルが削除されたことを確認します。

```
SELECT * FROM system_schema.tables WHERE keyspace_name = 'catalog' ;
```

出力は次のようになります。多少時間がかかる場合があるため、この結果が表示されない場合は、1 分後に再びステートメントを実行してください。

```
keyspace_name | table_name | bloom_filter_fp_chance | caching | cdc | comment
 | compaction | compression | crc_check_chance | dclocal_read_repair_chance
 | default_time_to_live | extensions | flags | gc_grace_seconds | id |
 max_index_interval | memtable_flush_period_in_ms | min_index_interval |
 read_repair_chance | speculative_retry
---------------+------------+-------------------------+---------+-----+---------
+------------+-------------+------------------+---------------------------
+----------------------+------------+-------+------------------+----
+--------------------+-----------------------------+-------------------
+--------------------+------------------

(0 rows)
```

の使用 AWS CLI

以下の手順では、 AWS CLIを使用してテーブルとそのデータをすべて削除します。

を使用してテーブルを削除するには AWS CLI

1. CloudShell を開きます。
2. 次のステートメントを使用して、テーブルを削除します。

```
aws keyspaces delete-table --keyspace-name 'catalog' --table-name 'book_awards'
```

3. テーブルが削除されたことを確認するには、キースペース内のすべてのテーブルを一覧表示します。

```
aws keyspaces list-tables --keyspace-name 'catalog'
```

次のような出力が表示されます。この非同期オペレーションには時間がかかる場合があります。しばらくしてからコマンドを再び実行して、テーブルが削除されたことを確認してください。

```
{
    "tables": []
}
```

Amazon Keyspaces でキースペースを削除する

キースペースに対して課金されないように、使用していないキースペースはすべて削除してください。キー空間を削除すると、キー空間とそのテーブルが削除され、それらの料金が発生しなくなります。

コンソール、CQL、または AWS CLIを使用して、キースペースを削除することができます。

コンソールを使用する

以下の手順では、コンソールを使用してキースペースと、そのテーブルとデータをすべて削除します。

コンソールを使用してキースペースを削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Keyspaces (キー空間)] を選択します。
3. 削除する各キースペースの名前の左にあるチェックボックスを選択します。
4. [Delete (削除)] をクリックします。
5. [Delete keyspace (キースペースの削除)] 画面で、ボックスに **Delete** を入力します。次に、[Delete keyspace (キースペースの削除)] を選択します。
6. キースペース `catalog` が削除されたことを確認するには、ナビゲーションペインで [Keyspaces (キースペース)] を選択して、そのキースペースが表示されなくなったことを確認します。そのキースペースを削除したため、[Tables (テーブル)] に `book_awards` テーブルも表示されなくなります。

CQL の使用

以下の手順では、CQL を使用してキースペースとそのテーブルとデータをすべて削除します。

CQL を使用してキースペースを削除するには

1. を開き AWS CloudShell 、次のコマンドを使用して Amazon Keyspaces に接続します。*us-east-1* は実際のリージョンに置き換えてください。

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

2. 次のステートメントを入力して、キースペースを削除します。

```
DROP KEYSPACE IF EXISTS catalog ;
```

3. そのキー空間が削除されたことを確認します。

```
SELECT * from system_schema.keyspaces ;
```

そのキースペースは表示されていないはずです。これは非同期オペレーションであるため、キースペースが削除されるまで、しばらく時間がかかる場合があります。キースペースが削除された後は、ステートメントの出力が次のようになります。

```
keyspace_name           | durable_writes | replication
```

```
-------------------------+----------------
+-------------------------------------------------------------------------------------------
           system_schema |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}
       system_schema_mcs |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}
                  system |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}
 system_multiregion_info |           True | {'class':
 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}

(4 rows)
```

の使用 AWS CLI

以下の手順では、 AWS CLIを使用してキースペースとそのテーブルとデータをすべて削除します。

を使用してキースペースを削除するには AWS CLI

1. AWS CloudShell を開きます。
2. 次のステートメントを入力して、キースペースを削除します。

```
aws keyspaces delete-keyspace --keyspace-name 'catalog'
```

3. そのキー空間が削除されたことを確認します。

```
aws keyspaces list-keyspaces
```

出力は次のようになります。これは非同期オペレーションであるため、キースペースが削除されるまで、しばらく時間がかかる場合があります。

```
{
    "keyspaces": [
        {
            "keyspaceName": "system_schema",
            "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/
system_schema/",
            "replicationStrategy": "SINGLE_REGION"
        },
        {
            "keyspaceName": "system_schema_mcs",
```

```
            "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/
system_schema_mcs/",
            "replicationStrategy": "SINGLE_REGION"
        },
        {
            "keyspaceName": "system",
            "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/
system/",
            "replicationStrategy": "SINGLE_REGION"
        },
        {
            "keyspaceName": "system_multiregion_info",
            "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/
system_multiregion_info/",
            "replicationStrategy": "SINGLE_REGION"
        }
    ]
}
```

Amazon Keyspaces のチュートリアルとソリューション

このトピックでは、Amazon Keyspaces にプログラムで接続したり、Amazon Keyspaces を他の AWS サービスで使用してソリューションを作成したりするために必要な詳細な手順について説明します。利用可能なチュートリアルでは、さまざまな一般的なクロスサービスアクセスシナリオについて説明し、Amazon Keyspaces を Amazon Virtual Private Cloud、Amazon Elastic Kubernetes Service、Amazon Simple Storage Service、および と統合する方法 AWS Glue、または Apache Spark などのオープンソーステクノロジーと統合する方法を示します。

Apache Cassandra ドライバーを使用して Amazon Keyspaces に接続する方法を示すstep-by-stepのチュートリアルについては、「」を参照してくださいthe section called “Cassandra クライアントドライバーの使用”。

トピック

- チュートリアル: インターフェイス VPC エンドポイントを使用して Amazon Keyspaces に接続する
- チュートリアル: Apache Spark と統合してデータをインポートまたはエクスポートする
- チュートリアル: Amazon Elastic Kubernetes Service でホストされているコンテナ化されたアプリケーションから接続する
- チュートリアル: を使用して Amazon Keyspaces テーブルを Amazon S3 にエクスポートする AWS Glue

チュートリアル: インターフェイス VPC エンドポイントを使用して Amazon Keyspaces に接続する

このチュートリアルでは、Amazon Keyspaces 用のインターフェイス VPC エンドポイントのセットアップ方法と使用方法を説明します。

インターフェイス VPC エンドポイントでは、Amazon VPC で実行されている仮想プライベートクラウド (VPC) と Amazon Keyspaces 間のプライベート通信ができます。インターフェイス VPC エンドポイントは AWS PrivateLink、VPC と AWS サービス間のVPCs。 AWS 詳細については、「the section called “インターフェイス VPC エンドポイントの使用”」を参照してください。

トピック

- チュートリアルの前提条件と考慮事項

- ステップ 1: Amazon EC2 インスタンスを起動する
- ステップ 2: Amazon EC2 インスタンスを設定する
- ステップ 3: Amazon Keyspaces 用の VPC エンドポイントを作成する
- ステップ 4: VPC エンドポイント接続の権限を設定する
- ステップ 5: CloudWatch でモニタリングを設定する
- ステップ 6: (オプション) アプリケーションの接続プールサイズを設定するベストプラクティス
- ステップ 7: (オプション) クリーンアップする

チュートリアルの前提条件と考慮事項

このチュートリアルを開始する前に、 AWS 「」のセットアップ手順に従ってくださいAmazon Keyspaces (Apache Cassandra 向け) へのアクセス。これらのステップには、Amazon Keyspaces にアクセスできる AWS Identity and Access Management (IAM) プリンシパルへのサインアップ AWS と作成が含まれます。IAM ユーザーの名前とアクセスキーは、このチュートリアルの後半で必要になるので、メモしておいてください。

このチュートリアルの後半では、VPC エンドポイントからの接続をテストします。そのために、名前 `myKeyspace` のキー空間と少なくとも 1 つのテーブルを作成します。詳細な手順については、「入門」を参照してください。

前提条件のステップが完了したら、ステップ 1: Amazon EC2 インスタンスを起動する に進みます。

ステップ 1: Amazon EC2 インスタンスを起動する

このステップでは、デフォルトの Amazon VPC で Amazon EC2 インスタンスを起動します。これで、Amazon Keyspaces 用の VPC エンドポイントを作成して使用できます。

Amazon EC2 インスタンスを起動するには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. [Launch Instance (インスタンスを起動)] を選択して、以下を実行します。

   EC2 コンソールダッシュボードから、[Launch instance (インスタンスを起動する)] ボックス内で [Launch instance (インスタンスを起動する)] を選び、表示されるオプションから [Launch instance (インスタンスを起動する)] を選びます。

   [Names and tags (名前とタグ)] の [Name (名前)] には、インスタンス用にわかりやすい名前を入力します。

アプリケーションと OS イメージ (Amazon マシンイメージ):

- [Quick Start (クイックスタート)] を選択し、[Ubuntu] を選択します。これが、インスタンスのオペレーティングシステム (OS) です。
- Amazon マシンイメージ (AMI) では、無料利用枠対象としてマークされているデフォルトのイメージを使用できます。[Amazon Machine Image (AMI) (Amazon マシンイメージ (AMI))] はインスタンスのテンプレートとして機能する基本設定です。

[インスタンスタイプ] で:

- [インスタンスタイプ] リストから、t2.micro インスタンスタイプを選択します。デフォルトではこれが選択されています。

[キーペア名] の場合、[キーペア (ログイン)] で、このチュートリアルで使用する次のオプションのいずれかを選択します。

- Amazon EC2 キーペアがない場合は、[Create a new key pair (新しいキーペアの作成)] を選択して画面に表示される指示に従います。プライベートキーファイル (.pem ファイル) のダウンロードを求めるプロンプトが表示されます。このファイルは、後で Amazon EC2 インスタンスにログインするときに必要になるので、ファイルパスを書き留めておいてください。
- 既存の Amazon EC2 キーペアがすでにある場合は、[Select a key pair (キーペアの選択)] を選択して、リストからキーペアを選択します。Amazon EC2 インスタンスにログインするには、既にプライベートキーファイル (.pem ファイル) が利用可能になっている必要があります。

[ネットワーク設定] の下:

- [Edit (編集)] を選択します。
- [Select an existing security group (既存のセキュリティグループの選択)] を選択します。
- セキュリティグループのリストで、[default (デフォルト)] を選択します。これは VPC のデフォルトのセキュリティグループです。

[概要] に進みます。

- [概要] パネルで、インスタンス設定の概要を確認します。準備ができたら、[インスタンスの起動] を選択します。

3. 新しい Amazon EC2 インスタンスの完了画面で、[インスタンスに接続] タイルを選択します。次の画面には、新しいインスタンスに接続するために必要な情報と必要なステップが表示されます。以下の情報を書き留めておきます。

   - キーファイルを保護するサンプルコマンド
   - 接続文字列
   - パブリック IPv4 DNS DNS 名

   このページの情報をメモしたら、このチュートリアルの次のステップに進むことができます (ステップ 2: Amazon EC2 インスタンスを設定する)。

> **Note**
>
> Amazon EC2 インスタンスが使用可能になるまでに数分かかります。次のステップに行く前に、[Instance State (インスタンスの状態)] が `running` で、その [Status Checks (ステータスチェック)] がすべてパスしていることを確認します。

ステップ 2: Amazon EC2 インスタンスを設定する

Amazon EC2 インスタンスの使用準備が整ったら、インスタンスにログインして、初めての使用のための準備をします。

> **Note**
>
> 次のステップは、Linux を実行するコンピュータから Amazon EC2 インスタンスに接続することを前提に説明します。その他の接続方法については、「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。

Amazon EC2 インスタンスに設定するには

1. Amazon EC2 インスタンスへのインバウンド SSH トラフィックを認可します。このために、新しい EC2 セキュリティグループを作成して、そのセキュリティグループを EC2 インスタンスに割り当てます。

   a. ナビゲーションペインで、[Security Groups (セキュリティグループ)] を選択します。

   b. [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の作成] ウィンドウで、以下を行います。

      - [Security group name (セキュリティグループ名)] に、セキュリティグループの名前を入力します。例: `my-ssh-access`
      - [Description (説明)] — セキュリティグループの簡単な説明を入力します。
      - VPC — デフォルトの VPCを選択します。
      - [Inbound rules (インバウンドルール)] セクションで、[Add Rule (ルールの追加)] を選択して、次の操作を行います
        - [Type (タイプ)] — SSH を選択します。
        - [Source (ソース)] — My IP を選択します。
        - [ルールを追加] を選択してください。

      ページの下部で、構成設定を確認し、[セキュリティグループを作成] を選択します。

   c. 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。

   d. ステップ 1: Amazon EC2 インスタンスを起動する で起動した Amazon EC2 インスタンスを選択します。

   e. [アクション]、[セキュリティ]、[セキュリティグループの変更] の順に選択します。

   f. [Change Security Groups (セキュリティグループの変更)] で、この手順で先に作成したセキュリティグループを選択します (例: `my-ssh-access`)。既存の `default` のセキュリティグループも選択する必要があります。設定を確認し、[セキュリティグループを割り当て] を選択します。

2. 次のコマンドを実行して、プライベートキーファイルをアクセスから保護します。このステップをスキップすると、接続は失敗します。

```
chmod 400 path_to_file/my-keypair.pem
```

3. 次の例のように、`ssh` コマンドを使用して Amazon EC2 インスタンスにログインします。

```
ssh -i path_to_file/my-keypair.pem ubuntu@public-dns-name
```

プライベートキーファイル (.pem ファイル) とインスタンスのパブリック DNS 名を指定してください。(「ステップ 1: Amazon EC2 インスタンスを起動する」を参照してください)。

ログイン ID は ubuntu です。パスワードは不要です。

Amazon EC2 インスタンスへの接続を許可する方法の詳細と AWS CLI の手順については、「Amazon EC2 ユーザーガイド」の「Linux インスタンス用のインバウンドトラフィック承認」を参照してください。

4. AWS Command Line Interfaceの最新バージョンをダウンロードしてインストールします。

   a. unzip をインストールします。

   ```
   sudo apt install unzip
   ```

   b. AWS CLIで zip ファイルをダウンロードします。

   ```
   curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
    "awscliv2.zip"
   ```

   c. ファイル を解凍します。

   ```
   unzip awscliv2.zip
   ```

   d. をインストールします AWS CLI。

   ```
   sudo ./aws/install
   ```

   e. AWS CLI インストールのバージョンを確認します。

   ```
   aws --version
   ```

   出力は次のようになります:

   ```
   aws-cli/2.9.19 Python/3.9.11 Linux/5.15.0-1028-aws exe/x86_64.ubuntu.22 prompt/
   off
   ```

5. 次の例に示すように、 AWS 認証情報を設定します。プロンプトが表示されたら、 AWS アクセスキー ID、シークレットキー、デフォルトのリージョン名を入力します。

```
aws configure

AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]:
```

6. Amazon Keyspaces までの cqlsh 接続で、VPC エンドポイントが正しく設定されていることを確認してください。でローカル環境または Amazon Keyspaces CQL エディタを使用する場合 AWS Management Console、接続は自動的に VPC エンドポイントではなくパブリックエンドポイントを通過します。このチュートリアルの VPC エンドポイント接続のデストで cqlsh を使用するには、事前に、cqlsh を使用した Amazon Keyspaces への接続 に記載されているセットアップ手順を済ませておいてください。

これで、Amazon Keyspaces 用の VPC エンドポイントを作成する準備ができました。

ステップ 3: Amazon Keyspaces 用の VPC エンドポイントを作成する

このステップでは、 AWS CLIで Amazon Keyspaces 用の VPC エンドポイントを作成します。VPC コンソールで VPC エンドポイントを作成するには、『AWS PrivateLink ガイド』の「VPC エンドポイントの作成」の手順に従ってください。サービス名をフィルタリングするときは、**Cassandra** と入力します。

を使用して VPC エンドポイントを作成するには AWS CLI

1. 開始する前に、パブリックエンドポイントで Amazon Keyspaces と通信できることを確認します。

```
aws keyspaces list-tables --keyspace-name 'myKeyspace'
```

出力には、指定されたキー空間に含まれている Amazon Keyspaces テーブルのリストが表示されます。テーブルがない場合、リストは空になります。

```
{
    "tables": [
        {
```

```
            "keyspaceName": "myKeyspace",
            "tableName": "myTable1",
            "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
catalog/table/myTable1"
        },
        {
            "keyspaceName": "myKeyspace",
            "tableName": "myTable2",
            "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
catalog/table/myTable2"
        }
    ]
}
```

2. Amazon Keyspaces が現在の AWS リージョンで VPC エンドポイントを作成するための利用可能なサービスであることを確認します。(コマンドは太字で示され、その後に出力例が続きます。)

```
aws ec2 describe-vpc-endpoint-services

{
    "ServiceNames": [
        "com.amazonaws.us-east-1.cassandra",
        "com.amazonaws.us-east-1.cassandra-fips"
    ]
}
```

出力例では、Amazon Keyspaces が利用可能なサービスの 1 つなので、VPC エンドポイントの作成を続けることができます。

3. VPC 識別子を決定します。

```
aws ec2 describe-vpcs

{
    "Vpcs": [
        {
            "VpcId": "vpc-a1234bcd",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "111.31.0.0/16",
            "IsDefault": true
```

```
        }
    ]
}
```

出力例では、VPC ID は vpc-a1234bcd です。

4. フィルターで VPC のサブネットに関する詳細を収集します。

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=vpc-a1234bcd"

{
     {
    "Subnets":[
       {
          "AvailabilityZone":"us-east-1a",
          "AvailabilityZoneId":"use2-az1",
          "AvailableIpAddressCount":4085,
          "CidrBlock":"111.31.0.0/20",
          "DefaultForAz":true,
          "MapPublicIpOnLaunch":true,
          "MapCustomerOwnedIpOnLaunch":false,
          "State":"available",
          "SubnetId":"subnet-920aacf9",
          "VpcId":"vpc-a1234bcd",
          "OwnerId":"111122223333",
          "AssignIpv6AddressOnCreation":false,
          "Ipv6CidrBlockAssociationSet":[

          ],
          "SubnetArn":"arn:aws:ec2:us-east-1:111122223333:subnet/subnet-920aacf9",
          "EnableDns64":false,
          "Ipv6Native":false,
          "PrivateDnsNameOptionsOnLaunch":{
             "HostnameType":"ip-name",
             "EnableResourceNameDnsARecord":false,
             "EnableResourceNameDnsAAAARecord":false
          }
       },
       {
          "AvailabilityZone":"us-east-1c",
          "AvailabilityZoneId":"use2-az3",
          "AvailableIpAddressCount":4085,
          "CidrBlock":"111.31.32.0/20",
          "DefaultForAz":true,
```

```
            "MapPublicIpOnLaunch":true,
            "MapCustomerOwnedIpOnLaunch":false,
            "State":"available",
            "SubnetId":"subnet-4c713600",
            "VpcId":"vpc-a1234bcd",
            "OwnerId":"111122223333",
            "AssignIpv6AddressOnCreation":false,
            "Ipv6CidrBlockAssociationSet":[

            ],
            "SubnetArn":"arn:aws:ec2:us-east-1:111122223333:subnet/subnet-4c713600",
            "EnableDns64":false,
            "Ipv6Native":false,
            "PrivateDnsNameOptionsOnLaunch":{
               "HostnameType":"ip-name",
               "EnableResourceNameDnsARecord":false,
               "EnableResourceNameDnsAAAARecord":false
            }
         },
         {
            "AvailabilityZone":"us-east-1b",
            "AvailabilityZoneId":"use2-az2",
            "AvailableIpAddressCount":4086,
            "CidrBlock":"111.31.16.0/20",
            "DefaultForAz":true,
            "MapPublicIpOnLaunch":true,


         }
      ]
}
```

出力例では、2 つのサブネット ID (subnet-920aacf9 と subnet-4c713600) を使用できます。

5. VPC エンドポイントを作成します。--vpc-id パラメータで、前のステップの VPC ID を指定します。--subnet-id パラメータで、前のステップのサブネット ID を指定します。--vpc-endpoint-type パラメータで、エンドポイントをインターフェイスとして定義します。コマンドの詳細については、『AWS CLI コマンドリファレンス』の「create-vpc-endpoint」を参照してください。

```
aws ec2 create-vpc-endpoint --vpc-endpoint-type Interface --vpc-id vpc-a1234bcd
 --service-name com.amazonaws.us-east-1.cassandra --subnet-id subnet-920aacf9
 subnet-4c713600

{
    "VpcEndpoint": {
        "VpcEndpointId": "vpce-000ab1cdef23456789",
        "VpcEndpointType": "Interface",
        "VpcId": "vpc-a1234bcd",
        "ServiceName": "com.amazonaws.us-east-1.cassandra",
        "State": "pending",
        "RouteTableIds": [],
        "SubnetIds": [
            "subnet-920aacf9",
            "subnet-4c713600"
        ],
        "Groups": [
            {
                "GroupId": "sg-ac1b0e8d",
                "GroupName": "default"
            }
        ],
        "IpAddressType": "ipv4",
        "DnsOptions": {
            "DnsRecordIpType": "ipv4"
        },
        "PrivateDnsEnabled": true,
        "RequesterManaged": false,
        "NetworkInterfaceIds": [
            "eni-043c30c78196ad82e",
            "eni-06ce37e3fd878d9fa"
        ],
        "DnsEntries": [
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz.cassandra.us-
east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz-us-
east-1a.cassandra.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
```

```
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz-us-
east-1c.cassandra.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz-us-
east-1b.cassandra.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz-us-
east-1d.cassandra.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
            {
                "DnsName": "cassandra.us-east-1.amazonaws.com",
                "HostedZoneId": "ZONEIDPENDING"
            }
        ],
        "CreationTimestamp": "2023-01-27T16:12:36.834000+00:00",
        "OwnerId": "111122223333"
    }
}

}
```

ステップ 4: VPC エンドポイント接続の権限を設定する

このステップの手順では、Amazon Keyspaces でVPC エンドポイントを使用するためのルールと権限の設定方法を説明します。

TCP インバウンドトラフィックを許可するインバウンドルールを新しいエンドポイントに設定するには

1. Amazon VPC コンソールの左側のパネルで、[エンドポイント] を選択し、前のステップで作成したエンドポイントを選択します。
2. [セキュリティグループ] を選択し、このエンドポイントに関連付けられているセキュリティグループを選択します。
3. [インバウンドルール] を選択し、[インバウンドルールを編集] を選択します。

4. インバウンドルールを追加し、[タイプ] を [CQLSH/CASSANDRA] にします。これにより、[ポート範囲] が自動的に [9142] に設定されます。
5. 新しいインバウンドルールを保存するには、[ルールを保存] を選択します。

IAM ユーザーの権限を設定するには

1. Amazon Keyspaces までの接続に使用する IAM ユーザーに、適切な権限があることを確認します。 AWS Identity and Access Management (IAM) では、 AWS 管理ポリシーを使用してAmazonKeyspacesReadOnlyAccess、IAM ユーザーに Amazon Keyspaces への読み取りアクセスを許可できます。

   a. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
   b. IAM コンソールダッシュボードで [Users (ユーザー)] を選択してから、リストから IAM ユーザーを選択します。
   c. [Summary (概要)] ページで、[Add permissions (許可の追加)] を選択します。
   d. [Attach existing policies directly (既存のポリシーを直接アタッチする)] を選択します。
   e. ポリシーリストから AmazonKeyspacesReadOnlyAccess、[Next: Review (次へ: 確認)] を選択します。
   f. [Add permissions (許可の追加)] を選択します。
2. VPC エンドポイント経由で Amazon Keyspaces にアクセスできることを確認します。

```
aws keyspaces list-tables --keyspace-name 'my_Keyspace'
```

必要に応じて、Amazon Keyspaces の他の AWS CLI コマンドを試すことができます。詳細については、『AWS CLI コマンドリファレンス』を参照してください。

> **Note**
>
> 以下のポリシーにあるように、IAM ユーザーやロールが Amazon Keyspaces にアクセスに最低限必要な権限は、システムテーブルの読み取り権限です。このポリシーベースの許可に関する詳細については、「the section called “アイデンティティベースのポリシーの例”」を参照してください。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:555555555555:/keyspace/system*"
         ]
      }
   ]
}
```

3. IAM ユーザーに VPC の Amazon EC2 インスタンスの読み取りアクセス権限を与えます。

   Amazon Keyspaces を VPC エンドポイントで使用するとき、Amazon Keyspaces にアクセスする IAM ユーザーまたはロールに、Amazon EC2 インスタンスと VPC に対する読み取り専用アクセス権限を設定して、エンドポイントとネットワークインターフェイスのデータを収集する必要があります。Amazon Keyspaces はこの情報を `system.peers` テーブルに保存し、それで接続を管理します。

   > **Note**
   >
   > 管理ポリシー `AmazonKeyspacesReadOnlyAccess_v2` と `AmazonKeyspacesFullAccess` には、Amazon Keyspaces が Amazon EC2 インスタンスにアクセスして、使用可能なインターフェイス VPC エンドポイントに関する情報を読み取るためのアクセス権限が設定されています。

   a. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
   b. IAM コンソールのダッシュボードで [ポリシー] を選択します。
   c. [ポリシーを作成] を選択し、[JSON] タブを選択します。
   d. 次のポリシーをコピーして [次へ: タグ] を選択します。

      ```
      {
      ```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListVPCEndpoints",
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcEndpoints"
         ],
         "Resource": "*"
      }
   ]
}
```

e. [ポリシーを確認] を選択してポリシーの名前 keyspacesVPCendpoint を入力し、[ポリシーを作成] を選択します。

f. IAM コンソールダッシュボードで [Users (ユーザー)] を選択して、リストから IAM ユーザーを選択します。

g. [Summary (概要)] ページで、[Add permissions (許可の追加)] を選択します。

h. [Attach existing policies directly (既存のポリシーを直接アタッチする)] を選択します。

i. ポリシーリストから [keyspacesVPCendpoint] を選択し、次に [Next: Review (次へ: 確認)] を選択します。

j. [Add permissions (許可の追加)] を選択します。

4. Amazon Keyspaces system.peers テーブルが VPC 情報で更新されていることを確認するには、cqlsh で Amazon EC2 インスタンスから次のクエリを実行します。ステップ 2 で Amazon EC2 インスタンスに cqlsh をインストールしていない場合は、the section called “cqlsh-expansion の使用” の指示に従ってください。

```
SELECT peer FROM system.peers;
```

出力は、 AWS リージョンの VPC とサブネットの設定に応じて、プライベート IP アドレスを持つノードを返します。

```
peer
---------------
112.11.22.123
112.11.22.124
112.11.22.125
```

> **Note**
>
> Amazon Keyspaces までの `cqlsh` 接続で、VPC エンドポイントが正しく設定されていることを確認してください。ローカル環境または AWS Management Consoleの Amazon Keyspaces CQL エディタを使用する場合、接続ルートは VPC エンドポイントではなくパブリックエンドポイントを自動的に経由します。9 個の IP アドレスが表示される場合、これらはパブリックエンドポイント接続の際に Amazon Keyspaces によって `system.peers` テーブルに自動的に書き込まれるエントリです。

ステップ 5: CloudWatch でモニタリングを設定する

このステップでは、Amazon CloudWatch で Amazon Keyspaces への VPC エンドポイント接続をモニタリングする方法を説明します。

AWS PrivateLink は、インターフェイスエンドポイントに関するデータポイントを CloudWatch に発行します。メトリクスを使用して、システムが正常に実行されていることを確認できます。CloudWatch の `AWS/PrivateLinkEndpoints` 名前空間には、インターフェイスエンドポイントのメトリックスが含まれます。詳細については、『AWS PrivateLink ガイド』の「 AWS PrivateLink用の「CloudWatch メトリクス」を参照してください。

VPC エンドポイントメトリクスを含む CloudWatch ダッシュボードを作成する

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。次に、[ダッシュボードを作成] を選択します。ダッシュボードの名前を入力し、[作成] を選択します。
3. [ウィジェットを追加] で [番号] を選択します。
4. [メトリクス] で [AWS/プライベートリンクエンドポイント] を選択します。
5. エンドポイントタイプ、サービス名、VPC エンドポイント ID、VPC ID を選択します。
6. グラフ化するメトリクス `ActiveConnections` と `NewConnections` を選択し、[ウィジェットを作成] を選択します。
7. ダッシュボードを保存します。

ActiveConnections メトリクスは、過去 1 分間にエンドポイントが受信した同時アクティブ接続の数として定義されます。NewConnections メトリクスは、過去 1 分間にエンドポイントを通じて確立された新しい接続の数として定義されます。

ダッシュボードの作成の詳細については、『CloudWatch ユーザーガイド』の「ダッシュボードを作成」を参照してください。

ステップ 6: (オプション) アプリケーションの接続プールサイズを設定するベストプラクティス

このセクションでは、アプリケーションのクエリースループット要件に基づいて理想的な接続プールサイズを決定する方法の概要を説明します。

Amazon Keyspaces では、TCP 接続ごとに 1 秒あたり最大 3,000 件の CQL クエリを実行できます。そのため、ドライバーが Amazon Keyspaces で確立できる接続数には実質的に制限がありません。ただし、Amazon Keyspaces を VPC エンドポイント接続で使用する場合は、接続プールのサイズをアプリケーションの要件に合わせて、使用可能なエンドポイント数を検討することをお勧めします。

接続プールのサイズはクライアントドライバーで設定します。たとえば、ローカルプールサイズが 2 で、3 つのアベイラビリティーゾーンに作成された VPC インターフェイスエンドポイントに基づいて、ドライバーはクエリ用に 6 本の接続 (制御接続を含む合計 7 本) を確立します。これら 6 本の接続を使用すると、1 秒あたり最大 18,000 件の CQL クエリをサポートできます。

アプリケーションで 1 秒あたり 40,000 件の CQL クエリをサポートする必要がある場合は、必要なクエリの数から逆算して、必要な接続プールのサイズを決定します。1 秒あたり 40,000 件の CQL クエリをサポートするには、ローカルプールのサイズを 5 以上に設定する必要があります。これにより、1 秒あたり最低 45,000 件の CQL クエリをサポートできます。

AWS/Cassandra 名前空間の PerConnectionRequestRateExceeded CloudWatch メトリクスで、接続 1 つあたりの 1 秒あたりの最大オペレーション数のクォータを超えているかどうかをモニタリングできます。PerConnectionRequestRateExceeded メトリクスは、接続ごとのリクエストレートのクォータを超える Amazon Keyspaces へのリクエスト数を示します。

このステップのコード例では、インターフェイス VPC エンドポイントを使用する際の接続プールのサイズを見積もって設定する方法を示します。

Java

Java ドライバーでは、プールごとの接続数を設定できます。Java クライアントドライバ接続の一式の例については、「the section called “Cassandra Java クライアントドライバーの使用”」を参照してください。

クライアントドライバーを起動すると、まず、スキーマやトポロジーの変更などの管理タスクのための制御接続がつながります。次に、追加の接続が作成されます。

次の例では、ローカルプールサイズドライバー設定は 2 に指定されています。VPC エンドポイントが VPC 内の 3 つのサブネットにわたって作成されている場合、次の式に示すように、インターフェイスエンドポイントは CloudWatch で 7 NewConnections になります。

```
NewConnections = 3 (VPC subnet endpoints created across) * 2 (pool size) + 1
 ( control connection)
```

```
datastax-java-driver {

    basic.contact-points = [ "cassandra.us-east-1.amazonaws.com:9142"]
    advanced.auth-provider{
        class = PlainTextAuthProvider
           username = "ServiceUserName"
           password = "ServicePassword"
    }
    basic.load-balancing-policy {
        local-datacenter = "us-east-1"
        slow-replica-avoidance = false
    }

    advanced.ssl-engine-factory {
        class = DefaultSslEngineFactory
        truststore-path = "./src/main/resources/cassandra_truststore.jks"
        truststore-password = "my_password"
        hostname-validation = false
      }
    advanced.connection {
        pool.local.size = 2
          }
}
```

アクティブな接続の数が、設定したプールサイズ (サブネット全体の集計) + 1 つの制御接続と一致しない場合、何らかの原因で接続が作成を妨げられています。

Node.js

Node.js ドライバーでプールごとの接続数を設定できます。Node.js クライアントドライバ接続の一式の例については、「the section called “Cassandra Node.js クライアントドライバーの使用”」を参照してください。

次のコード例では、ローカルプールサイズドライバー設定は 1 と指定されています。VPC エンドポイントが VPC 内の 4 つのサブネットにわたって作成されている場合、次の式に示すように、インターフェイスエンドポイントは CloudWatch で 5 NewConnectionsになります。

```
NewConnections = 4 (VPC subnet endpoints created across) * 1 (pool size) + 1
 ( control connection)
```

```
const cassandra = require('cassandra-driver');
const fs = require('fs');
const types = cassandra.types;
const auth = new cassandra.auth.PlainTextAuthProvider('ServiceUserName',
 'ServicePassword');
const sslOptions1 = {
         ca: [
                    fs.readFileSync('/home/ec2-user/sf-class2-root.crt', 'utf-8')],
                    host: 'cassandra.us-east-1.amazonaws.com',
                    rejectUnauthorized: true
        };
const client = new cassandra.Client({
                   contactPoints: ['cassandra.us-east-1.amazonaws.com'],
                   localDataCenter: 'us-east-1',
                   pooling: { coreConnectionsPerHost: { [types.distance.local]:
 1 } },
                   consistency: types.consistencies.localQuorum,
                   queryOptions: { isIdempotent: true },
                   authProvider: auth,
                   sslOptions: sslOptions1,
                   protocolOptions: { port: 9142 }
        });
```

ステップ 7: (オプション) クリーンアップする

このチュートリアルで作成したリソースを削除する場合は、次の手順に従ってください。

Amazon Keyspaces 用の VPC エンドポイントを削除するには

1. Amazon EC2 インスタンスにログインします。
2. Amazon Keyspaces に使用されている VPC エンドポイント ID を決定します。grep パラメータを省略すると、すべてのサービスの VPC エンドポイント情報が表示されます。

```
aws ec2 describe-vpc-endpoint-services | grep ServiceName | grep cassandra

{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.cassandra",
        "RouteTableIds": [],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

   出力例では、VPC エンドポイント ID は vpce-9b15e2f2 です。
3. VPC エンドポイントを削除します。

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2

{
    "Unsuccessful": []
}
```

   空の配列 [] は成功を示します (失敗したリクエストはありませんでした)。

Amazon EC2 インスタンスを終了するには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. 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。
3. Amazon EC2 インスタンスを選択します。
4. [Actions (アクション)] に、[Instance State (インスタンスの状態)] を選択して、[Terminate (終了)] の順に選択します。

5. 確認ウィンドウで、[Yes, Terminate (はい、終了します)] を選択します。

チュートリアル: Apache Spark と統合してデータをインポートまたはエクスポートする

Apache Spark は、大規模データ分析のためのオープンソースエンジンです。Apache Spark では、Amazon Keyspaces に保存されているデータの分析を効率的に実行できます。Amazon Keyspaces を利用すれば、Spark からの分析データへの一貫した 1 桁ミリ秒単位の読み取りアクセスをアプリケーションに提供することもできます。オープンソースの Spark Cassandra コネクタがあれば、Amazon Keyspaces と Spark 間でデータの読み取りや書き込みが簡単にできます。

Amazon Keyspaces は Spark Cassandra コネクタをサポートしています。そのため、フルマネージドのサーバーレスデータベースサービスで、Spark ベースの分析パイプラインで効率的に Cassandra ワークロードを実行できます。Amazon Keyspaces があれば、Spark で、テーブルと同じ基盤インフラストラクチャリソースをめぐって競合の心配をする必要がなくなります。Amazon Keyspaces テーブルでは、アプリケーションのトラフィックに基づいて拡大、縮小が行われます。

以下のチュートリアルでは、Spark Cassandra コネクタで Amazon Keyspaces にデータを読み書きするために必要な手順とベストプラクティスについて説明します。このチュートリアルでは、Spark Cassandra コネクタでファイルからデータを読み込み、Amazon Keyspaces テーブルに書き込んで、Amazon Keyspaces にデータを移行する方法を示します。次に、このチュートリアルでは、Spark Cassandra コネクタで Amazon Keyspaces からデータを読み取る方法を説明します。この方法で、Spark ベースの分析パイプラインで Cassandra ワークロードを実行できます。

トピック

- Spark Cassandra コネクタで Amazon Keyspaces までの接続をつなぐための前提条件
- ステップ 1: Amazon Keyspaces と Apache Cassandra Spark コネクタとの統合を設定する
- ステップ 2: Apache Cassandra Spark コネクタを設定する
- ステップ 3: アプリケーション設定ファイルを作成する
- ステップ 4: Amazon Keyspaces でソースデータとターゲットテーブルを準備する
- ステップ 5: Apache Cassandra Spark コネクタで Amazon Keyspaces データの書き込みや読み込みを行う
- Amazon Keyspaces で Spark Cassandra コネクタを使用する場合の一般的なエラーのトラブルシューティング

Spark Cassandra コネクタで Amazon Keyspaces までの接続をつなぐための前提条件

Spark Cassandra コネクタで Amazon Keyspaces に接続する前に、以下のソフトウェアがインストールされていることを確認してください。Amazon Keyspaces と Spark Cassandra コネクタの互換性は、以下の推奨バージョンでテスト済みです。

- Java バージョン 8
- Scala 2.12
- Spark 3.4
- Cassandra コネクタ 2.5 以上
- Cassandra ドライバー 4.12

1. https://www.scala-lang.org/download/scala2.html は、次の手順に従ってインストールしてください。
2. Spark 3.4.1 は、次の例に従ってインストールしてください。

```
curl -o spark-3.4.1-bin-hadoop3.tgz -k https://dlcdn.apache.org/spark/spark-3.4.1/
spark-3.4.1-bin-hadoop3.tgz

# now to untar
tar -zxvf spark-3.4.1-bin-hadoop3.tgz

# set this variable.
export SPARK_HOME=$PWD/spark-3.4.1-bin-hadoop3
```

ステップ 1: Amazon Keyspaces と Apache Cassandra Spark コネクタとの統合を設定する

このステップでは、アカウントのパーティショナーに Apache Spark コネクタと互換性があることを確認し、必要な IAM 権限を設定します。以下のベストプラクティスは、テーブルに十分な読み取り/書き込みキャパシティをプロビジョニングするときに利用できます。

1. `Murmur3Partitioner` パーティショナーがアカウントのデフォルトパーティショナーであることを確認してください。このパーティショナーには Spark Cassandra コネクタと互換性があ

ります。パーティショナーとその変更方法の詳細については、「the section called “パーティショナーの操作”」を参照してください。

2. Apache Spark のインターフェイス VPC エンドポイントで、Amazon Keyspaces の IAM アクセス権限を設定します。

   - 以下に示す IAM ポリシーの例に従って、ユーザーテーブルに対する読み取り/書き込みアクセス権限と、システムテーブルに対する読み取りアクセス権限を割り当てます。
   - VPC エンドポイント経由で Spark によって Amazon Keyspaces にアクセスするクライアントには、使用可能なインターフェイス VPC エンドポイントを system.peers テーブルをポピュレートする必要があります。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select",
            "cassandra:Modify"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      },
      {
         "Sid":"ListVPCEndpoints",
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcEndpoints"
         ],
         "Resource":"*"
      }
   ]
}
```

3. Amazon Keyspaces テーブルで Spark Cassandra コネクタからのトラフィックをサポートできる十分な読み取り/書き込みスループットキャパシティを設定する場合、以下のベストプラクティスを検討してください。

   - 最初は、シナリオのテストに役立つオンデマンドキャパシティから始めます。
   - コネクタからのトラフィックにレートリミッターを使用し、自動スケーリングでプロビジョニングされたキャパシティを使用するようにテーブルを設定して本番環境のテーブルスループットのコストを最適化します。詳細については、「the section called “自動スケーリングによるスループットキャパシティの管理”」を参照してください。
   - Cassandra ドライバーに付属の固定レートリミッターを使用できます。AWS サンプルリポジトリには、Amazon Keyspaces に合わせたレートリミッターがいくつかあります。
   - キャパシティイベントの詳細については、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。

ステップ 2: Apache Cassandra Spark コネクタを設定する

Apache Spark は、さまざまな方法で設定できる汎用コンピューティングプラットフォームです。Spark と Spark Cassandra Connector を Amazon Keyspaces との統合を設定するには、以下のセクションで説明する最小設定から始め、後でワークロードに応じてそれらを拡張することをお勧めします。

- 8 MB 未満のサイズの Spark パーティションを作成します。

  Spark では、パーティションは並列で実行可能なデータのアトミックチャンクを表します。Spark Cassandra コネクタで Amazon Keyspaces にデータを書き込むとき、Spark パーティションが小さいほど、タスクで書き込まれるレコードの量は少なくなります。Spark タスクで複数のエラーが発生した場合、指定された回数の再試行回数を過ぎると失敗します。大きなタスクのやり直しや、大量のデータの再処理を防ぐため、Spark パーティションのサイズは小さく設定してください。

- 実行者 1 人あたりの同時書き込み回数を少なく設定して、再試行回数を多く設定してください。

  Amazon Keyspaces は、オペレーションがタイムアウトになると、キャパシティ不足のエラーを Cassandra ドライバーに返します。Spark Cassandra コネクタは `MultipleRetryPolicy` でリクエストの透過的な再試行を試みるため、設定されているタイムアウト時間を変更しても、キャパシティ不足によるタイムアウトには対処できません。再試行によるドライバーの接続プールの圧迫を防ぐには、エグゼキューターあたりの同時書き込み回数を少なく設定して、再試行回数を多く設定してください。以下のコードスニペットに、この例を示します。

```
spark.cassandra.query.retry.count = 500
spark.cassandra.output.concurrent.writes = 3
```

- 総スループットを細分化し、複数の Cassandra セッションに分散させます。
  - Cassandra Spark コネクタは、Spark エクゼキューターごとに 1 つのセッションを作成します。このセッションは、必要なスループットと必要な接続数を決定するスケールの単位と考えてください。
  - エクゼキューター 1 人あたりのコア数とタスク 1 つあたりのコア数を定義するときは、最初は少ない数から始めて、必要に応じて増やしてください。
  - 一時的なエラーが発生した場合に処理できるように Spark タスク障害を設定します。アプリケーションのトラフィック特性と要件が理解できたら、spark.task.maxFailures を境界値に設定することをおすすめします。
  - たとえば、以下の設定では、エクゼキューター、セッションごとに 2 つの同時タスクを処理できます。

```
spark.executor.instances = configurable -> number of executors for the session.
spark.executor.cores = 2 -> Number of cores per executor.
spark.task.cpus = 1 -> Number of cores per task.
spark.task.maxFailures = -1
```

- バッチ処理をオフにします。
  - ランダムアクセスのパターンを改善するために、バッチ処理をオフにすることをお勧めします。以下のコードスニペットに、この例を示します。

```
spark.cassandra.output.batch.size.rows = 1 (Default = None)
spark.cassandra.output.batch.grouping.key = none (Default = Partition)
spark.cassandra.output.batch.grouping.buffer.size = 100 (Default = 1000)
```

- **SPARK_LOCAL_DIRS** は、十分なキャパシティがある高速のローカルディスクに設定してください。
  - デフォルトで、Spark はマップ出力ファイルと耐障害性のある分散データセット (RDD) を /tmp フォルダに保存します。 Spark ホストの設定によっては、デバイススタイルエラーのための空きキャパシティがなくなるおそれがあります。
  - SPARK_LOCAL_DIRS 環境変数を /example/spark-dir というディレクトリーに設定するには、以下のコマンドを使用できます。

```
export SPARK_LOCAL_DIRS=/example/spark-dir
```

ステップ 3: アプリケーション設定ファイルを作成する

オープンソースの Spark Cassandra コネクタを AmazonKeyspaces で使用するには、DataStax Java ドライバーとの接続に必要な設定をしたアプリケーション設定ファイルを用意する必要があります。接続には、サービス固有の認証情報または SigV4 プラグインを使用できます。

Starfield のデジタル証明書を TrustStore ファイルにまだ変換していない場合は、変換してください。Java ドライバー接続チュートリアルの詳細な手順については、「the section called “[開始する前に]”」を参照してください。trustStore のファイルパスとパスワードを書き留めておいてください。この情報はアプリケーション設定ファイルを作成するときに必要になります。

SigV4 認証による 接続

このセクションでは、 AWS 認証情報と SigV4 プラグインを使用して接続するときに使用できるapplication.confファイルの例を示します。まだ作成していない場合は、IAM アクセスキー (アクセスキー ID とシークレットアクセスキー) AWS を生成し、設定ファイルまたは環境変数として保存する必要があります。詳細な手順については、「the section called “ AWS 認証に必要な認証情報”」を参照してください。

次の例では、 trustStore ファイルまでのファイルパスを置き換え、パスワードを置き換えます。

```
datastax-java-driver {
        basic.contact-points = ["cassandra.us-east-1.amazonaws.com:9142"]
        basic.load-balancing-policy {
            class = DefaultLoadBalancingPolicy
            local-datacenter = us-east-1
            slow-replica-avoidance = false
        }
        basic.request {
              consistency = LOCAL_QUORUM
        }
        advanced {
                auth-provider = {
                   class = software.aws.mcs.auth.SigV4AuthProvider
                   aws-region = us-east-1
                 }
            ssl-engine-factory {
```

```
                class = DefaultSslEngineFactory
                truststore-path = "path_to_file/cassandra_truststore.jks"
                truststore-password = "password"
        hostname-validation=false
            }
    }
        advanced.connection.pool.local.size = 3
}
```

この設定ファイルを更新し、/home/user1/application.conf という名前で保存します。次の例ではこのパスを使用しています。

サービス固有の認証情報で接続する

このセクションでは、サービス固有の認証情報で接続するときに使用できるサンプルapplication.confファイルを示します。まだ作成していない場合は、Amazon Keyspaces のサービス固有の認証情報を生成する必要があります。詳細な手順については、「the section called “サービス固有の認証情報の作成”」を参照してください。

以下の例で、username と password を自分の認証情報あに置き換えます。また、 trustStore ファイルまでのファイルパスを置き換え、パスワードも置き換えてください。

```
datastax-java-driver {
        basic.contact-points = ["cassandra.us-east-1.amazonaws.com:9142"]
        basic.load-balancing-policy {
            class = DefaultLoadBalancingPolicy
            local-datacenter = us-east-1
        }
        basic.request {
              consistency = LOCAL_QUORUM
        }
        advanced {
            auth-provider = {
            class = PlainTextAuthProvider
                    username = "username"
                    password = "password"
                    aws-region = "us-east-1"
            }
            ssl-engine-factory {
                class = DefaultSslEngineFactory
                truststore-path = "path_to_file/cassandra_truststore.jks"
                truststore-password = "password"
                hostname-validation=false
```

```
            }
            metadata = {
                schema {
                     token-map.enabled = true
                }
            }
        }
}
```

この構成ファイルをコード例で使用できるように /home/user1/application.conf に更新して保存します。

固定料金で接続する

Spark エクゼキューターごとに固定レートを強制的に使用するには、リクエストスロットラーを定義します。このリクエストスロットラーで 1 秒あたりのリクエスト数を制限します。Spark Cassandra コネクタは、エクゼキューターごとに 1 つの Cassandra セッションをデプロイします。次の式を使用すると、テーブルに対して一貫したスループットを実現できます。

```
max-request-per-second * numberOfExecutors = total throughput against a table
```

この例の内容は、前に作成したアプリケーション設定ファイルに追加できます。

```
datastax-java-driver {
  advanced.throttler {
    class = RateLimitingRequestThrottler

    max-requests-per-second = 3000
    max-queue-size = 30000
    drain-interval = 1 millisecond
  }
}
```

ステップ 4: Amazon Keyspaces でソースデータとターゲットテーブルを準備する

このステップでは、サンプルデータと Amazon Keyspaces テーブルがあるソースファイルを作成します。

1. ソースファイルを作成します。次のオプションのいずれかを選択します。

- このチュートリアルでは、keyspaces_sample_table.csv という名前のカンマ区切り値 (CSV) ファイルをデータ移行用のソースファイルとして使用します。提供されたサンプルファイルには、book_awards という名前のテーブルに関する数行のデータが含まれています。

  - 次のアーカイブファイル samplemigration.zip にあるサンプル CSV ファイル (keyspaces_sample_table.csv) をダウンロードします。アーカイブを解凍し、keyspaces_sample_table.csv へのパスをメモしておきます。

- 自分の CSV ファイルで Amazon Keyspaces にデータを書き込む場合は、データがランダム化されていることを確認してください。データベースから直接読み取られるデータやフラットファイルにエクスポートされるデータは、通常、パーティションとプライマリキーによって順序付けられます。注文したデータを Amazon Keyspaces にインポートすると、Amazon Keyspaces パーティションの小さなセグメントに書き込まれて、トラフィックが偏って分散するおそれがあります。その場合、パフォーマンスが低下し、エラー率が高くなるおそれがあります。

  対照的に、トラフィックをパーティション間でより均等に分散してデータをランダム化すれば、Amazon Keyspaces に組み込まれている負荷分散機能を活かすことができます。データのランダム化にはさまざまなツールを使用できます。オープンソースツールの Shuf の使用例については、データ移行チュートリアルの「the section called “ステップ 2: データを準備する”」を参照してください。以下に示した例は、DataFrame としてデータをシャッフルする方法です。

```
import org.apache.spark.sql.functions.randval
shuffledDF = dataframe.orderBy(rand())
```

2. Amazon Keyspaces でターゲットのキースペースとテーブルを作成します。

   a. cqlsh で Amazon Keyspaces に接続し、次の例のサービスエンドポイント、ユーザー名、およびパスワードをそれぞれ自分の値に置き換えます。

```
cqlsh cassandra.us-east-2.amazonaws.com 9142 -u "111122223333" -
p "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY" --ssl
```

   b. 次の例に示すように、catalog という名前の新しいキースペースを作成します。

```
CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

c. 新しいキースペースが利用可能な状態になったら、次のコードを使用してターゲットテーブル book_awards を作成します。非同期的なリソース作成と、リソースが利用可能かどうかを確認する方法については、「the section called “キースペースの作成ステータスの確認”」を参照してください。

```
CREATE TABLE catalog.book_awards (
   year int,
   award text,
   rank int,
   category text,
   book_title text,
   author text,
   publisher text,
   PRIMARY KEY ((year, award), category, rank)
   );
```

ステップ 5: Apache Cassandra Spark コネクタで Amazon Keyspaces データの書き込みや読み込みを行う

このステップでは、まず Spark Cassandra DataFrame コネクタを使用してサンプルファイルのデータを読み込みます。次に、DataFrame のデータを、 Amazon Keyspaces テーブルに書き込みます。この部分を別々に利用して、たとえば Amazon Keyspaces テーブルにデータを移行することもできます。最後に、Spark Cassandra DataFrame コネクタで、テーブルからデータをに読み込みます。この部分を別々に利用して、たとえば、Amazon Keyspaces テーブルからデータを読み取り、Apache Spark でデータ分析を実行することができます。

1. 次の例のように Spark シェルを起動します。この例では、SigV4 認証を使用しています。注意してください。

```
./spark-shell --files application.conf --conf
 spark.cassandra.connection.config.profile.path=application.conf
 --packages software.aws.mcs:aws-sigv4-auth-cassandra-java-driver-
plugin:4.0.5,com.datastax.spark:spark-cassandra-connector_2.12:3.1.0 --conf
 spark.sql.extensions=com.datastax.spark.connector.CassandraSparkExtensions
```

2. 以下のコードで Spark Cassandra コネクタをインポートします。

```
import org.apache.spark.sql.cassandra._
```

3. CSV ファイルからデータを読み取って DataFrame に保存するには、次のコード例を利用できます。

```
var df =
 spark.read.option("header","true").option("inferSchema","true").csv("keyspaces_sample_tabl
```

次のコマンドで結果を表示できます。

```
scala> df.show();
```

出力は次のようになります。

```
+----------------+----+-----------+----+------------------+--------------------
+-------------+
|           award|year|   category|rank|            author|          book_title|
 publisher|
+----------------+----+-----------+----+------------------+--------------------
+-------------+
|Kwesi Manu Prize|2020|    Fiction|   1|        Akua Mansa|   Where did you go?|
SomePublisher|
|Kwesi Manu Prize|2020|    Fiction|   2|       John Stiles|           Yesterday|
Example Books|
|Kwesi Manu Prize|2020|    Fiction|   3|        Nikki Wolf|Moving to the Cha...|
 AnyPublisher|
|            Wolf|2020|Non-Fiction|   1|       Wang Xiulan|    History of Ideas|
Example Books|
|            Wolf|2020|Non-Fiction|   2|Ana Carolina Silva|       Science Today|
SomePublisher|
|            Wolf|2020|Non-Fiction|   3| Shirley Rodriguez|The Future of Sea...|
 AnyPublisher|
|     Richard Roe|2020|    Fiction|   1| Alejandro Rosalez|         Long Summer|
SomePublisher|
|     Richard Roe|2020|    Fiction|   2|       Arnav Desai|             The Key|
Example Books|
|     Richard Roe|2020|    Fiction|   3|     Mateo Jackson|    Inside the Whale|
 AnyPublisher|
+----------------+----+-----------+----+------------------+--------------------
+-------------+
```

DataFrame のデータのスキーマは、次の例のように確認できます。

```
scala> df.printSchema
```

出力は次のようになります。

```
root
|-- award: string (nullable = true)
|-- year: integer (nullable = true)
|-- category: string (nullable = true)
|-- rank: integer (nullable = true)
|-- author: string (nullable = true)
|-- book_title: string (nullable = true)
|-- publisher: string (nullable = true)
```

4. DataFrame のデータを次のコマンドで Amazon Keyspaces テーブルに書き込みます。

```
df.write.cassandraFormat("book_awards", "catalog").mode("APPEND").save()
```

5. データが保存されたことを確認するには、次の例のように、データをデータフレームに読み戻します。

```
var newDf = spark.read.cassandraFormat("book_awards", "catalog").load()
```

これで、データフレームに組み込まれたデータを表示できます。

```
scala> newDf.show()
```

このコマンドの出力は次のようになります。

```
+--------------------+-----------------+---------------+----------+-------------
+----+----+
|          book_title|           author|          award|  category|
 publisher|rank|year|
+--------------------+-----------------+---------------+----------+-------------
+----+----+
|         Long Summer| Alejandro Rosalez|     Richard Roe|    Fiction|
SomePublisher|   1|2020|
|    History of Ideas|       Wang Xiulan|           Wolf|Non-Fiction|Example
 Books|   1|2020|
|   Where did you go?|        Akua Mansa|Kwesi Manu Prize|    Fiction|
SomePublisher|   1|2020|
```

```
|    Inside the Whale|     Mateo Jackson|     Richard Roe|    Fiction|
 AnyPublisher|   3|2020|
|           Yesterday|       John Stiles|Kwesi Manu Prize|    Fiction|Example
 Books|   2|2020|
|Moving to the Cha...|        Nikki Wolf|Kwesi Manu Prize|    Fiction|
 AnyPublisher|   3|2020|
|The Future of Sea...| Shirley Rodriguez|            Wolf|Non-Fiction|
 AnyPublisher|   3|2020|
|       Science Today|Ana Carolina Silva|            Wolf|Non-Fiction|
SomePublisher|   2|2020|
|             The Key|       Arnav Desai|     Richard Roe|    Fiction|Example
 Books|   2|2020|
+--------------------+------------------+----------------+-----------+-------------
+----+----+
```

Amazon Keyspaces で Spark Cassandra コネクタを使用する場合の一般的なエラーのトラブルシューティング

Amazon Virtual Private Cloud で Amazon Keyspaces に接続している場合、Spark コネクタを使用するときに発生する最も一般的なエラーは、以下の設定の問題が原因です。

- VPC で使用する IAM ユーザーまたはロールに、Amazon Keyspaces の `system.peers` テーブルにアクセスするために必要な権限がない。詳細については、「the section called “インターフェイス VPC エンドポイント情報を含む `system.peers` テーブルエントリの入力”」を参照してください。
- IAM ユーザーまたはロールに、ユーザーテーブルへの必要な読み取り/書き込み許可と Amazon Keyspaces のシステムテーブルへの読み取り権限がない。詳細については、「the section called “ステップ 1: Amazon Keyspaces を設定する”」を参照してください。
- Java ドライバーの設定では、SSL/TLS 接続を作成するときにホスト名検証が無効になりません。例については「the section called “ステップ 2: ドライバーを設定する”」を参照してください。

接続トラブルシューティングの詳細な手順については、「the section called “VPC エンドポイント接続エラー”」を参照してください。

さらに、Amazon CloudWatch メトリクスは、Amazon k Keyspaces の、 Spark Cassandra コネクタ設定に関する問題のトラブルシューティングに利用できます。CloudWatch における Amazon Keyspaces の使用の詳細については、「the section called “CloudWatch によるモニタリング”」を参照してください。

以下のセクションでは、Spark Cassandra コネクタを使用する際に確認しておくと便利なメトリクスについて説明します。

PerConnectionRequestRateExceeded

Amazon Keyspaces のクォータは、接続ごとに 1 秒あたり 3,000 件のリクエストです。各 Spark エクゼキューターが Amazon Keyspaces との接続を確立します。再試行を複数回実行すると、接続ごとのリクエストレートのクォータを使い果たす可能性があります。このクォータを超えると、Amazon Keyspaces は CloudWatch で `PerConnectionRequestRateExceeded` メトリクスを出力します。

PerConnectionRequestrateExceeded イベントが、その他のシステムエラーやユーザーエラーとともに表示される場合は、Spark が接続ごとに割り当てられたリクエスト数を超えて複数回リトライを実行している可能性があります。

他のエラーのない `PerConnectionRequestRateExceeded` イベントが表示される場合は、スループットを高めるためにドライバー設定の接続数を増やすか、Spark ジョブのエグゼキューターの数を増やす必要があるかもしれません。

StoragePartitionThroughputCapacityExceeded

Amazon Keyspaces クォータは、パーティションごとに 1 秒あたり 1,000 WCU または WRU、1 秒あたり 3,000 RCU または RRU です。`StoragePartitionThroughputCapacityExceeded` CloudWatch イベントが表示される場合は、データがロード時にランダム化されていない可能性があります。データをシャッフルする方法の例については、「the section called “ステップ 4: ソースデータとターゲットテーブルを準備する”」を参照してください。

一般的なエラーおよび警告

Amazon 仮想プライベートクラウドを使用していて Amazon Keyspaces に接続している場合、Cassandra ドライバーで `system.peers` テーブル内のコントロールノードそのものに関する警告メッセージが発行されることがあります。詳細については、「the section called “よくあるエラーおよび警告”」を参照してください。この警告は、無視しても問題ありません。

チュートリアル: Amazon Elastic Kubernetes Service でホストされているコンテナ化されたアプリケーションから接続する

このチュートリアルでは、SigV4 認証を使用して Amazon Keyspaces に接続するコンテナ化されたアプリケーションを、Amazon Elastic Kubernetes Service (Amazon EKS) クラスターでホストするために必要な設定手順を学びます。

Amazon EKS はマネージドサービスであるため、Kubernetes コントロールプレーンを自前でインストール、運用、メンテナンスする必要がありません。Kubernetes は、コンテナ化されたアプリケーションのデプロイ、スケール、管理を自動化するためのオープンソースシステムです。

このチュートリアルでは、コンテナ化された Java アプリケーションを設定、構築し、Amazon EKS にデプロイする方法をステップバイステップで説明します。最終的には、このアプリケーションを実行して、Amazon Keyspaces テーブルにデータを書き込みます。

トピック

- Amazon EKS から Amazon Keyspaces に接続するための前提条件
- ステップ 1: Amazon EKS クラスターを設定し、IAM アクセス許可を設定する
- ステップ 2: アプリケーションを設定する
- ステップ 3: アプリケーションイメージを作成し、Docker ファイルを Amazon ECR リポジトリにアップロードする
- ステップ 4: アプリケーションを Amazon EKS にデプロイし、テーブルにデータを書き込む
- ステップ 5: (オプション) クリーンアップする

Amazon EKS から Amazon Keyspaces に接続するための前提条件

チュートリアルを開始する前に、次の AWS リソースを作成します。

1. このチュートリアルを開始する前に、 AWS 「」のセットアップ手順に従ってくださいAmazon Keyspaces (Apache Cassandra 向け) へのアクセス。これらのステップには、Amazon Keyspaces にアクセスできる AWS Identity and Access Management (IAM) プリンシパルへのサインアップ AWS と作成が含まれます。
2. `aws` という名前の Amazon Keyspaces キースペースと、`user` という名前のテーブルを作成します。このチュートリアルの後半で、Amazon EKS で実行されているコンテナ化されたアプリケーションからこのテーブルに書き込みます。これは、 AWS CLI または を使用して実行できます`cqlsh`。

AWS CLI

```
aws keyspaces create-keyspace --keyspace-name 'aws'
```

キースペースが作成されたことを確認するには、次のコマンドを使用します。

```
aws keyspaces list-keyspaces
```

テーブルを作成するには、次のコマンドを使用します。

```
aws keyspaces create-table --keyspace-name 'aws' --table-name 'user' --schema-
definition 'allColumns=[
            {name=username,type=text}, {name=fname,type=text},
{name=last_update_date,type=timestamp},{name=lname,type=text}],
            partitionKeys=[{name=username}]'
```

テーブルが作成されたことを確認するには、次のコマンドを使用します。

```
aws keyspaces list-tables --keyspace-name 'aws'
```

詳細については、「AWS CLI コマンドリファレンス」の「create-keyspace」と「create-table」を参照してください。

cqlsh

```
CREATE KEYSPACE aws WITH replication = {'class': 'SimpleStrategy',
 'replication_factor': '3'}  AND durable_writes = true;
CREATE TABLE aws.user (
    username text PRIMARY KEY,
    fname text,
    last_update_date timestamp,
    lname text
);
```

テーブルが作成されたことを確認するには、次のステートメントを使用します。

```
SELECT * FROM system_schema.tables WHERE keyspace_name = "aws";
```

このステートメントの出力に、作成したテーブルが表示されるはずです。テーブルが作成されるまで、時間がかかる場合があります。詳細については、「the section called “CREATE TABLE”」を参照してください。

3. Fargate - Linux ノードタイプを使用して Amazon EKS クラスターを作成します。Fargate はサーバーレスのコンピューティングエンジンです。Amazon EC2 インスタンスを管理することなく、Kubernetes ポッドをデプロイすることができます。このチュートリアルに実際に取り組む上で、すべてのコマンド例のクラスター名を変更しなくても済むように、「Amazon EKS ユーザーガイド」の「Amazon EKS – eksctl の使用を開始する」の手順に従って、my-eks-cluster という名前のクラスターを作成しておきましょう。クラスターが作成されたら、ノードと 2 つのデフォルトのポッドが正常に実行されていることを確認します。それには、次のコマンドを使用します。

```
kubectl get pods -A -o wide
```

次のような出力が表示されるはずです。

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE   IP
     NODE                                                NOMINATED NODE   READINESS
 GATES
kube-system   coredns-1234567890-abcde   1/1     Running   0          18m
 192.0.2.0   fargate-ip-192-0-2-0.region-code.compute.internal   <none>
 <none>
kube-system   coredns-1234567890-12345   1/1     Running   0          18m
 192.0.2.1   fargate-ip-192-0-2-1.region-code.compute.internal   <none>
 <none>
```

4. Docker をインストールします。Amazon EC2 インスタンスに Docker をインストールする方法については、「Amazon Elastic Container Registry ユーザーガイド」の「Docker のインストール」を参照してください。

   Docker はさまざまなオペレーティングシステムで使用できます。Ubuntu のような最新の Linux ディストリビューションや、 macOS や Windows でも使用できます。特定のオペレーティングシステムに Docker をインストールする方法の詳細については、Docker インストールガイド を参照してください。

5. Amazon ECR リポジトリを作成します。Amazon ECR は、任意の CLI で Docker イメージをプッシュ、プル、管理するために使用できる AWS マネージドコンテナイメージレジストリサービスです。Amazon ECR リポジトリの詳細については、「Amazon Elastic Container Registry

ユーザーガイド」を参照してください。次のコマンドを使用して、my-ecr-repository という名前のリポジトリを作成できます。

```
aws ecr create-repository --repository-name my-ecr-repository
```

前提条件のステップが完了したら、the section called “ステップ 1: Amazon EKS クラスターの設定” に進みます。

ステップ 1: Amazon EKS クラスターを設定し、IAM アクセス許可を設定する

Amazon EKS クラスターを設定し、Amazon EKS サービスアカウントから Amazon Keyspaces テーブルへの接続を許可するために必要な IAM リソースを作成する

1. Amazon EKS クラスターの OpenID Connect (OIDC) プロバイダーを作成します。これは、サービスアカウントで IAM ロールを使用するために必要です。OIDC プロバイダーの詳細とその作成方法については、「Amazon EKS ユーザーガイド」の「クラスターの IAM OIDC プロバイダーを作成する」を参照してください。

   a. 次のコマンドを使用して、クラスターの IAM OIDC ID プロバイダーを作成します。この例では、クラスター名を my-eks-cluster としています。クラスターの名前が異なる場合は、以降のすべてのコマンドで名前を変更してください。

   ```
   eksctl utils associate-iam-oidc-provider --cluster my-eks-cluster --approve
   ```

   b. 次のコマンドを使用して、OIDC ID プロバイダーが IAM に登録されていることを確認します。

   ```
   aws iam list-open-id-connect-providers --region aws-region
   ```

   出力は次のようになります。OIDC の Amazon リソースネーム (ARN) を書き留めておいてください。次の手順でサービスアカウントの信頼ポリシーを作成する際に必要になります。

   ```
   {
       "OpenIDConnectProviderList": [
           ..
           {
   ```

```
            "Arn": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.aws-
region.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
        }
    ]
}
```

2. Amazon EKS クラスター用のサービスアカウントを作成します。サービスアカウントで、ポッドで実行されるプロセス用のアイデンティティを用意できます。ポッドは、最小かつ最もシンプルな Kubernetes オブジェクトであり、コンテナ化されたアプリケーションをデプロイするために使用できます。次に、リソースへのアクセス許可を取得するためにサービスアカウントが引き受けることができる IAM ロールを作成します。その AWS サービスへのアクセス許可を持つ IAM ロールを引き受けることができるサービスアカウントを使用するように設定された Pod から任意のサービスにアクセスできます。

   a. サービスアカウント用の新しい名前空間を作成します。名前空間を使用して、このチュートリアル用に作成したクラスターリソースを分離することができます。次のコマンドを使用して、新しい名前空間を作成できます。

   ```
   kubectl create namespace my-eks-namespace
   ```

   b. カスタムの名前空間を使用するには、その名前空間を Fargate プロファイルに関連付ける必要があります。次のコードは、この例です。

   ```
   eksctl create fargateprofile \
       --cluster my-eks-cluster \
       --name my-fargate-profile \
       --namespace my-eks-namespace \
       --labels *=*
   ```

   c. 次のコマンドを使用して、Amazon EKS クラスターの名前空間 `my-eks-namespace` に `my-eks-serviceaccount` という名前のサービスアカウントを作成します。

   ```
   cat >my-serviceaccount.yaml <<EOF
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: my-eks-serviceaccount
     namespace: my-eks-namespace
   EOF
   kubectl apply -f my-serviceaccount.yaml
   ```

d. 次のコマンドを実行して、先ほど作成したサービスアカウントを信頼するように IAM ロールに指示する信頼ポリシーファイルを作成します。この信頼関係が確立されていないと、プリンシパルでロールを引き受けることができません。ファイルで以下の編集を行います。

- Principal として、IAM が list-open-id-connect-providers コマンドに返した ARN を入力します。ARN には、アカウント番号とリージョンが含まれています。
- condition ステートメントで、 AWS リージョン と OIDC ID を置き換えます。
- サービスアカウントの名前と名前空間が正しいことを確認します。

次の手順で IAM ロールを作成するときに、この信頼ポリシーファイルをアタッチする必要があります。

```
cat >trust-relationship.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.aws-region.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.aws-region.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:my-eks-
namespace:my-eks-serviceaccount",
                    "oidc.eks.aws-region.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
EOF
```

オプション: StringEquals または StringLike の条件に複数のエントリを追加して、複数のサービスアカウントや名前空間にロールの引き受けを許可することができます。サービスアカウントが別の AWS アカウントで IAM ロールを引き受けることを許可するには、

「Amazon EKS ユーザーガイド」の「クロスアカウントの IAM アクセス許可」を参照してください。

3. Amazon EKS サービスアカウントが引き受ける IAM ロールを my-iam-role という名前で作成します。直前の手順で作成した信頼ポリシーファイルをロールにアタッチします。信頼ポリシーには、IAM ロールが信頼できるサービスアカウントと OIDC プロバイダーが指定されています。

```
aws iam create-role --role-name my-iam-role --assume-role-policy-document file://
trust-relationship.json --description "EKS service account role"
```

4. アクセスポリシーをアタッチして、IAM ロールのアクセス許可を Amazon Keyspaces に割り当てます。

   a. アクセスポリシーをアタッチして、IAM ロールが特定の Amazon Keyspaces リソースに対して実行できるアクションを定義します。このチュートリアルでは、アプリケーションが Amazon Keyspaces テーブルにデータを書き込むためAmazonKeyspacesFullAccess、AWS マネージドポリシー を使用します。ただし、ベストプラクティスとしては、最小特権の原則を実装したカスタムのアクセスポリシーを作成することをお勧めします。詳細については、「the section called “Amazon Keyspaces で IAM が機能する仕組み”」を参照してください。

```
aws iam attach-role-policy --role-name my-iam-role --policy-
arn=arn:aws:iam::aws:policy/AmazonKeyspacesFullAccess
```

   次のステートメントを使用して、ポリシーが IAM ロールに正しくアタッチされたことを確認します。

```
aws iam list-attached-role-policies --role-name my-iam-role
```

   出力は次のようになります。

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonKeyspacesFullAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonKeyspacesFullAccess"
        }
    ]
```

```
}
```

b. サービスアカウントに、引き受け可能な IAM ロールの Amazon リソースネーム (ARN) を注釈として追加します。ロールの ARN にアカウント ID を反映してください。

```
kubectl annotate serviceaccount -n my-eks-namespace my-eks-serviceaccount
 eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/my-iam-role
```

5. IAM ロールとサービスアカウントが正しく設定されていることを確認します。

a. 次のステートメントを使用して、IAM ロールの信頼ポリシーが正しく設定されていることを確認します。

```
aws iam get-role --role-name my-iam-role --query Role.AssumeRolePolicyDocument
```

出力は次のようになります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.aws-region.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.aws-region/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.aws-region.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:my-eks-
namespace:my-eks-serviceaccount"
                }
            }
        }
    ]
}
```

b. Amazon EKS サービスアカウントに IAM ロールの注釈が追加されていることを確認します。

```
kubectl describe serviceaccount my-eks-serviceaccount -n my-eks-namespace
```

出力は次のようになります。

```
Name: my-eks-serviceaccount
Namespace:my-eks-namespace
Labels: <none>
Annotations: eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-iam-role
Image pull secrets: <none>
Mountable secrets: <none>
Tokens: <none>
[...]
```

Amazon EKS サービスアカウントと IAM ロールを作成し、必要な関係とアクセス許可を設定したら、the section called “ステップ 2: アプリケーションを設定する” に進んでください。

ステップ 2: アプリケーションを設定する

このステップでは、SigV4 プラグインを使用して Amazon Keyspaces に接続するアプリケーションを構築します。Github の Amazon Keyspaces サンプルコードリポジトリから、サンプル Java アプリケーションを表示し、ダウンロードできます。または、独自のアプリケーションを使用することもできます。以下の手順に従って、すべての設定手順を終わらせてください。

アプリケーションを設定して必要な依存関係を追加する

1. 次のコマンドを使用して Github リポジトリのクローンを作成し、サンプル Java アプリケーションをダウンロードできます。

   ```
   git clone https://github.com/aws-samples/amazon-keyspaces-examples.git
   ```

2. Github リポジトリをダウンロードしたら、ダウンロードしたファイルを解凍し、`resources` ディレクトリの `application.conf` ファイルに移動します。

   a. アプリケーションの設定

      この手順では、SigV4 認証プラグインを設定します。以下のサンプルをアプリケーションで使用できます。まだ作成していない場合は、IAM アクセスキー (アクセスキー ID とシークレットアクセスキー) AWS を生成し、設定ファイルまたは環境変数として保存する必要が

あります。詳細な手順については、「the section called “ AWS 認証に必要な認証情報”」を参照してください。必要に応じて Amazon Keyspaces の AWS リージョンとサービスエンドポイントを更新します。サービスエンドポイントの詳細については、「the section called “サービスエンドポイント”」を参照してください。トラストストアの場所、トラストストア名、トラストストアパスワードは、実際の値に置き換えてください。

```
datastax-java-driver {
  basic.contact-points = ["cassandra.aws-region.amazonaws.com:9142"]
  basic.load-balancing-policy.local-datacenter = "aws-region"
  advanced.auth-provider {
    class = software.aws.mcs.auth.SigV4AuthProvider
    aws-region = "aws-region"
  }
  advanced.ssl-engine-factory {
    class = DefaultSslEngineFactory
    truststore-path = "truststore_locationtruststore_name.jks"
    truststore-password = "truststore_password;"
  }
}
```

b. STS モジュールの依存関係を追加します。

これにより、サービスアカウントWebIdentityTokenCredentialsProviderが IAM ロールを引き受けられるように、アプリケーションが提供する必要がある AWS 認証情報を返す を使用する機能が追加されます。この作業は、次の例を参考にして行います。

```
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-sts</artifactId>
            <version>1.11.717</version>
        </dependency>
```

c. SigV4 の依存関係を追加します。

このパッケージは、Amazon Keyspaces への認証に必要な SigV4 認証プラグインを実装します。

```
        <dependency>
            <groupId>software.aws.mcs</groupId>
            <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin</
artifactId>
            <version>4.0.3</version>
```

```
        </dependency>
```

3. ログ記録の依存関係を追加します。

   ログがなければ、接続の問題をトラブルシューティングすることができません。このチュートリアルでは、slf4j をログ記録のフレームワークとして使用し、logback.xml を使用してログ出力を保存します。ログ記録のレベルを debug に設定して、接続を確立します。次の例を使用して、依存関係を追加できます。

```
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>2.0.5</version>
        </dependency>
```

   次のコードスニペットを使用して、ログ記録を設定できます。

```
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</
pattern>
        </encoder>
    </appender>

    <root level="debug">
        <appender-ref ref="STDOUT" />
    </rootv
</configuration>
```

> **Note**
>
> debug レベルは、接続エラーを調査するために必要です。アプリケーションから Amazon Keyspaces に正常に接続できたら、ログ記録レベルを適宜 info または warning に変更できます。

ステップ 3: アプリケーションイメージを作成し、Docker ファイルを Amazon ECR リポジトリにアップロードする

このステップでは、サンプルアプリケーションをコンパイルし、Docker イメージを構築して、そのイメージを Amazon ECR リポジトリにプッシュします。

アプリケーションを構築し、Docker イメージを構築して Amazon Elastic Container Registry に送信する

1. ビルドの AWS リージョンを定義する環境変数を設定します。以降の例のリージョンは、実際のリージョンに置き換えてください。

```
export CASSANDRA_HOST=cassandra.aws-region.amazonaws.com:9142
export CASSANDRA_DC=aws-region
```

2. 次のコマンドを使用して、アプリケーションを Apache Maven バージョン 3.6.3 以降でコンパイルします。

```
mvn clean install
```

   すべての依存関係を target ディレクトリに収録した JAR ファイルが作成されます。

3. 次のコマンドを使用して、次の手順で必要になる ECR リポジトリ URI を取得します。リージョンは、実際のリージョンに置き換えてください。

```
aws ecr describe-repositories --region aws-region
```

   出力は次の例のようになります。

```
"repositories": [
 {
 "repositoryArn": "arn:aws:ecr:aws-region:111122223333:repository/my-ecr-
repository",
 "registryId": "111122223333",
 "repositoryName": "my-ecr-repository",
 "repositoryUri": "111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-
repository",
 "createdAt": "2023-11-02T03:46:34+00:00",
 "imageTagMutability": "MUTABLE",
 "imageScanningConfiguration": {
```

```
"scanOnPush": false
},
"encryptionConfiguration": {
"encryptionType": "AES256"
}
},
```

4. アプリケーションのルートディレクトリから、直前の手順で取得したリポジトリ URI を使用して Docker イメージを構築します。必要に応じて Docker ファイルを変更してください。ビルドコマンドで、アカウント ID を置き換え、 AWS リージョン を Amazon ECR リポジトリがあるリージョンに設定しますmy-ecr-repository。

```
docker build -t 111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-
repository:latest .
```

5. 認証トークンを取得して、Docker イメージを Amazon ECR にプッシュします。それには、次のコマンドを使用します。

```
aws ecr get-login-password --region aws-region | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.aws-region.amazonaws.com
```

6. まず、Amazon ECR リポジトリ内の既存のイメージを確認します。以下のコマンドを使用できます。

```
aws ecr describe-images --repository-name my-ecr-repository --region aws-region
```

次に、Docker イメージをリポジトリにプッシュします。以下のコマンドを使用できます。

```
docker push 111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-repository:latest
```

ステップ 4: アプリケーションを Amazon EKS にデプロイし、テーブルにデータを書き込む

チュートリアルのこのステップでは、アプリケーションの Amazon EKS デプロイを設定し、アプリケーションが正常に実行されていて、Amazon Keyspaces に接続できることを確認します。

アプリケーションを Amazon EKS にデプロイするには、関連するすべての設定を deployment.yaml というファイルに記述する必要があります。このファイルを Amazon EKS が使

用して、アプリケーションをデプロイします。ファイル内のメタデータには、次の情報を指定しておく必要があります。

- アプリケーション名: アプリケーションの名前。このチュートリアルでは、`my-keyspaces-app` を使用します。
- Kubernetes 名前空間: Amazon EKS クラスターの名前空間。このチュートリアルでは、`my-eks-namespace` を使用します。
- Amazon EKS サービスアカウント名: Amazon EKS サービスアカウントの名前。このチュートリアルでは、`my-eks-serviceaccount` を使用します。
- イメージ名: アプリケーションイメージの名前。このチュートリアルでは、`my-keyspaces-app` を使用します。
- イメージ URI: Amazon ECR の Docker イメージの URI。
- AWS アカウント ID AWS アカウント ID。
- IAM ロールの ARN: サービスアカウントが引き受けるために作成された IAM ロールの ARN。このチュートリアルでは、`my-iam-role` を使用します。
- AWS リージョン Amazon EKS クラスターを AWS リージョン 作成した Amazon EKS クラスターの 。

このステップでは、Amazon Keyspaces に接続し、テーブルにデータを書き込むアプリケーションをデプロイし、実行します。

1. `deployment.yaml` ファイルを設定します。以下の値を置き換える必要があります。

   - `name`
   - `namespace`
   - `serviceAccountName`
   - `image`
   - `AWS_ROLE_ARN value`
   - AWS リージョン の `CASSANDRA_HOST`
   - `AWS_REGION`

   次のファイルをサンプルとして使用できます。

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: my-keyspaces-app
  namespace: my-eks-namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-keyspaces-app
  template:
    metadata:
      labels:
        app: my-keyspaces-app
    spec:
      serviceAccountName: my-eks-serviceaccount
      containers:
      - name: my-keyspaces-app
        image: 111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-
repository:latest
        ports:
        - containerPort: 8080
        env:
        - name: CASSANDRA_HOST
          value: "cassandra.aws-region.amazonaws.com:9142"
        - name: CASSANDRA_DC
          value: "aws-region"
        - name: AWS_WEB_IDENTITY_TOKEN_FILE
          value: /var/run/secrets/eks.amazonaws.com/serviceaccount/token
        - name: AWS_ROLE_ARN
          value: "arn:aws:iam::111122223333:role/my-iam-role"
        - name: AWS_REGION
          value: "aws-region"
```

2. deployment.yaml をデプロイします。

```
kubectl apply -f deployment.yaml
```

出力は次のようになります。

```
deployment.apps/my-keyspaces-app created
```

3. Amazon EKS クラスターの名前空間におけるポッドのステータスを確認します。

```
kubectl get pods -n my-eks-namespace
```

この例のような出力が得られます。

```
NAME                    READY STATUS RESTARTS AGE
my-keyspaces-app-123abcde4f-g5hij 1/1 Running 0 75s
```

次のコマンドを使用して、詳細情報を確認できます。

```
kubectl describe pod my-keyspaces-app-123abcde4f-g5hij -n my-eks-namespace
```

```
Name:                 my-keyspaces-app-123abcde4f-g5hij
Namespace:            my-eks-namespace
Priority:             2000001000
Priority Class Name:  system-node-critical
Service Account:      my-eks-serviceaccount
Node:                 fargate-ip-192-168-102-209.ec2.internal/192.168.102.209
Start Time:           Thu, 23 Nov 2023 12:15:43 +0000
Labels:               app=my-keyspaces-app
                      eks.amazonaws.com/fargate-profile=my-fargate-profile
                      pod-template-hash=6c56fccc56
Annotations:          CapacityProvisioned: 0.25vCPU 0.5GB
                      Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
Status:               Running
IP:                   192.168.102.209
IPs:
  IP:           192.168.102.209
Controlled By:  ReplicaSet/my-keyspaces-app-6c56fccc56
Containers:
  my-keyspaces-app:
    Container ID:
 containerd://41ff7811d33ae4bc398755800abcdc132335d51d74f218ba81da0700a6f8c67b
    Image:          111122223333.dkr.ecr.aws-region.amazonaws.com/
my_eks_repository:latest
    Image ID:       111122223333.dkr.ecr.aws-region.amazonaws.com/
my_eks_repository@sha256:fd3c6430fc5251661efce99741c72c1b4b03061474940200d0524b84a951439c
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Thu, 23 Nov 2023 12:15:19 +0000
```

```
      Finished:     Thu, 23 Nov 2023 12:16:17 +0000
    Ready:          True
    Restart Count:  1
    Environment:
      CASSANDRA_HOST:               cassandra.aws-region.amazonaws.com:9142
      CASSANDRA_DC:                 aws-region
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
      AWS_ROLE_ARN:                 arn:aws:iam::111122223333:role/my-iam-role
      AWS_REGION:                   aws-region
      AWS_STS_REGIONAL_ENDPOINTS:   regional
    Mounts:
      /var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-fssbf (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  aws-iam-token:
    Type:                     Projected (a volume that contains injected data from
 multiple sources)
    TokenExpirationSeconds:  86400
  kube-api-access-fssbf:
    Type:                     Projected (a volume that contains injected data from
 multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for
 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for
 300s
Events:
  Type     Reason           Age                From               Message
  ----     ------           ----               ----               -------
  Warning  LoggingDisabled  2m13s              fargate-scheduler  Disabled logging
 because aws-logging configmap was not found. configmap "aws-logging" not found
```

```
  Normal   Scheduled        89s                fargate-scheduler  Successfully
 assigned my-eks-namespace/my-keyspaces-app-6c56fccc56-mgs2m to fargate-
ip-192-168-102-209.ec2.internal
  Normal   Pulled           75s                kubelet
 Successfully pulled image "111122223333.dkr.ecr.aws-region.amazonaws.com/
my_eks_repository:latest" in 13.027s (13.027s including waiting)
  Normal   Pulling          54s (x2 over 88s)  kubelet            Pulling image
 "111122223333.dkr.ecr.aws-region.amazonaws.com/my_eks_repository:latest"
  Normal   Created          54s (x2 over 75s)  kubelet            Created container
 my-keyspaces-app
  Normal   Pulled           54s                kubelet
 Successfully pulled image "111122223333.dkr.ecr.aws-region.amazonaws.com/
my_eks_repository:latest" in 222ms (222ms including waiting)
  Normal   Started          53s (x2 over 75s)  kubelet            Started container
 my-keyspaces-app
```

4. ポッドのログをチェックして、アプリケーションが実行中であり、Amazon Keyspaces のテーブルに接続できることを確認します。それには、次のコマンドを使用します。実際のデプロイの名前に置き換えてください。

```
kubectl logs -f my-keyspaces-app-123abcde4f-g5hij -n my-eks-namespace
```

以下の例のように、Amazon Keyspaces への接続を確認するアプリケーションログエントリを確認できるはずです。

```
2:47:20.553 [s0-admin-0] DEBUG c.d.o.d.i.c.metadata.MetadataManager
 - [s0] Adding initial contact points [Node(endPoint=cassandra.aws-
region.amazonaws.com/1.222.333.44:9142, hostId=null, hashCode=e750d92)]
22:47:20.562 [s0-admin-1] DEBUG c.d.o.d.i.c.c.ControlConnection - [s0] Initializing
 with event types [SCHEMA_CHANGE, STATUS_CHANGE, TOPOLOGY_CHANGE]
22:47:20.564 [s0-admin-1] DEBUG c.d.o.d.i.core.context.EventBus - [s0] Registering
 com.datastax.oss.driver.internal.core.metadata.LoadBalancingPolicyWrapper$$Lambda
$812/0x0000000801105e88@769afb95 for class
 com.datastax.oss.driver.internal.core.metadata.NodeStateEvent
22:47:20.566 [s0-admin-1] DEBUG c.d.o.d.i.c.c.ControlConnection -
 [s0] Trying to establish a connection to Node(endPoint=cassandra.us-
east-1.amazonaws.com/1.222.333.44:9142, hostId=null, hashCode=e750d92)
```

5. Amazon Keyspaces テーブルで次の CQL クエリを実行して、テーブルに 1 行のデータが書き込まれていることを確認します。

```
SELECT * from aws.user;
```

以下の出力が表示されます。

```
fname    | lname | username | last_update_date
----------+-------+----------+---------------------------
random    | k     | test     | 2023-12-07 13:58:31.57+0000
```

ステップ 5: (オプション) クリーンアップする

以下の手順に従って、このチュートリアルで作成したリソースをすべて削除してください。

このチュートリアルで作成したリソースを削除する

1. デプロイを削除します。それには、次のコマンドを使用します。

```
kubectl delete deployment my-keyspaces-app -n my-eks-namespace
```

2. Amazon EKS クラスターと、その中に含まれるすべてのポッドを削除します。サービスアカウントや OIDC ID プロバイダーなどの関連リソースも削除されます。それには、次のコマンドを使用します。

```
eksctl delete cluster --name my-eks-cluster --region aws-region
```

3. Amazon EKS サービスアカウントで使用される、Amazon Keyspaces へのアクセス許可を持つ IAM ロールを削除します。まず、ロールにアタッチされているマネージドポリシーを削除する必要があります。

```
aws iam detach-role-policy --role-name my-iam-role --policy-arn
 arn:aws:iam::aws:policy/AmazonKeyspacesFullAccess
```

その後、次のコマンドを使用してロールを削除できます。

```
aws iam delete-role --role-name my-iam-role
```

詳細については、「IAM ユーザーガイド」の「IAM ロールの削除 (AWS CLI)」を参照してください。

4. Amazon ECR リポジトリを、収録しているすべてのイメージごと削除します。それには、次のコマンドを使用します。

```
aws ecr delete-repository \
      --repository-name my-ecr-repository \
      --force \
      --region aws-region
```

イメージを含むリポジトリを削除するには、force フラグが必要です。イメージを先に削除するには、次のコマンドを使用します。

```
aws ecr batch-delete-image \
      --repository-name my-ecr-repository \
      --image-ids imageTag=latest \
      --region aws-region
```

詳細については、「Amazon Elastic Container Registry ユーザーガイド」の「イメージを削除する」を参照してください。

5. Amazon Keyspaces のキースペースとテーブルを削除します。キースペースを削除すると、そのキースペース内のすべてのテーブルが自動的に削除されます。これは、次のいずれかの方法で行うことができます。

AWS CLI

```
aws keyspaces delete-keyspace --keyspace-name 'aws'
```

キースペースが削除されたことを確認するには、次のコマンドを使用します。

```
aws keyspaces list-keyspaces
```

テーブルを先に削除するには、次のコマンドを使用します。

```
aws keyspaces delete-table --keyspace-name 'aws' --table-name 'user'
```

テーブルが削除されたことを確認するには、次のコマンドを使用します。

```
aws keyspaces list-tables --keyspace-name 'aws'
```

詳細については、「AWS CLI コマンドリファレンス」の「delete-keyspace」と「delete-table」を参照してください。

cqlsh

```
DROP KEYSPACE IF EXISTS "aws";
```

キースペースが削除されたことを確認するには、次のステートメントを使用します。

```
SELECT * FROM system_schema.keyspaces ;
```

このステートメントの出力に、キースペースが表示されなくなっているはずです。キースペースが削除されるまで、時間がかかる場合があります。詳細については、「the section called “DROP KEYSPACE”」を参照してください。

テーブルを先に削除するには、次のコマンドを使用します。

```
DROP TABLE "aws.user"
```

テーブルが削除されたことを確認するには、次のコマンドを使用します。

```
SELECT * FROM system_schema.tables WHERE keyspace_name = "aws";
```

このステートメントの出力に、テーブルが表示されなくなっているはずです。テーブルが削除されるまで、時間がかかる場合があります。詳細については、「the section called “DROP TABLE”」を参照してください。

チュートリアル: を使用して Amazon Keyspaces テーブルを Amazon S3 にエクスポートする AWS Glue

このチュートリアルでは、 を使用して Amazon Keyspaces テーブルを Amazon S3 バケットにエクスポートする方法を示します AWS Glue。このチュートリアルでは、Amazon Keyspaces Github リポジトリで利用可能なシェルスクリプトを使用して、多くの手動ステップが自動化されています。このプロセスを使用すると、Spark クラスターを設定することなく、Amazon Keyspaces データを Amazon S3 にエクスポートできます。

トピック

- Amazon Keyspaces から Amazon S3 にデータをエクスポートするための前提条件
- ステップ 1: Amazon S3 バケットを作成し、必要なツールをダウンロードして、環境を設定する
- ステップ 2: Amazon Keyspaces テーブルをエクスポートする AWS Glue ジョブを設定する
- ステップ 3: AWS Glue ジョブを実行して、Amazon Keyspaces テーブルを から Amazon S3 バケットにエクスポートする AWS CLI
- ステップ 4: (オプション) エクスポートジョブをスケジュールするトリガーを作成する
- ステップ 5: (オプション) クリーンアップする

Amazon Keyspaces から Amazon S3 にデータをエクスポートするための前提条件

チュートリアルを開始する前に、以下の前提条件を確認し、Amazon Keyspaces リソースを作成します。

1. このチュートリアルを開始する前に、 AWS 「」のセットアップ手順に従ってくださいAmazon Keyspaces (Apache Cassandra 向け) へのアクセス。これらのステップには、Amazon Keyspaces にアクセスできる AWS Identity and Access Management (IAM) プリンシパルへのサインアップ AWS と作成が含まれます。
2. このチュートリアルのスクリプトでは、既知の場所に AWS リージョン 保存された認証情報とデフォルトを使用します。詳細については、「the section called “ アクセスキーを管理する”」を参照してください。

   次の例は、デフォルトユーザーの環境変数として必要な値を保存する方法を示しています。

   ```
   $ export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
   $ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   $ export AWS_DEFAULT_REGION=aws-region
   ```

3. このチュートリアルのスクリプトを実行するには、次のソフトウェアとツールがマシンにインストールされている必要があります。

   - Java
   - Apache Maven
   - Git
   - AWS CLI

このチュートリアルは、 AWS CLI 2、Java 17.0.13、および Apache Maven 3.8.7 でテストされています。

4. このチュートリアルの後半でエクスポートするには、サンプルデータを含む Amazon Keyspaces テーブルが必要です。チュートリアルの手順に従って、独自の Amazon Keyspaces テーブルを使用するか、サンプルテーブルを作成できます入門。

   a. をインストールするにはcqlsh-expansion、「」の手順に従ってくださいthe section called “cqlsh-expansion の使用”。

   b. Murmur3Partitioner パーティショナーがアカウントのデフォルトパーティショナーであることを確認してください。このパーティショナーは、Apache Spark Cassandra Connector および と互換性があります AWS Glue。パーティショナーの詳細については、「」を参照してくださいthe section called “パーティショナーの操作”。

      アカウントのパーティショナーを変更するには、次のステートメントを使用できます。

      ```
      SELECT partitioner FROM system.local;

      UPDATE system.local set
       partitioner='org.apache.cassandra.dht.Murmur3Partitioner' where key='local';
      ```

   c. Amazon Keyspaces キースペースを作成するには、「」の手順に従いますthe section called “キースペースの作成”。

   d. Amazon Keyspaces テーブルを作成するには、「」のステップに従いますthe section called “テーブルの作成”。

   e. Amazon S3 にエクスポートするサンプルデータをテーブルにロードするには、「」の手順に従いますthe section called “作成”。

前提条件のステップが完了したら、the section called “ステップ 1: Amazon S3 バケットの作成、ツールのダウンロード、環境の設定” に進みます。

ステップ 1: Amazon S3 バケットを作成し、必要なツールをダウンロードして、環境を設定する

このステップでは、外部ツールをダウンロードし、 AWS AWS Glue ジョブを使用して Amazon Keyspaces テーブルの自動データエクスポートソリューションに必要なリソースを作成し、Amazon

S3 バケットに設定します。これらのタスクを効率的に実行するには、Github でsetup-connector.sh使用可能な名前でシェルスクリプトを実行します。

スクリプトは次のステップsetup-connector.shを自動化します。

1. を使用して Amazon S3 バケットを作成します AWS CloudFormation。このバケットには、ダウンロードした jar ファイルと設定ファイル、およびエクスポートされたテーブルデータが保存されます。
2. を使用して IAM ロールを作成します AWS CloudFormation。 AWS Glue ジョブはこのロールを使用して Amazon Keyspaces と Amazon S3 にアクセスします。
3. Apache Spark Cassandra Connector をダウンロードし、Amazon S3 バケットにアップロードします。
4. SigV4 認証プラグインをダウンロードし、Amazon S3 バケットにアップロードします。
5. Apache Spark Extensions をダウンロードし、Amazon S3 バケットにアップロードします。
6. Github から Keyspaces 再試行ポリシーをダウンロードし、Maven を使用してコードをコンパイルし、出力を Amazon S3 バケットにアップロードします。
7. **keyspaces-application.conf** ファイルを Amazon S3 バケットにアップロードします。

setup-connector.sh シェルスクリプトを使用して、セットアップと設定のステップを自動化します。

1. Github の aws-glue リポジトリからローカルマシンにファイルをコピーします。このディレクトリには、シェルスクリプトとその他の必要なファイルが含まれています。
2. シェルスクリプト を実行しますsetup-connector.sh。次の 3 つのオプションパラメータを指定できます。

    1. SETUP_STACKNAME – AWS リソースの作成に使用される AWS CloudFormation スタックの名前です。
    2. S3_BUCKET_NAME – これは Amazon S3 バケットの名前です。
    3. GLUE_SERVICE_ROLE_NAME – これは、Amazon Keyspaces と Amazon S3 に接続するジョブを実行するために が AWS Glue 使用する IAM サービスロールの名前です。

    次のコマンドを使用してシェルスクリプトを実行し、3 つのパラメータを次の名前で指定できます。

```
./setup-connector.sh cfn-setup s3-keyspaces iam-export-role
```

バケットが作成されたことを確認するには、次の AWS CLI コマンドを使用します。

```
aws s3 ls s3://s3-keyspaces
```

コマンドの出力は次のようになります。

```
                           PRE conf/
                           PRE jars/
```

IAM ロールが作成されたことを確認し、詳細を確認するには、次の AWS CLI ステートメントを使用します。

```
aws iam get-role --role-name "iam-export-role"
```

```
{
    "Role": {
        "Path": "/",
        "RoleName": "iam-export-role",
        "RoleId": "AKIAIOSFODNN7EXAMPLE",
        "Arn": "arn:aws:iam::1111-2222-3333:role/iam-export-role",
        "CreateDate": "2025-01-28T16:09:03+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "glue.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Description": "AWS Glue service role to import and export data from Amazon
 Keyspaces",
        "MaxSessionDuration": 3600,
        "RoleLastUsed": {
```

```
            "LastUsedDate": "2025-01-29T12:03:54+00:00",
            "Region": "us-east-1"
        }
    }
}
```

AWS CloudFormation スタックプロセスが失敗した場合、失敗したスタックに関する詳細なエラー情報を AWS CloudFormation コンソールで確認できます。

すべてのスクリプトとツールを含む Amazon S3 バケットが作成され、IAM ロールが設定されたら、「」に進みますthe section called “ステップ 2: AWS Glue ジョブを設定する”。

ステップ 2: Amazon Keyspaces テーブルをエクスポートする AWS Glue ジョブを設定する

チュートリアルの 2 番目のステップでは、Github でsetup-export.sh利用可能なスクリプトを使用して、SigV4 プラグインを使用して Amazon Keyspaces に接続する AWS Glue ジョブを作成および設定し、前のステップで作成した Amazon S3 バケットに指定されたテーブルをエクスポートします。スクリプトを使用すると、Apache Spark クラスターを設定せずに Amazon Keyspaces からデータをエクスポートできます。

Amazon Keyspaces テーブルを Amazon S3 バケットにエクスポートする AWS Glue ジョブを作成します。

- このステップでは、 export-to-s3/ ディレクトリにあるsetup-export.shシェルスクリプトを実行して、 AWS Glue エクスポートジョブの作成と設定 AWS CloudFormation に使用します。スクリプトは次のパラメータを使用します。

  ```
  PARENT_STACK_NAME, EXPORT_STACK_NAME, KEYSPACE_NAME, TABLE_NAME, S3_URI, FORMAT
  ```

  - PARENT_STACK_NAME – 前のステップで作成した AWS CloudFormation スタックの名前。
  - EXPORT_STACK_NAME – AWS Glue エクスポートジョブを作成する AWS CloudFormation スタックの名前。
  - KEYSPACE_NAME および TABLE_NAME – エクスポートするキースペースとテーブルの完全修飾名。このチュートリアルでは を使用しますがcatalog.book_awards、これを独自の完全修飾テーブル名に置き換えることができます。

- S3URI – Amazon S3 バケットのオプション URI。デフォルトは、親スタックの Amazon S3 バケットです。
- FORMAT – オプションのデータ形式。デフォルト値は parquet です。このチュートリアルでは、データのロードと変換を容易にするために、デフォルトを使用します。

例えば、次の コマンドを使用できます。

```
setup-export.sh cfn-setup cfn-glue catalog book_awards
```

ジョブが作成されたことを確認するには、次のステートメントを使用できます。

```
aws glue list-jobs
```

ステートメントの出力は次のようになります。

```
{
    "JobNames": [
        "AmazonKeyspacesExportToS3-cfn-setup-cfn-glue"
    ]
}
```

ジョブの詳細を表示するには、次のコマンドを使用します。

```
aws glue get-job --job-name AmazonKeyspacesExportToS3-cfn-setup-cfn-glue
```

コマンドの出力には、ジョブのすべての詳細が表示されます。これには、ジョブの実行時に上書きできるデフォルトの引数が含まれます。

```
{
    "Job": {
        "Name": "AmazonKeyspacesExportToS3-cfn-setup-cfn-glue",
        "JobMode": "SCRIPT",
        "JobRunQueuingEnabled": false,
        "Description": "export to s3",
        "Role": "iam-export-role",
        "CreatedOn": "2025-01-30T15:53:30.765000+00:00",
        "LastModifiedOn": "2025-01-30T15:53:30.765000+00:00",
        "ExecutionProperty": {
```

```
            "MaxConcurrentRuns": 1
        },
        "Command": {
            "Name": "glueetl",
            "ScriptLocation": "s3://s3-keyspaces/scripts/cfn-setup-cfn-glue-
export.scala",
            "PythonVersion": "3"
        },
        "DefaultArguments": {
            "--write-shuffle-spills-to-s3": "true",
            "--S3_URI": "s3://s3-keyspaces",
            "--TempDir": "s3://s3-keyspaces/shuffle-space/export-sample/",
            "--extra-jars": "s3://s3-keyspaces/jars/spark-cassandra-
connector-assembly_2.12-3.1.0.jar,s3://s3-keyspaces/jars/aws-sigv4-auth-
cassandra-java-driver-plugin-4.0.9-shaded.jar,s3://s3-keyspaces/jars/spark-
extension_2.12-2.8.0-3.4.jar,s3://s3-keyspaces/jars/amazon-keyspaces-helpers-1.0-
SNAPSHOT.jar",
            "--class": "GlueApp",
            "--user-jars-first": "true",
            "--enable-metrics": "true",
            "--enable-spark-ui": "true",
            "--KEYSPACE_NAME": "catalog",
            "--spark-event-logs-path": "s3://s3-keyspaces/spark-logs/",
            "--enable-continuous-cloudwatch-log": "true",
            "--write-shuffle-files-to-s3": "true",
            "--FORMAT": "parquet",
            "--TABLE_NAME": "book_awards",
            "--job-language": "scala",
            "--extra-files": "s3://s3-keyspaces/conf/keyspaces-application.conf",
            "--DRIVER_CONF": "keyspaces-application.conf"
        },
        "MaxRetries": 0,
        "AllocatedCapacity": 4,
        "Timeout": 2880,
        "MaxCapacity": 4.0,
        "WorkerType": "G.2X",
        "NumberOfWorkers": 2,
        "GlueVersion": "3.0"
    }
}
```

AWS CloudFormation スタックプロセスが失敗した場合、失敗したスタックのエラーを AWS CloudFormation コンソールで確認できます。左側のメニューで ETL ジョブを選択すると、 AWS Glue コンソールでエクスポートジョブの詳細を確認できます。

AWS Glue エクスポートジョブの詳細を確認したら、 the section called “ステップ 3: からエクスポート AWS Glue ジョブを実行する AWS CLI” に進み、ジョブを実行して Amazon Keyspaces テーブルからデータをエクスポートします。

ステップ 3: AWS Glue ジョブを実行して、Amazon Keyspaces テーブルをから Amazon S3 バケットにエクスポートする AWS CLI

このステップでは、 を使用して前のステップで作成した AWS Glue ジョブ AWS CLI を実行し、Amazon Keyspaces テーブルを Amazon S3 のバケットにエクスポートします。

からエクスポートジョブを実行する AWS CLI

1. 次の例では、 AWS CLI コマンドは前のステップで作成したジョブを実行します。

```
aws glue start-job-run --job-name AmazonKeyspacesExportToS3-cfn-setup-cfn-glue
```

   - AWS CLI コマンドのデフォルトの引数を含む任意の AWS Glue ジョブパラメータを上書きできます。キースペースやテーブル名など、ジョブのデフォルトの引数を上書きするには、引数として渡すことができます。引数の完全なリストについては、 AWS Glue 「 コマンドラインリファレンス」のstart-job-run」を参照してください。

     次のコマンドは AWS Glue エクスポートジョブを実行しますが、 AWS Glue ワーカーの数、ワーカータイプ、テーブル名を上書きします。

```
aws glue start-job-run --job-name AmazonKeyspacesExportToS3-cfn-setup-cfn-glue
 \
                    --number-of-workers 8 --worker-type G.2X \
                    --arguments '{"--TABLE_NAME":"my_table"}'
```

2. テーブルが Amazon S3 バケットにエクスポートされていることを確認します。テーブルのサイズによっては、時間がかかる場合があります。エクスポートジョブが完了すると、 コマンド例を使用してバケットに次のフォルダが表示されます。

```
aws s3 ls s3://s3-keyspaces
```

出力には、バケット内の次の構造が表示されます。

```
                           PRE conf/
                           PRE export/
                           PRE jars/
                           PRE scripts/
                           PRE spark-logs/
```

ファイルは の下の次のフォルダ構造に配置されexport、data/time 値には独自の値が表示されます。

```
\------- export
                \----- keyspace_name
                    \----- table_name
                       \----- snapshot
                           \----- year=2025
                               \----- month=01
                                  \----- day=02
                                      \----- hour=09
                                          \----- minute=22
                                              \--- YOUR DATA HERE
```

手動で実行した AWS Glue ジョブをスケジュールするには、「」に進みますthe section called “ステップ 4: (オプション) エクスポートジョブをスケジュールする”。

ステップ 4: (オプション) エクスポートジョブをスケジュールするトリガーを作成する

前のステップで作成したエクスポートジョブを定期的に実行するには、スケジュールされたトリガーを作成できます。詳細については、「 AWS Glue デベロッパーガイド」のAWS Glue 「 トリガー」を参照してください。

AWS Glue ジョブをスケジュールする

1. 次の AWS CLI コマンドは、月曜日の 12:00 UTC に週に 1 AmazonKeyspacesExportToS3-cfn-setup-cfn-glue回ジョブKeyspacesExportWeeklyTriggerを実行する名前 AWS Glue の単純なトリガーの例です。

```
aws glue create-trigger \
  --name KeyspacesExportWeeklyTrigger \
  --type SCHEDULED \
  --schedule "cron(0 12 ? * MON *)" \
  --start-on-creation \
  --actions '[{
     "JobName": "AmazonKeyspacesExportToS3-cfn-setup-cfn-glue"
  }]'
```

- スケジュールされたジョブのデフォルト設定を上書きするには、引数として渡すことができます。この例では、キースペース名、テーブル名、ワーカー数、ワーカータイプを引数として渡して上書きします。次のコマンドは、その一例です。

```
aws glue create-trigger \
  --name KeyspacesExportWeeklyTrigger \
  --type SCHEDULED \
  --schedule "cron(0 12 ? * MON *)" \
  --start-on-creation \
  --actions '[{
     "JobName": "AmazonKeyspacesExportToS3-cfn-setup-cfn-glue",
     "Arguments": {
       "--number-of-workers": "8",
       "--worker-type": "G.2X"},
       "--table_name": "my_table",
       "--keyspace_name": "my_keyspace"
  }]'
```

2. トリガーが作成されたことを確認するには、次のコマンドを使用します。

```
aws glue list-triggers
```

コマンドの出力は次のようになります。

```
{
    "TriggerNames": [
        "KeyspacesExportWeeklyTrigger"
    ]
}
```

このチュートリアルで作成した AWS リソースをクリーンアップするには、「」に進みますthe section called “ステップ 5: (オプション) クリーンアップする”。

ステップ 5: (オプション) クリーンアップする

このチュートリアルで作成したすべての AWS リソースを削除するには、次の手順に従います。

このチュートリアルで作成したリソースを削除する

1. このチュートリアルで作成した 2 番目の AWS CloudFormation スタックを削除します。これにより、このチュートリアルで作成した AWS Glue ジョブとトリガーが削除されます。これを行うには、次の コマンドを使用できます。

```
aws cloudformation delete-stack --stack-name cfn-glue
```

2. Amazon S3 バケットとそのバケットに保存されているすべてのデータを削除します。それには、次のコマンドを使用します。

```
aws s3 rm s3://s3-keyspaces --recursive
```

3. このチュートリアルで作成した最初のスタックを削除します。これにより、このチュートリアルで作成された IAM ロールと関連するアクセス許可が削除されます。例えば、次の コマンドを使用できます。

```
aws cloudformation delete-stack --stack-name cfn-setup
```

4. Amazon Keyspaces のキースペースとテーブルを削除します。キースペースを削除すると、そのキースペース内のすべてのテーブルが自動的に削除されます。これは、次のいずれかの方法で行うことができます。

   AWS CLI

   ```
   aws keyspaces delete-keyspace --keyspace-name 'aws'
   ```

   キースペースが削除されたことを確認するには、次のコマンドを使用します。

   ```
   aws keyspaces list-keyspaces
   ```

   テーブルを先に削除するには、次のコマンドを使用します。

```
aws keyspaces delete-table --keyspace-name 'aws' --table-name 'user'
```

テーブルが削除されたことを確認するには、次のコマンドを使用します。

```
aws keyspaces list-tables --keyspace-name 'aws'
```

詳細については、「AWS CLI コマンドリファレンス」の「delete-keyspace」と「delete-table」を参照してください。

cqlsh

```
DROP KEYSPACE IF EXISTS "aws";
```

キースペースが削除されたことを確認するには、次のステートメントを使用します。

```
SELECT * FROM system_schema.keyspaces ;
```

このステートメントの出力に、キースペースが表示されなくなっているはずです。キースペースが削除されるまで、時間がかかる場合があります。詳細については、「the section called “DROP KEYSPACE”」を参照してください。

テーブルを先に削除するには、次のコマンドを使用します。

```
DROP TABLE "aws.user"
```

テーブルが削除されたことを確認するには、次のコマンドを使用します。

```
SELECT * FROM system_schema.tables WHERE keyspace_name = "aws";
```

このステートメントの出力に、テーブルが表示されなくなっているはずです。テーブルが削除されるまで、時間がかかる場合があります。詳細については、「the section called “DROP TABLE”」を参照してください。

Amazon Keyspaces (Apache Cassandra 向け) でのサーバーレスリソースの管理

Amazon Keyspaces (Apache Cassandra 向け) はサーバーレスです。クラスター内のノードを介してワークロードのストレージリソースとコンピューティングリソースのデプロイ、管理、保守を行う代わりに、Amazon Keyspaces では、ストレージリソースと読み取り/書き込みスループットリソースをテーブルに直接割り当てます。

Amazon Keyspaces では、テーブルに保存されているデータに基づいてストレージが自動的にプロビジョニングされます。データの書き込み、更新、削除に従い、ストレージがスケールアップおよびスケールダウンされ、使用したストレージの分だけ料金を支払います。可用性を高めるため、データは複数のアベイラビリティーゾーンにレプリケートされます。Amazon Keyspaces では、テーブルサイズの継続的監視によりストレージ料金が決定されます。Amazon Keyspaces における課金対象のデータサイズの詳しい計算方法については、「the section called “行サイズの推定”」を参照してください。

この章では、Amazon Keyspaces でのリソース管理の主な側面を取り上げます。

- 行サイズを推定する – Amazon Keyspaces でエンコード後の行サイズを推定するには、パーティションキーのメタデータ、クラスタリング列のメタデータ、列の識別子、データ型、行のメタデータなどの要素を考慮します。エンコード後の行サイズは、請求、クォータ管理、プロビジョニングするスループットキャパシティのプランニングに使用されます。
- キャパシティ消費量を推定する – 範囲クエリ、限定クエリ、テーブルスキャン、軽量トランザクション、静的列、マルチリージョンテーブルなど、一般的なシナリオにおける読み取りと書き込みのキャパシティ消費量を推定する方法の例を紹介します。Amazon CloudWatch を使用して、実際のキャパシティ使用率をモニタリングできます。CloudWatch によるモニタリングの詳細については、「the section called “CloudWatch によるモニタリング”」を参照してください。
- 読み取り/書き込みキャパシティモードを設定する – テーブルへの読み取りと書き込みを処理するためのキャパシティモードを次の 2 つから選択できます。
  - オンデマンドモード (デフォルト) – 読み取りおよび書き込みのスループットに対し、リクエストごとに支払います。Amazon Keyspaces は、これまでに到達実績のあるトラフィックレベルまでは、キャパシティを瞬時にスケールアップできます。
  - プロビジョンドモード – 読み取りおよび書き込みに必要なキャパシティユニット数を事前に指定しておきます。このモードでは、予測可能なスループットパフォーマンスを維持できます。

- 自動スケーリングでスループットキャパシティを管理する – プロビジョンドモードのテーブルの場合、自動スケーリングを有効にして、実際のアプリケーショントラフィックに基づいてスループットキャパシティを自動調整できます。Amazon Keyspaces はターゲット追跡を使用して、プロビジョニングされたキャパシティを増減し、指定された目標使用率を維持します。
- バーストキャパシティを効果的に使用する – Amazon Keyspaces は、トラフィックの急増に対応するために未使用のスループットの一部を予約して、バーストキャパシティを確保します。このおかげで、プロビジョニング済みのスループットを超える一時的なアクティビティの急増にも柔軟に対応できます。

キャパシティエラーのトラブルシューティングについては、「the section called “サーバーレス容量エラー”」を参照してください。

トピック

- Amazon Keyspaces で行のサイズを推定する
- Amazon Keyspaces で読み取り/書き込みスループットのキャパシティ消費量を推定する
- Amazon Keyspaces で読み取り/書き込みのキャパシティモードを設定する
- Amazon Keyspaces 自動スケーリングでスループットキャパシティを自動的に管理する
- Amazon Keyspaces でバーストキャパシティを効果的に使用する

Amazon Keyspaces で行のサイズを推定する

Amazon Keyspaces は、1 桁ミリ秒の読み取りおよび書き込みパフォーマンスを提供し、複数の AWS アベイラビリティーゾーンにデータを永続的に保存するフルマネージドストレージを提供します。Amazon Keyspaces では、効率的なデータアクセスと高可用性をサポートするために、すべての行とプライマリキー列にメタデータをアタッチします。

このトピックでは、Amazon Keyspaces のエンコードされた行サイズを推定する方法について詳しく説明します。エンコードされた行サイズは、請求額とクォータの使用量を計算するときに使用されます。テーブルのプロビジョンドスループットキャパシティ要件を推定するときに、エンコードされた行サイズを使用することもできます。

Amazon Keyspaces 内のエンコードされた行サイズを計算するには、次のガイドラインを使用します。

トピック

- エンコードされた列のサイズを推定する

- データ型に基づいてデータ値のエンコードされたサイズを推定する
- Amazon Keyspaces 機能が行サイズに与える影響を考慮する
- 行のエンコードされたサイズを計算する適切な式を選択する
- 行サイズ計算の例

エンコードされた列のサイズを推定する

このセクションでは、Amazon Keyspaces でエンコードされた列のサイズを推定する方法を示します。

- 通常の列 – プライマリキー、クラスタリング列、または列ではない通常のSTATIC列の場合、データ型に基づいてセルデータの raw サイズを使用し、必要なメタデータを追加します。Amazon Keyspaces がデータ型値とメタデータを保存する方法におけるデータ型といくつかの主な違いを次のセクションに示します。
- パーティションキー列 – パーティションキーには最大 2048 バイトのデータを含めることができます。パーティションキーの各キー列には、最大 3 バイトのメタデータが必要です。行のサイズを計算するときには、各パーティションキー列で上限である 3 バイトのメタデータが使用されていることを想定しておくべきです。
- クラスタリング列 – クラスタリング列には最大 850 バイトのデータを保存できます。データ値のサイズに加えて、各クラスタリング列のメタデータにはデータ値サイズの最大 20% が必要です。行のサイズを計算するときには、5　バイトのクラスタリング列データ値ごとに 1 バイトのメタデータを追加する必要があります。

> **Note**
>
> 効率的なクエリと組み込みインデックス作成をサポートするために、Amazon Keyspaces は各パーティションキーとクラスタリングキー列のデータ値を 2 回保存します。

- 列名 – 各列名に必要なスペースは、列識別子を使用して保存され、列に保存されている各データ値に追加されます。列識別子の格納値は、テーブル内の列の総数によって異なります。
  - 1 ~ 62 カラム: 1 バイト
  - 63 ~ 124 カラム: 2 バイト
  - 125 ~ 186 カラム: 3 バイト

62 列を追加するごとに 1 バイトを追加します。Amazon Keyspaces では、1 つの INSERT または UPDATE ステートメントで最大 225 個の標準列を変更できることに注意してください。詳細については、「the section called “Amazon Keyspaces サービスクォータ”」を参照してください。

データ型に基づいてデータ値のエンコードされたサイズを推定する

このセクションでは、Amazon Keyspaces のさまざまなデータ型のエンコードされたサイズを推定する方法を示します。

- 文字列型 – Cassandra ASCII、TEXT、および VARCHAR文字列データ型はすべて、UTF-8 バイナリエンコーディングの Unicode を使用して Amazon Keyspaces に保存されます。Amazon Keyspaces 文字列のサイズは、UTF-8 でエンコードされたバイト数と同じです。
- 数値型 – Cassandra INT、BIGINT、SMALLINT、、および VARINT データ型はTINYINT、最大 38 桁の有効数字を持つ可変長のデータ値として Amazon Keyspaces に保存されます。先頭と末尾の 0 は切り捨てられます。これらのデータ型のサイズはいずれも、有効数字 2 桁あたり約 1 バイト+1 バイトです。
- Blob タイプ – Amazon Keyspaces BLOBの は、値の raw バイト長で保存されます。
- ブール型 – Boolean値のサイズまたはNull値は 1 バイトです。
- コレクションタイプ – コンテンツに関係なく、 LISTや などのコレクションデータ型を保存する列には 3 バイトのメタデータMAPが必要です。LIST または MAP のサイズは、(列 ID) + 合計 (入れ子要素のサイズ) + (3 バイト) です。空の LIST または MAP のサイズは (列 ID) + (3 バイト) です。個々の LIST または MAP 要素にはそれぞれ、余分な 1 バイトが必要です。
- ユーザー定義型 – ユーザー定義型 (UDT) では、その内容に関係なく、メタデータに 3 バイトが必要です。UDT 要素ごとに、Amazon Keyspaces には追加の 1 バイトのメタデータが必要です。

  UDT のエンコードされたサイズを計算するには、UDT のフィールドfield valueの field nameと から始めます。

  - フィールド名 – 最上位 UDT の各フィールド名は、識別子を使用して保存されます。識別子のストレージ値は、最上位の UDT のフィールドの総数に依存し、1～3 バイトの間で異なる場合があります。
    - 1～62 フィールド: 1 バイト
    - 63～124 フィールド: 2 バイト
    - 125 – 最大フィールド: 3 バイト

- フィールド値 – 最上位 UDT のフィールド値を保存するために必要なバイト数は、保存されるデータ型によって異なります。
  - スカラーデータ型 – ストレージに必要なバイト数は、通常の列に保存されているのと同じデータ型と同じです。
  - フローズン UDT – フローズンネストされた UDT ごとに、ネストされた UDT のサイズは CQL バイナリプロトコルと同じです。ネストされた UDT の場合、各フィールド (空のフィールドを含む) に 4 バイトが保存され、保存されたフィールドの値はフィールド値の CQL バイナリプロトコルシリアル化形式です。
  - フリーズコレクション：
    - LIST と SET – ネストされたフリーズ`LIST`または の場合`SET`、コレクションの各要素に 4 バイトとコレクションの値の CQL バイナリプロトコルシリアル化形式が保存されます。
    - MAP – ネストされたフリーズ の場合`MAP`、各キーと値のペアには次のストレージ要件があります。
      - キーごとに 4 バイトを割り当て、キーの CQL バイナリプロトコルシリアル化形式を追加します。
      - 値ごとに 4 バイトを割り当て、値の CQL バイナリプロトコルシリアル化形式を追加します。
- FROZEN キーワード – フリーズコレクション内にネストされたフリーズコレクションの場合、Amazon Keyspaces はメタデータに追加のバイトを必要としません。
- STATIC キーワード – `STATIC`列データは最大行サイズ 1 MB にはカウントされません。静的列のデータサイズを計算するには、「the section called “各論理パーティションの静的列サイズの計算”」を参照してください。

Amazon Keyspaces 機能が行サイズに与える影響を考慮する

このセクションでは、Amazon Keyspaces の機能が行のエンコードされたサイズにどのように影響するかを示します。

- クライアント側のタイムスタンプ – 機能をオンにすると、クライアント側のタイムスタンプは各行の各列に保存されます。これらのタイムスタンプで約 20 ~ 40 バイト (データによって異なります) が使用され、行のストレージとスループットのコストに影響します。クライアント側のタイムスタンプの詳細については、「」を参照してくださいthe section called “クライアントサイドのタイムスタンプ”。

- Time to Live (TTL) – この機能がオンになっている場合、TTL メタデータは行に対して約 8 バイトかかります。さらに、TTL メタデータは各行の各列に保存されます。TTL メタデータは、スカラーデータ型またはフリーズコレクションを保存する列ごとに約 8 バイトかかります。列にフリーズされていないコレクションデータ型が保存されている場合、コレクションの各要素について、TTL にはメタデータに約 8 バイトの追加が必要です。TTL が有効になっているときにコレクションデータ型を保存する列の場合、次の式を使用できます。

```
total encoded size of column = (column id) + sum (nested elements + collection
 metadata (1 byte) + TTL metadata (8 bytes)) +  collection column metadata (3 bytes)
```

  TTL メタデータは、行のストレージとスループットのコストに影響します。TTL の詳細については、「the section called “TTL によるデータの期限切れ”」を参照してください。

行のエンコードされたサイズを計算する適切な式を選択する

このセクションでは、Amazon Keyspaces 内のデータ行のストレージまたは容量スループット要件を推定するために使用できるさまざまな式を示します。

データの行の合計エンコードサイズは、目標に基づいて、次のいずれかの式に基づいて推定できます。

- スループットキャパシティ – 行のエンコードされたサイズを推定して、必要な読み取り/書き込みリクエストユニット (RRUs/WRUs) または読み取り/書き込みキャパシティユニット (RCUs/WCUs) を評価します。

```
total encoded size of row = partition key columns + clustering columns + regular
 columns
```

- ストレージサイズ – を予測する行のエンコードされたサイズを推定するには`BillableTableSizeInBytes`、行のストレージに必要なメタデータを追加します。

```
total encoded size of row = partition key columns + clustering columns + regular
 columns + row metadata (100 bytes)
```

> **Important**
>
> 列 ID、パーティションキーメタデータ、クラスタリング列メタデータ、クライアント側のタイムスタンプ、TTL、行メタデータなどのすべての列メタデータは、最大行サイズ 1 MB にカウントされます。

行サイズ計算の例

すべての列が整数型であるテーブルの例を考えてみましょう。テーブルには、パーティションキー列が 2 つ、クラスタリング列が 2 つ、通常の列が 1 つあります。このテーブルには 5 つの列があるため、列名識別子に必要なスペースは 1 バイトです。

```
CREATE TABLE mykeyspace.mytable(pk_col1 int, pk_col2 int, ck_col1 int, ck_col2 int,
 reg_col1 int, primary key((pk_col1, pk_col2),ck_col1, ck_col2));
```

この例では、次のステートメントに示すように、テーブルに行を書き込むときにデータのサイズを計算します。

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, ck_col1, ck_col2, reg_col1)
 values(1,2,3,4,5);
```

この書き込みオペレーションに必要な合計バイト数を見積もるために、次のステップを使用します。

1. 列に保存されているデータ型のバイトとメタデータバイトを追加して、パーティションキー列のサイズを計算します。この計算をすべてのパーティションキー列に対して繰り返します。

   a. パーティションキー (pk_col1) の最初の列のサイズを計算します。

   ```
   (2 bytes for the integer data type) x 2 + 1 byte for the column id + 3 bytes
    for partition key metadata = 8 bytes
   ```

   b. パーティションキー (pk_col2) の 2 番目の列のサイズを計算します。

   ```
   (2 bytes for the integer data type) x 2 + 1 byte for the column id + 3 bytes
    for partition key metadata = 8 bytes
   ```

   c. 両方の列を足して、パーティションキー列の合計サイズを見積もります。

```
8 bytes + 8 bytes = 16 bytes for the partition key columns
```

2. 列に保存されているデータ型のバイトとメタデータのバイトを足して、クラスタリング列のサイズを計算します。すべてのクラスタリング列に対してこの計算を繰り返します。

   a. クラスタリング列 (ck_col1) の最初の列のサイズを計算します。

   ```
   (2 bytes for the integer data type) x 2 + 20% of the data value (2 bytes) for
    clustering column metadata + 1 byte for the column id  = 6 bytes
   ```

   b. クラスタリング列 (ck_col2) の 2 番目の列のサイズを計算します。

   ```
   (2 bytes for the integer data type) x 2 + 20% of the data value (2 bytes) for
    clustering column metadata + 1 byte for the column id = 6 bytes
   ```

   c. 両方の列を足して、クラスタリング列の合計サイズを見積もります。

   ```
   6 bytes + 6 bytes = 12 bytes for the clustering columns
   ```

3. 通常の列のサイズを足します。この例では、1 バイトが必要で列 ID が 1 バイトが必要で、余分な 1 バイトが必要です。
4. 最後に、エンコードされた行の合計サイズを取得するには、すべての列のバイトを合計します。ストレージの請求対象サイズを見積もるには、行メタデータに 100 バイトを追加します。

   ```
   16 bytes for the partition key columns + 12 bytes for clustering columns + 3 bytes
    for the regular column + 100 bytes for row metadata = 131 bytes.
   ```

Amazon CloudWatch でサーバーレスリソースをモニタリングする方法については、「the section called “CloudWatch によるモニタリング”」を参照してください。

Amazon Keyspaces で読み取り/書き込みスループットのキャパシティ消費量を推定する

Amazon Keyspaces でデータを読み書きした際に、クエリで消費される読み取り/書き込みリクエストユニット (RRU/WRU) または読み取り/書き込みキャパシティユニット (RCU/WCU) の量は、Amazon Keyspaces がそのクエリを実行するために処理しなければならないデータの総量によって決まります。場合によっては、クライアントに返されるデータが、Amazon Keyspaces がク

エリを処理するために読み取る必要があるデータの一部に過ぎないこともあります。条件付き書き込みの場合、条件チェックが失敗に終わったとしても、書き込みキャパシティは消費されます。

リクエストのために処理されるデータの総量を推定するには、行のエンコード後のサイズと、行の合計数を考慮する必要があります。このトピックでは、一般的なシナリオとアクセスパターンの例をいくつか取り上げ、Amazon Keyspaces がクエリを処理する方法と、それがキャパシティ消費量にどのように影響するかを説明します。これらの例を参考にして実際のテーブルのキャパシティ要件を見積もり、Amazon CloudWatch を使用して、該当するユースケースの読み取りおよび書き込みのキャパシティ消費状況を観察できます。

Amazon Keyspaces でエンコード後の行サイズを計算する方法については、「the section called “行サイズの推定”」を参照してください。

トピック

- Amazon Keyspaces で範囲クエリのキャパシティ消費量を推定する
- 限定クエリの読み取りキャパシティ消費量を推定する
- テーブルスキャンの読み取りキャパシティ消費量を推定する
- Amazon Keyspaces で軽量トランザクションのキャパシティ消費量を推定する
- Amazon Keyspaces の静的列のキャパシティ消費量を推定する
- Amazon Keyspaces でマルチリージョンテーブルのキャパシティを推定およびプロビジョニングする
- Amazon CloudWatch を使用して Amazon Keyspaces の読み取りと書き込みのキャパシティ消費量を推定する

Amazon Keyspaces で範囲クエリのキャパシティ消費量を推定する

範囲クエリの読み取りキャパシティの消費量を調べるために、次のサンプルテーブルを使用します。このテーブルはオンデマンドキャパシティモードを使用しています。

```
pk1 | pk2 | pk3 | ck1 | ck2 | ck3 | value
-----+-----+-----+-----+-----+-----+-------
a | b | 1 | a | b | 50 | <any value that results in a row size larger than 4KB>
a | b | 1 | a | b | 60 | value_1
a | b | 1 | a | b | 70 | <any value that results in a row size larger than 4KB>
```

このテーブルに対して、次のクエリを実行します。

```
SELECT * FROM amazon_keyspaces.example_table_1 WHERE pk1='a' AND pk2='b' AND pk3=1 AND
 ck1='a' AND ck2='b' AND ck3 > 50 AND ck3 < 70;
```

クエリから次の結果セットが返されます。Amazon Keyspaces が実行する読み取りオペレーションでは、LOCAL_QUORUM 整合性モードで 2 RRU が消費されます。

```
pk1 | pk2 | pk3 | ck1 | ck2 | ck3 | value
-----+-----+-----+-----+-----+-----+-------
a | b | 1 | a | b | 60 | value_1
```

Amazon Keyspaces はこのクエリを処理するため、2 RRU を消費して ck3=60 と ck3=70 の値を含む行を評価します。ただし、Amazon Keyspaces が返すのは、クエリに指定されている WHERE 条件が true である行、つまり値が ck3=60 の行だけです。クエリで指定された範囲を評価するために、範囲の上限に一致する行 (この例では ck3 = 70) を読み取りますが、その行を結果では返しません。読み取りキャパシティの消費量は、返されたデータではなく、クエリの処理時に読み取られたデータに基づきます。

限定クエリの読み取りキャパシティ消費量を推定する

LIMIT 句を使用したクエリを処理する際、Amazon Keyspaces はクエリで指定された条件と一致するデータを探すために、最大ページサイズまでの行数を読み取ります。最初のページで LIMIT 値を満たす十分なデータが見つからない場合、1 回以上のページ分割呼び出しが必要になることがあります。次のページの読み取りを続行するには、ページ分割トークンを使用できます。デフォルトのページサイズは 1 MB です。LIMIT 句の使用時に読み取りキャパシティの消費量を減らすには、ページサイズを小さくします。ページ分割の詳細については、「the section called “結果のページ分割”」を参照してください。

例として、次のクエリを見てみましょう。

```
SELECT * FROM my_table WHERE partition_key=1234 LIMIT 1;
```

ページサイズが設定されていない場合、Amazon Keyspaces は 1 行しか返さないとしても 1 MB 分のデータを読み取ります。Amazon Keyspaces に 1 行だけ読み取らせるには、このクエリでページサイズを 1 に設定します。その場合、Amazon Keyspaces は 1 行のみを読み取ります。ただし、Time-to-Live の設定やクライアント側のタイムスタンプに基づいて期限切れになった行がないことが前提です。

PAGE SIZE パラメータは、Amazon Keyspaces がクライアントに返す行数ではなく、リクエストごとにディスクから Amazon Keyspaces がスキャンする行数を決定します。Amazon Keyspaces は、

指定したフィルターを適用します。たとえば、非キー列の不等式や、ディスク上のデータをスキャンLIMITした後の などです。を明示的に設定しない場合PAGE SIZE、Amazon Keyspaces はフィルターを適用する前に最大 1MB のデータを読み取ります。例えば、 を指定LIMIT 1せずに を使用している場合PAGE SIZE、Amazon Keyspaces は limit 句を適用して 1 行のみ返す前に、ディスクから数千行を読み取ることができます。

オーバーリードを回避するには、 を減らしPAGE SIZEて、各フェッチの Amazon Keyspaces スキャン行数を減らします。例えば、クエリLIMIT 5で を定義した場合、Amazon Keyspaces がページ分割された呼び出しごとに 5～10 行のみをスキャンするように、 PAGE SIZEを 5～10 の値に設定します。この数を変更して、フェッチの数を減らすことができます。ページサイズより大きい制限の場合、Amazon Keyspaces は合計結果数をページ分割状態で維持します。LIMIT が 10,000 行の場合、Amazon Keyspaces はこれらの結果を 5,000 行ずつの 2 ページで取得できます。1MB の制限は、設定されたページサイズの上限です。

テーブルスキャンの読み取りキャパシティ消費量を推定する

ALLOW FILTERING オプションを使用したクエリなど、結果的にテーブルのフルスキャンが実行されるクエリも、結果として返されるデータ量よりも読み取り処理量の方が多いクエリの例です。なお、読み取りキャパシティの消費量は、返されたデータではなく、読み取られたデータに基づきます。

テーブルスキャンの例として、次のオンデマンドキャパシティモードのサンプルテーブルを使用します。

```
pk | ck | value
---+----+---------
pk | 10 | <any value that results in a row size larger than 4KB>
pk | 20 | value_1
pk | 30 | <any value that results in a row size larger than 4KB>
```

Amazon Keyspaces でオンデマンドキャパシティモードのテーブルを作成した場合、デフォルトのパーティション数は 4 です。このサンプルテーブルでは、すべてのデータが 1 つのパーティションに保存され、残り 3 つのパーティションは空です。

このテーブルに対して、次のクエリを実行します。

```
SELECT * from amazon_keyspaces.example_table_2;
```

このクエリの結果、テーブルスキャンが実行されます。Amazon Keyspaces がテーブルの 4 つのパーティションをすべてスキャンし、LOCAL_QUORUM 整合性モードで 6 RRU を消費します。ま

ず、Amazon Keyspaces は 3 RRU を消費して、pk='pk' である 3 つの行を読み取ります。次に、さらに 3 RRU を消費して、テーブルの 3 つの空パーティションをスキャンします。このクエリではテーブルスキャンが行われるため、データが空のパーティションも含め、テーブルのすべてのパーティションがスキャンされます。

Amazon Keyspaces で軽量トランザクションのキャパシティ消費量を推定する

軽量トランザクション (LWT) では、テーブルのデータに対して条件付きの書き込みオペレーションを実行できます。条件付きの更新オペレーションは、現在の状態を評価する条件に基づいてレコードを挿入、更新、削除する場合に便利です。

Amazon Keyspaces では、すべての書き込みオペレーションで LOCAL_QUORUM 整合性が必要であり、LWT の使用に追加料金はかかりません。LWTs の違いは、LWT 条件チェックの結果が の場合FALSE、Amazon Keyspaces は書き込みキャパシティーユニット (WCUs) または書き込みリクエストユニット (WRUs。消費WCUs/WRUsの数は、行のサイズによって異なります。

たとえば、行サイズが 2 KB の場合、失敗した条件付き書き込みは 2 つの WCUs/WRUs。行がテーブルに現在存在しない場合、オペレーションは 1 つの WCUs/WRUs。

条件チェックの失敗の原因となったリクエストの数を確認するには、CloudWatch で ConditionalCheckFailedメトリクスをモニタリングできます。

Time to Live (TTL) を使用したテーブルの LWT コストの見積もり

LWTsでは、クライアント側のタイムスタンプを使用しない TTL で設定されたテーブルに対して、追加の読み込みキャパシティーユニット (RCUs) または読み込みリクエストユニット (RRUs) が必要になる場合があります。で IF EXISTSまたは IF NOT EXISTSキーワード条件チェック結果を使用する場合FALSE、次のキャパシティユニットが消費されます。

- RCUs/RRUs 行が存在する場合、消費される RCUs/RRUs は既存の行のサイズに基づきます。
- RCUs/RRUs 行が存在しない場合、単一の RCU/RRU が消費されます。

評価された条件が書き込みオペレーションに成功した場合、WCUs/WRUsは新しい行のサイズに基づいて消費されます。

Amazon Keyspaces の静的列のキャパシティ消費量を推定する

クラスター化列を含む Amazon Keyspaces テーブルでは、STATIC キーワードを使用して静的列を作成できます。静的列に保存されている値は論理パーティション内のすべての行で共有されます。この列の値を更新すると、Amazon Keyspaces によりパーティション内のすべての行に変更が自動で適用されます。

このセクションでは、静的列に書き込むときのエンコードされたデータサイズを計算する方法について説明します。このプロセスは、行の非静的列にデータを書き込むプロセスとは別に処理されます。静的データのサイズクォータに加えて、静的列の読み取りオペレーションと書き込みオペレーションは、テーブルの計測とスループットキャパシティにも個別に影響します。静的列と範囲読み取り結果のページ分割を使用する場合の Apache Cassandra の機能上の違いについては、「the section called “Pagination (ページ分割)”」を参照してください。

トピック

- Amazon Keyspaces で各論理パーティションの静的列サイズを計算する
- Amazon Keyspaces で静的データに対する読み取り/書き込みオペレーションのキャパシティスループット要件を推定する

Amazon Keyspaces で各論理パーティションの静的列サイズを計算する

このセクションでは、Amazon Keyspaces でエンコードされた静的列サイズを推定する方法について説明します。エンコードされたサイズは、請求額とクォータの使用量を計算するときに使用されます。テーブルのプロビジョンドスループット性能要件を計算するときも、エンコードされたサイズを使用する必要があります。Amazon Keyspaces 内のエンコードされた静的列サイズを計算するには、次のガイドラインを使用します。

- パーティションには、最大 2048 バイトのデータを保存できます。パーティションキーの各キー列には、最大 3 バイトのメタデータが必要です。これらのメタデータバイトは、パーティションあたり 1 MB の静的データサイズクォータにカウントされます。静的データのサイズを計算するときには、各パーティションキー列で上限である 3 バイトのメタデータが使用されていることを想定しておくべきです。
- データ型に基づいて、静的列データ値の生のサイズを使用します。 のデータ型の詳細については、「the section called “データ型”」を参照してください。
- メタデータのために静的データのサイズに 104 バイトを足します。

- クラスタリング列と通常の非プライマリキー列は、静的データのサイズにはカウントされません。行内の非静的データのサイズを見積もる方法については、「the section called “行サイズの推定”」を参照してください。

エンコードされた静的列の合計サイズは、次の式に基づいています。

```
partition key columns + static columns + metadata = total encoded size of static data
```

すべての列が整数型であるテーブルの例を考えてみましょう。テーブルには、パーティションキー列が 2 つ、クラスタリング列が 2 つ、通常の列が 1 つ、静的列が 1 つあります。

```
CREATE TABLE mykeyspace.mytable(pk_col1 int, pk_col2 int, ck_col1 int, ck_col2
 int, reg_col1 int, static_col1 int static, primary key((pk_col1, pk_col2),ck_col1,
 ck_col2));
```

この例では、次のステートメントの静的データのサイズを計算します。

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, static_col1) values(1,2,6);
```

この書き込みオペレーションに必要な合計バイト数を見積もるために、次のステップを使用します。

1. 列に保存されているデータ型のバイトとメタデータバイトを追加して、パーティションキー列のサイズを計算します。この計算をすべてのパーティションキー列に対して繰り返します。

   a. パーティションキー (pk_col1) の最初の列のサイズを計算します。

   ```
   4 bytes for the integer data type + 3 bytes for partition key metadata = 7
    bytes
   ```

   b. パーティションキー (pk_col2) の 2 番目の列のサイズを計算します。

   ```
   4 bytes for the integer data type + 3 bytes for partition key metadata = 7
    bytes
   ```

   c. 両方の列を足して、パーティションキー列の合計サイズを見積もります。

   ```
   7 bytes + 7 bytes = 14 bytes for the partition key columns
   ```

2. 静的列のサイズを足します。この例では、整数を保存している列 (4 バイトが必要) が 1 つしかありません。

3. 最後に、静的列データのエンコードされたサイズの合計を算出するには、プライマリキー列と静的列のバイト数を合計し、メタデータのために追加で 104 バイトを足します。

```
14 bytes for the partition key columns + 4 bytes for the static column + 104 bytes
 for metadata = 122 bytes.
```

静的データと非静的データを同じステートメントで更新することもできます。書き込みオペレーションの合計サイズを見積もるには、まず非静的データ更新のサイズを計算する必要があります。次に、次の the section called “行サイズの推定” での例に示すように、行の更新のサイズを計算し、結果を足します。

この場合、合計で 2 MB を書き込むことができます。1 MB が生の最大行サイズクォータで、もう 1 MB は論理パーティションごとの最大静的データサイズのクォータです。

同じステートメント内の静的データと非静的データの更新の合計サイズを計算するには、次の式を使用します。

```
(partition key columns + static columns + metadata = total encoded size of static data)
 + (partition key columns + clustering columns + regular columns + row metadata = total
 encoded size of row)
= total encoded size of data written
```

すべての列が整数型であるテーブルの例を考えてみましょう。テーブルには、パーティションキー列が 2 つ、クラスタリング列が 2 つ、通常の列が 1 つ、静的列が 1 つあります。

```
CREATE TABLE mykeyspace.mytable(pk_col1 int, pk_col2 int, ck_col1 int, ck_col2
 int, reg_col1 int, static_col1 int static, primary key((pk_col1, pk_col2),ck_col1,
 ck_col2));
```

この例では、次のステートメントに示すように、テーブルに行を書き込むときにデータのサイズを計算します。

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, ck_col1, ck_col2, reg_col1,
 static_col1) values(2,3,4,5,6,7);
```

この書き込みオペレーションに必要な合計バイト数を見積もるために、次のステップを使用します。

1. 前述のように、静的データのエンコードされたサイズの合計を計算します。この例では、この合計は 122 バイトです。

2. the section called “行サイズの推定” の手順に従い、非静的データの更新に基づいて、行のエンコードされたサイズの合計を足します。この例では、行の更新の合計サイズは 134 バイトです。

```
122 bytes for static data + 134 bytes for nonstatic data = 256 bytes.
```

Amazon Keyspaces で静的データに対する読み取り/書き込みオペレーションのキャパシティスループット要件を推定する

静的データは、個々の行ではなく、Cassandra の論理パーティションに関連付けられます。Amazon Keyspaces の論理パーティションは、複数のストレージパーティションにまたがることができるので、そのサイズは事実上無制限です。その結果、Amazon Keyspaces により、静的データと非静的データに対する書き込みオペレーションが別々に計測されます。さらに、静的データと非静的データの両方を含む書き込みには、データの整合性を確保するために、基盤となる追加のオペレーションが必要です。

静的データと非静的データを混合した書き込みオペレーションを実行すると、2 つの個別の書き込みオペレーション (非静的データ用と静的データ用) が発生します。これは、オンデマンドおよび読み取り/書き込みのプロビジョンドキャパシティモードの両方に適用されます。

次の例では、静的列がある Amazon Keyspaces のテーブルのプロビジョンドスループット性能要件を計算するときに、必要な読み取りキャパシティユニット (RCU) と書き込みキャパシティユニット (WCU) を見積もる方法について説明します。次の式を使用して、静的データと非静的データの両方を含む書き込みを処理するためにテーブルで必要となるキャパシティを見積もることができます。

```
2 x WCUs required for nonstatic data + 2 x WCUs required for static data
```

例えば、アプリケーションにより 1 秒あたり 27 KB のデータが書き込まれ、各書き込みに 25.5 KB の非静的データと 1.5 KB の静的データが含まれている場合、テーブルには 56 WCU (2 x 26 WCU + 2 x 2 WCU) が必要です。

Amazon Keyspaces で、複数の行の読み取りと同じ静的データと非静的データの読み取りが計測されます。その結果、同じオペレーション内で静的データと非静的データを読み取る場合の料金は、読み取りを実行するために処理されるデータの総サイズに基づきます。

Amazon CloudWatch でサーバーレスリソースをモニタリングする方法については、「the section called “CloudWatch によるモニタリング”」を参照してください。

Amazon Keyspaces でマルチリージョンテーブルのキャパシティを推定およびプロビジョニングする

マルチリージョンテーブルのスループットキャパシティは、次の 2 とおりの方法で設定できます。

- オンデマンドキャパシティモード。書き込みリクエストユニット (WRU) で測定します。
- 自動スケーリングが有効なプロビジョンドキャパシティモード。書き込みキャパシティユニット (WCU) で測定します。

プロビジョンドキャパシティモードを自動スケーリングまたはオンデマンドキャパシティモードで使用すると、マルチリージョンテーブルにすべての に対してレプリケートされた書き込みを実行するのに十分なキャパシティを確保できます AWS リージョン。

> **Note**
>
> いずれかのリージョンでテーブルのキャパシティモードを変更すると、すべてのレプリカのキャパシティモードが変更されます。

デフォルトでは、Amazon Keyspaces はマルチリージョンテーブルでオンデマンドモードを使用します。オンデマンドモードでは、アプリケーションが実行する読み込みおよび書き込みの予想スループットを指定する必要がありません。Amazon Keyspaces は、ワークロードがこれまでに到達実績のあるトラフィックレベルまで増減した場合は、瞬時に対応します。ワークロードのトラフィックレベルが新しいピークに達した場合も、ワークロードに対応できるよう迅速に適応します。

テーブルでプロビジョンドキャパシティモードを選択した場合は、アプリケーションに必要な 1 秒あたりの読み取りキャパシティユニット (RCU) と書き込みキャパシティユニット (WCU) の数を設定する必要があります。

マルチリージョンテーブルのスループットキャパシティのニーズを計画するには、まず各リージョンで必要な 1 秒あたりの WCU の数を見積もります。次に、テーブルのレプリケート先のすべてのリージョンでの書き込みを合計し、その合計を各リージョンのキャパシティとしてプロビジョニングします。これは、1 つのリージョンで実行されるすべての書き込みを各レプリカリージョンでも繰り返す必要があるためです。

すべてのリージョンでの書き込みを処理するにはテーブルのキャパシティが不足している場合、キャパシティの例外が発生します。さらに、リージョン間レプリケーションの待機時間が長くなります。

例えば、米国東部 (バージニア北部) で 1 秒あたり 5 回の書き込み、米国東部 (オハイオ) で 1 秒あたり 10 回の書き込み、欧州 (アイルランド) で 1 秒あたり 5 回の書き込みが予想されるマルチリージョンテーブルがある場合、米国東部 (バージニア北部) 、米国東部 (オハイオ) 、欧州 (アイルランド) の各リージョンでテーブルの消費量は 20 WCU になると予想されます。つまり、この例では、テーブルの各レプリカに 20 WCU をプロビジョニングする必要があります。Amazon CloudWatch を使用して、テーブルのキャパシティ消費量をモニタリングできます。詳細については、「the section called “CloudWatch によるモニタリング”」を参照してください。

各書き込みは 1 WCU として請求されるため、この例では合計 60 WCUsが請求されます。料金の詳細については、「Amazon Keyspaces (for Apache Cassandra) pricing (Amazon Keyspaces (Apache Cassandra 向け) の料金)」を参照してください。

Amazon Keyspaces 自動スケーリングを有効にしたプロビジョンドキャパシティの詳細については、「the section called “自動スケーリングによるスループットキャパシティの管理”」を参照してください。

> **Note**
>
> テーブルがプロビジョンドキャパシティーモードで自動スケーリングを使用している場合、プロビジョニングされた書き込みキャパシティは、各リージョンの自動スケーリング設定の範囲内で変動させることができます。

Amazon CloudWatch を使用して Amazon Keyspaces の読み取りと書き込みのキャパシティ消費量を推定する

読み取りと書き込みのキャパシティ消費量を推定し、モニタリングするには、CloudWatch ダッシュボードを使用できます。Amazon Keyspaces で使用可能なメトリクスの詳細については、「the section called “メトリクスとディメンション”」を参照してください。

特定のステートメントで消費される読み取りと書き込みのキャパシティユニットを CloudWatch でモニタリングするには、以下の手順に従ってください。

1. サンプルデータを使用して新しいテーブルを作成する
2. テーブルの Amazon Keyspaces CloudWatch ダッシュボードを設定します。まずは、Github で利用可能なダッシュボードテンプレートを使用できます。
3. `ALLOW FILTERING` オプションなどを使用して CQL ステートメントを実行し、テーブルのフルスキャンで消費される読み取りキャパシティユニットをダッシュボードで確認します。

Amazon Keyspaces で読み取り/書き込みのキャパシティモードを設定する

Amazon Keyspaces には、テーブルで読み込みおよび書き込みを処理するための読み取り/書き込みキャパシティモードが 2 つあります。

- オンデマンド (デフォルト)
- プロビジョン済み

読み取り/書き込みキャパシティモードを選択すると、読み取りと書き込みのスループットの料金体系、およびテーブルスループットキャパシティの管理状態を制御できます。

トピック

- オンデマンドキャパシティモードを設定する
- プロビジョンドスループットキャパシティモードを設定する
- Amazon Keyspaces でテーブルのキャパシティモードを表示する
- キャパシティモードを変更する
- Amazon Keyspaces でオンデマンドキャパシティモード用に新規テーブルを事前ウォーミングする
- Amazon Keyspaces でオンデマンドキャパシティモード用に既存のテーブルを事前ウォーミングする

オンデマンドキャパシティモードを設定する

Amazon Keyspaces (Apache Cassandra 向け) のオンデマンドキャパシティモードは、キャパシティ計画なしで 1 秒あたりに数千ものリクエストを処理できる柔軟な請求オプションです。このオプションには、読み取りおよび書き込みリクエストのリクエストごとの支払い料金が用意されているため、使用した分だけ課金されます。

オンデマンドモードを選択すると、Amazon Keyspaces により、テーブルのスループットキャパシティを以前に到達したトラフィックレベルまで即座に拡張でき、アプリケーショントラフィックが減少した時点で元に戻すことができます。ワークロードのトラフィックレベルが新たなピークに達すると、サービスは瞬時に適応してテーブルのスループットキャパシティを増やします。新規テーブルと既存のテーブルの両方で、オンデマンドキャパシティモードを有効にできます。

以下の条件のいずれかに該当する場合、オンデマンドモードは適切なオプションです。

- 不明なワークロードを含む新しいテーブルを作成します。
- アプリケーションのトラフィックが予測不可能です。
- わかりやすい従量課金制の支払いを希望します。

オンデマンドモードを開始するには、コンソールまたは数行の Cassandra クエリ言語 (CQL) コードを使用して、新規のテーブルを作成するか、既存のテーブルを更新して、オンデマンドキャパシティモードを使用することができます。詳細については、「the section called “テーブル”」を参照してください。

トピック

- 読み取りリクエスト単位と書き込みリクエスト単位
- ピークトラフィックとスケーリングプロパティ
- オンデマンドキャパシティモードの初期スループット

読み取りリクエスト単位と書き込みリクエスト単位

オンデマンドキャパシティモードのテーブルでは、アプリケーションによりあらかじめ使用されることが予測される読み取りスループットと書き込みスループットを指定する必要はありません。Amazon Keyspaces では、読み込みリクエスト単位 (RRU) と書き込みリクエスト単位 (WRU) に関して、テーブルに対して実行する読み取りと書き込みの料金が発生します。

- 1 RRU は、最大 4 KB の行に対して、LOCAL_QUORUM 読み取りリクエスト 1 件、または LOCAL_ONE リクエスト 2 件を表します。4 KB より大きい行を読み取る必要がある場合、読み取りオペレーションには追加の RRU が使用されます。必要な RRU の総数は、行サイズと、LOCAL_QUORUM または LOCAL_ONE の読み取り整合性の使用の有無によって異なります。例えば、8 KB の行の読み取りには、LOCAL_QUORUM 読み取り整合性を使用する 場合は 2 RRU、LOCAL_ONE 読み取り整合性を選択した場合は 1 RRU が必要です。
- 1 WRU は、最大で 1 KB の行 1 つに対する 1 回の書き込みを表します。すべての書き込みでは LOCAL_QUORUM 整合性が使用されており、軽量トランザクション (LWT) の使用には追加料金はかかりません。1 KB より大きい行を書き込む必要がある場合、書き込みオペレーションでは追加の WRU が使用されます。必要な WRU の総数は、行サイズに応じて異なります。例えば、行サイズが 2 KB の場合、1 件の書き込みリクエストを実行するには 2 WRU が必要です。

サポートされている整合性レベルの詳細については、「the section called “サポートされている Cassandra の整合性レベル”」を参照してください。

ピークトラフィックとスケーリングプロパティ

オンデマンドキャパシティモードが使用されている Amazon Keyspaces テーブルは、アプリケーションのトラフィックボリュームに自動的に適応します。オンデマンドキャパシティーモードは、テーブルにおける前のピークトラフィックの最大 2 倍まで瞬時に対応します。例えば、アプリケーションのトラフィックパターンは、1 秒あたりの LOCAL_QUORUM 読み取り数 5,000 ~ 10,000 の範囲で変動する可能性があります。1 秒あたりの読み取り数 10,000 は過去のトラフィックピークです。

このパターンでは、オンデマンドキャパシティモードは、1 秒あたりの最大読み取り数 20,000 の持続トラフィックに即座に対応します。アプリケーションによって 1 秒あたりの読み取り数 20,000 のトラフィックを維持する場合、そのピークは新たに過去のピークになり、その後のトラフィックでは 1 秒あたりの最大読み取り数が 40,000 に到達できます。

1 つのテーブルで過去のピークの 2 倍以上が必要な場合、Amazon Keyspaces では、トラフィックボリュームの増加に合わせて、割り当るキャパシティが自動的に増加されます。これにより、そのテーブルに、追加のリクエストを処理できるだけの十分なスループットキャパシティがあることを確認できます。ただし、30 分以内に過去のピークの 2 倍を超えると、スループットキャパシティ不足エラーが発生する可能性があります。

例えば、読み取りの整合性が強力で、アプリケーションのトラフィックパターンが 1 秒あたりの読み取り数 5,000 ~ 10,000 の範囲で変動し、1 秒あたりの読み取り数 20,000 が過去に到達したトラフィックピークである場合を考えてみます。この場合、このサービスでは、1 秒あたりの最大読み取り数 40,000 に到達する 30 分以上前の段階で、トラフィックを増やすことが推奨されます。

テーブルの読み取りと書き込みのキャパシティ消費量の推定方法については、「the section called “キャパシティ消費量の推定”」を参照してください。

アカウントのデフォルトクォータの詳細およびクォータを引き上げる方法については、「クォータ」を参照してください。

オンデマンドキャパシティモードの初期スループット

オンデマンドキャパシティモードが有効になっている新規のテーブルを作成した場合、または、初めて既存のテーブルを最近初めてオンデマンドキャパシティモードに切り替えた場合に、テーブルには、過去にオンデマンドキャパシティモードでトラフィックが扱われたことがなくても、以下に示す過去のピーク設定があります。

- オンデマンドキャパシティモードで新たに作成されたテーブル: 過去のピークは、2,000 WRU および 6,000 RRU です。過去のピークを即座に 2 倍まで引き上げることができます。これにより、新たに作成されたオンデマンドテーブルにおいて、最大で 4,000 WRU および 12,000 RRU です。

- オンデマンドキャパシティモードに切り替えられた既存のテーブル: 過去のピークは、テーブルにプロビジョニングされた過去の WCU と RCU の半分、またはオンデマンドキャパシティモードで新しく作成されたテーブルの設定の、どちらか高い方になります。

プロビジョンドスループットキャパシティモードを設定する

プロビジョンドスループット性能モードを選択した場合は、アプリケーションに必要な 1 秒あたりの読み込みと書き込みの回数を指定します。これにより、Amazon Keyspaces の使用状況を管理して、定義されたリクエストレート以下を維持し、予測可能性を維持できます。プロビジョンドスループットのオートスケーリングの詳細については、「the section called “自動スケーリングによるスループットキャパシティの管理”」を参照してください。

以下の条件のいずれかに該当する場合、プロビジョンドスループット性能モードは適切なオプションです。

- アプリケーションのトラフィックが予測可能です。
- トラフィックが一定している、または徐々に増加するアプリケーションを実行します。
- 容量要件を予測できます。

読み取りキャパシティユニットと書き込みキャパシティユニット

プロビジョンドスループット性能モードのテーブルでは、読み取りキャパシティユニット (RCU) と書き込みキャパシティユニット (WCU) の観点でスループットキャパシティを指定できます。

- 1 RCU は、最大 4 KB の 1 行に対して、1 秒あたり 1 回の LOCAL_QUORUM 読み取り、または 1 秒あたり 2 回の LOCAL_ONE 読み取りを表しています。4 KB より大きい行を読み取る必要がある場合、読み取りオペレーションには追加の RCU が使用されます。

  必要な RCU の総数は、行サイズ、および、LOCAL_QUORUM と LOCAL_ONE の読み取りのどちらを使用するかによって異なります。例えば、行サイズが 8 KB の場合、1 秒あたり 1 回の LOCAL_QUORUM 読み取りを維持するには 2 RCU が必要になり、LOCAL_ONE 読み取りを選択した場合は 1 RCU が必要になります。
- 1 WCU は、最大 1 KB の 1 行 に対する 1 回の書き込みを表します。すべての書き込みでは LOCAL_QUORUM 整合性が使用されており、軽量トランザクション (LWT) の使用には追加料金はかかりません。1 KB より大きい行を書き込む必要がある場合、書き込みオペレーションでは追加の WCU が使用されます。

必要な WCU の総数は、行サイズに応じて異なります。例えば、行サイズが 2 KB の場合、1 秒あたり 1 件の書き込みリクエスト処理を維持するには 2 WCU が必要です。テーブルの読み取りと書き込みのキャパシティ消費量を推定する方法の詳細については、「the section called “キャパシティ消費量の推定”」を参照してください。

アプリケーションでこれより大きな行 (上限は Amazon Keyspaces の最大行サイズ 1 MB) の読み取りまたは書き込みが行われると、消費されるキャパシティユニットがさらに増えます。行サイズを見積もる方法については、「the section called “行サイズの推定”」を参照してください。例えば、6 RCU と 6 WCU のプロビジョニングされたテーブルを作成するとします。これらの設定により、アプリケーションで次のことが可能になります。

- 1 秒あたり最大 24 KB の LOCAL_QUORUM 読み取りの実行 (4 KB × 6 RCU)。
- 1 秒あたり最大 48 KB の LOCAL_ONE 読み取りの実行 (読み取りスループットは 2 倍)
- 1 秒あたり最大 6 KB の書き込みの実行 (1 KB × 6 WCU)。

プロビジョンドスループットとは、アプリケーションによりテーブルから消費されるスループットキャパシティの上限です。アプリケーションがプロビジョンドスループット性能を超えると、キャパシティ不足エラーが発生する可能性があります。

例えば、十分なスループットキャパシティがない読み取りリクエストは、Read_Timeout 例外が発生して失敗し、ReadThrottleEvents メトリクスに投稿されます。十分なスループットキャパシティがない書き込みリクエストは、Write_Timeout 例外が発生して失敗し、WriteThrottleEvents メトリクスに投稿されます。

Amazon CloudWatch を使用して、プロビジョンドスループットメトリクス、実際のスループットメトリクス、キャパシティ不足イベントを監視することができます。これらのメトリクスの詳細については、the section called “メトリクスとディメンション” を参照してください。

> **Note**
>
> キャパシティ不足によるエラーが繰り返し発生すると、クライアント側のドライバー固有例外が発生する可能性があります。例えば、DataStax Java ドライバーは NoHostAvailableException により失敗します。

テーブルのスループットキャパシティ設定を変更する場合は、 AWS Management Console や、CQL を使用する ALTER TABLE ステートメントを使用できます。詳細については「the section called "ALTER TABLE"」を参照してください。

アカウントのデフォルトクォータの詳細およびクォータを引き上げる方法については、「クォータ」を参照してください。

Amazon Keyspaces でテーブルのキャパシティモードを表示する

Amazon Keyspaces システムキースペースでシステムテーブルのクエリを実行して、テーブルに関するキャパシティモード情報を確認することができます。テーブルにオンデマンドスループット性モードまたはプロビジョンドスループット性能モードが使用されているかどうかも確認できます。テーブルにプロビジョンドスループット性能モードが設定されている場合、テーブルにプロビジョニングされたスループットキャパシティを確認できます。

を使用して、テーブルのキャパシティモード AWS CLI を表示することもできます。

テーブルのプロビジョンドスループットを変更するには、「the section called "キャパシティモードを変更する"」を参照してください。

Cassandra Query Language (CQL)

例

```
SELECT * from system_schema_mcs.tables where keyspace_name = 'mykeyspace' and
 table_name = 'mytable';
```

オンデマンドキャパシティモードで設定されたテーブルからは、次の結果が返されます。

```
{
   "capacity_mode":{
      "last_update_to_pay_per_request_timestamp":"1579551547603",
      "throughput_mode":"PAY_PER_REQUEST"
   }
}
```

プロビジョンドスループット性能モードで構成されたテーブルからは、次の結果が返されます。

```
{
   "capacity_mode":{
      "last_update_to_pay_per_request_timestamp":"1579048006000",
      "read_capacity_units":"5000",
```

```
      "throughput_mode":"PROVISIONED",
      "write_capacity_units":"6000"
   }
}
```

`last_update_to_pay_per_request_timestamp` 値はミリ秒単位で測定されます。

CLI

を使用してテーブルのスループットキャパシティモードを表示する AWS CLI

```
aws keyspaces get-table --keyspace-name myKeyspace --table-name myTable
```

このコマンドの出力は、テーブルがプロビジョンドキャパシティモードの場合は、次のようになります。

```
"capacitySpecification": {
        "throughputMode": "PROVISIONED",
        "readCapacityUnits": 4000,
        "writeCapacityUnits": 2000
    }
```

オンデマンドモードのテーブルの場合、出力は次のようになります。

```
"capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2024-10-03T10:48:19.092000+00:00"
    }
```

キャパシティモードを変更する

テーブルをプロビジョンドキャパシティモードからオンデマンドキャパシティモードに切り替えると、Amazon Keyspaces によりテーブルおよびパーティションの構造にいくつかの変更が加えられます。この処理には数分かかることもあります。切り替え期間中、テーブルでは、以前にプロビジョニングされた WCU および RCU の両方と整合性のあるスループットが得られます。

オンデマンドキャパシティモードからプロビジョンドキャパシティモードに戻すと、テーブルでは、テーブルがオンデマンドキャパシティモードに設定されたときに到達した前のピークと整合性のあるスループットが得られます。

キャパシティモードを切り替える際は、次の待機期間が適用されます。

- オンデマンドモードで新しく作成したテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、オンデマンドモードに戻すことができるのは、テーブルの作成タイムスタンプから 24 時間後のみです。
- オンデマンドモードの既存のテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、キャパシティモードをプロビジョンドからオンデマンドに切り替えることができるのは、24 時間に 1 回のみです。

Cassandra Query Language (CQL)

CQL を使用してテーブルのスループットキャパシティモードを変更する

1. テーブルのキャパシティモードを PROVIOSIONED に変更するには、ワークロードの予想されるピーク値に基づいて読み取りキャパシティユニットと書き込みキャパシティユニットを設定する必要があります。次のステートメントは、その一例です。このステートメントを実行して、テーブルの読み取りキャパシティユニットまたは書き込みキャパシティユニットを調整することもできます。

```
ALTER TABLE catalog.book_awards WITH CUSTOM_PROPERTIES={'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 6000,
 'write_capacity_units': 3000}};
```

プロビジョンドキャパシティモードと自動スケーリングを設定するには、「the section called “既存のテーブルに対して自動スケーリングを設定する”」を参照してください。

2. テーブルのキャパシティモードをオンデマンドモードに変更するには、スループットモードを PAY_PER_REQUEST に設定します。以下のステートメントは、この例です。

```
ALTER TABLE catalog.book_awards WITH CUSTOM_PROPERTIES={'capacity_mode':
{'throughput_mode': 'PAY_PER_REQUEST'}};
```

3. 次のステートメントを使用して、テーブルのキャパシティモードを確認できます。

```
SELECT * from system_schema_mcs.tables where keyspace_name = 'catalog' and
 table_name = 'book_awards';
```

オンデマンドキャパシティモードで設定されたテーブルからは、次の結果が返されます。

```
{
   "capacity_mode":{
```

```
      "last_update_to_pay_per_request_timestamp":"1727952499092",
      "throughput_mode":"PAY_PER_REQUEST"
   }
}
```

last_update_to_pay_per_request_timestamp 値はミリ秒単位で測定されます。

CLI

を使用してテーブルのスループットキャパシティモードを変更する AWS CLI

1. テーブルのキャパシティモードを PROVIOSIONED に変更するには、ワークロードの予想されるピーク値に基づいて読み取りキャパシティユニットと書き込みキャパシティユニットを設定する必要があります。次のコマンドは、その一例です。このコマンドを実行して、テーブルの読み取りキャパシティユニットまたは書き込みキャパシティユニットを調整することもできます。

```
aws keyspaces update-table --keyspace-name catalog --table-name book_awards
                                    \--capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=6000,writeCapacityUnits=3000
```

プロビジョンドキャパシティモードと自動スケーリングを設定するには、「the section called “既存のテーブルに対して自動スケーリングを設定する”」を参照してください。

2. テーブルのキャパシティモードをオンデマンドモードに変更するには、スループットモードを PAY_PER_REQUEST に設定します。以下のステートメントは、この例です。

```
aws keyspaces update-table --keyspace-name catalog --table-name book_awards
                                    \--capacity-specification
 throughputMode=PAY_PER_REQUEST
```

3. 次のコマンドを使用して、テーブルに設定されているキャパシティモードを確認できます。

```
aws keyspaces get-table --keyspace-name catalog --table-name book_awards
```

オンデマンドモードのテーブルの場合、出力は次のようになります。

```
"capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2024-10-03T10:48:19.092000+00:00"
```

```
    }
```

Amazon Keyspaces でオンデマンドキャパシティモード用に新規テーブルを事前ウォーミングする

Amazon Keyspaces はスループットに基づいてストレージパーティションを自動的にスケールしますが、新しいテーブルの作成時や新しいスループットピークの到達時には、必要なストレージパーティションの割り当てに時間がかかる場合があります。オンデマンドキャパシティモードおよびプロビジョンドキャパシティモードのテーブルで、スループットの急増に対応できる十分なストレージパーティションを確保するために、新規または既存のテーブルを事前ウォーミングすることができます。

新しいテーブルを事前ウォーミングする一般的なシナリオは、別のデータベースからのデータの移行時に、短期間でテラバイト単位のデータをロードしなければならないような場合です。

オンデマンドテーブルでは、トラフィック量が増えるに従い、Amazon Keyspaces がキャパシティの割り当て量を自動的に増やします。新規のオンデマンドテーブルでは、1 秒あたり最大 4,000 回の書き込みと、1 秒あたり 12,000 回 (強整合性) または 24,000 回 (結果整合性) の読み取りを処理できます。オンデマンドテーブルのトラフィックは、過去に記録されたスループットに基づいて、時間の経過とともに増えていきます。

新しいテーブルの設定を上回るピークキャパシティの急増が予想される場合は、予想されるピークキャパシティに合わせてテーブルを事前ウォーミングしておくことができます。

Amazon Keyspaces でオンデマンドキャパシティモード用に新規テーブルを事前ウォーミングするには、以下の手順に従ってください。既存のテーブルを事前ウォーミングするには、「the section called “オンデマンドキャパシティに備えた既存のテーブルの事前ウォーミング”」を参照してください。

まず、プロビジョンドモードのアカウントとテーブルのクォータを確認し、必要に応じて調整しておきましょう。

Console

オンデマンドキャパシティモード用に新規テーブルを事前ウォーミングする方法

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。

2. ナビゲーションペインで [Tables] (テーブル) を選択して、[Create table (テーブルの作成)] を選択します。
3. [Table details (テーブルの詳細)] セクションの [Create table (テーブルの作成)] ページで、キースペースを選択し、新しいテーブルに名前を付けます。
4. [列] セクションで、テーブルのスキーマを作成します。
5. [プライマリキー] セクションで、テーブルのプライマリキーを定義し、オプションでクラスタリング列を選択します。
6. [Table settings (テーブルの設定)] セクションで、[Customize settings (設定のカスタマイズ)] を選択します。
7. [Read/write capacity settings (読み取り/書き込みキャパシティの設定)] に進みます。
8. [Capacity mode (キャパシティモード)] で、[Provisioned (プロビジョン)] を選択します。
9. [読み込みキャパシティ] セクションで、[自動スケーリング] の選択を解除します。

   テーブルの [プロビジョニングされたキャパシティユニット] を、予想されるピーク値に設定します。
10. [書き込みキャパシティ] セクションで、前の手順で定義した読み取りキャパシティと同じ設定を選択するか、またはキャパシティの値を手動で設定します。
11. [テーブルの作成] を選択します。指定したキャパシティ設定でテーブルが作成されます。
12. テーブルのステータスが [アクティブ] に変わったら、テーブルを [オンデマンド] キャパシティモードに切り替えることができます。

Cassandra Query Language (CQL)

CQL を使用してオンデマンドモード用に新規テーブルを事前ウォーミングする

1. プロビジョンドモードで新しいテーブルを作成し、新しいテーブルの読み取りと書き込みの予想ピークキャパシティを指定します。以下のステートメントは、この例です。

```
CREATE TABLE catalog.book_awards (
   year int,
   award text,
   rank int,
   category text,
   book_title text,
   author text,
   publisher text,
   PRIMARY KEY ((year, award), category, rank))
```

```
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 18000,
        'write_capacity_units': 6000
     }
};
```

2. テーブルのステータスを確認します。次のステートメントを使用できます。

```
SELECT keyspace_name, table_name, status FROM system_schema_mcs.tables WHERE
 keyspace_name = 'catalog' AND table_name = 'book_awards';

 keyspace_name | table_name      | status
---------------+-----------------+--------
       catalog | book_awards     | ACTIVE

(1 rows)
```

3. テーブルのステータスが ACTIVE の場合、次のステートメントを使用してスループットモードを PAY_PER_REQUEST に設定し、テーブルのキャパシティモードをオンデマンドモードに変更できます。以下のステートメントは、この例です。

```
ALTER TABLE catalog.book_awards WITH CUSTOM_PROPERTIES={'capacity_mode':
{'throughput_mode': 'PAY_PER_REQUEST'}};
```

4. 次のステートメントを使用して、テーブルがオンデマンドモードになったことを確認し、テーブルのステータスを確認できます。

```
SELECT * from system_schema_mcs.tables where keyspace_name = 'catalog' and
 table_name = 'book_awards';
```

CLI

を使用して、オンデマンドキャパシティモード用に新しいテーブルを事前ウォーミングする AWS CLI

1. プロビジョンドモードで新しいテーブルを作成し、その新規テーブルの読み取りと書き込みの予想ピークキャパシティの値を指定します。以下のステートメントは、この例です。

```
aws keyspaces create-table --keyspace-name catalog --table-name book_awards
```

```
                                    \--schema-definition
 'allColumns=[{name=pk,type=int},{name=ck,type=int}],partitionKeys=[{name=pk},
{name=ck}]'
                                    \--capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=18000,writeCapacityUnits=6000
```

2. テーブルのステータスを確認します。次のステートメントを使用できます。

```
aws keyspaces get-table --keyspace-name catalog --table-name book_awards
```

3. テーブルがアクティブになっていて、キャパシティがプロビジョニングされている場合は、テーブルをオンデマンドモードに変更できます。次に例を示します。

```
aws keyspaces update-table --keyspace-name catalog --table-name book_awards --
capacity-specification throughputMode=PAY_PER_REQUEST
```

4. 次のステートメントを使用して、テーブルがオンデマンドモードになったことを確認し、テーブルのステータスを確認できます。

```
aws keyspaces get-table --keyspace-name catalog --table-name book_awards
```

テーブルがオンデマンドキャパシティモードでアクティブになったら、以前のプロビジョンドキャパシティモードと同様のスループットキャパシティを処理できるようになっています。

Amazon Keyspaces でオンデマンドキャパシティモード用に既存のテーブルを事前ウォーミングする

Amazon Keyspaces はスループットに基づいてストレージパーティションを自動的にスケールしますが、新しいテーブルの作成時や新しいスループットピークの到達時には、必要なストレージパーティションの割り当てに時間がかかる場合があります。オンデマンドキャパシティモードおよびプロビジョンドキャパシティモードのテーブルで、スループットの急増に対応できる十分なストレージパーティションを確保するために、新規または既存のテーブルを事前ウォーミングすることができます。

テーブルのピーク時のキャパシティが 30 分以内に前回のピークの 2 倍に急増すると予想される場合は、予想されるピークキャパシティに合わせてテーブルを事前ウォーミングしておくことができます。

Amazon Keyspaces で既存のオンデマンドテーブルを事前ウォーミングするには、以下の手順に従ってください。新しいテーブルを事前ウォーミングするには、「the section called “オンデマンドキャパシティに備えた新規テーブルの事前ウォーミング”」を参照してください。

まず、プロビジョンドモードのアカウントとテーブルのクォータを確認し、必要に応じて調整しておきましょう。

次に、キャパシティモードを変更する際に必要な待機期間を確認します。テーブルがオンデマンドモードに戻るまで、プロビジョニングされたキャパシティのコストが発生する点に注意してください。

Console

オンデマンドモードの既存のテーブルを事前ウォーミングする方法

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. 使用するテーブルを選択してから、[キャパシティ] タブを選択します。
3. [キャパシティ設定] セクションで、[編集] を選択します。
4. [キャパシティモード] で、テーブルを [プロビジョンド] キャパシティモードに変更します。
5. [読み込みキャパシティ] セクションで、[自動スケーリング] の選択を解除します。

   テーブルの [プロビジョニングされたキャパシティユニット] を、予想されるピーク値に設定します。
6. [書き込みキャパシティ] セクションで、前の手順で定義した読み取りキャパシティと同じ設定を選択するか、またはキャパシティの値を手動で設定します。
7. プロビジョニングされたキャパシティの設定が定義されたら、[保存] を選択します。変更内容を保存した後、キャパシティがプロビジョニングされるまで、テーブルのステータスは [更新中] と表示されます。テーブルが大きい場合、データをパーティション間で分割する必要があるため、事前ウォーミンに時間がかかる場合があります。その間も、テーブルに引き続きアクセスでき、以前に設定したピークキャパシティを期待どおりに利用できます。
8. テーブルのステータスがアクティブに変わったら、テーブルをオンデマンドキャパシティモードに戻すことができます。

Cassandra Query Language (CQL)

CQL を使用してオンデマンドモード用に既存のテーブルを事前ウォーミングする

1. テーブルのキャパシティモードを PROVIOSIONED に変更し、予想されるピーク値に基づいて読み取りキャパシティと書き込みキャパシティを設定します。

```
ALTER TABLE catalog.book_awards WITH CUSTOM_PROPERTIES={'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 18000,
 'write_capacity_units': 6000}};
```

2. テーブルがアクティブであることを確認します。次のステートメントは、その一例です。

```
SELECT * from system_schema_mcs.tables where keyspace_name = 'catalog' and
 table_name = 'book_awards';
```

3. テーブルのステータスが ACTIVE の場合、次のステートメントを使用してスループットモードを PAY_PER_REQUEST に設定し、テーブルのキャパシティモードをオンデマンドモードに変更できます。以下のステートメントは、この例です。

```
ALTER TABLE catalog.book_awards WITH CUSTOM_PROPERTIES={'capacity_mode':
{'throughput_mode': 'PAY_PER_REQUEST'}};
```

4. 次のステートメントを使用して、テーブルがオンデマンドモードになったことを確認し、テーブルのステータスを確認できます。

```
SELECT * from system_schema_mcs.tables where keyspace_name = 'catalog' and
 table_name = 'book_awards';
```

CLI

を使用してオンデマンドモードで既存のテーブルを事前ウォーミングする AWS CLI

1. テーブルのキャパシティモードを PROVIOSIONED に変更し、予想されるピーク値に基づいて読み取りキャパシティと書き込みキャパシティを設定します。次のコマンドは、その一例です。

```
aws keyspaces update-table --keyspace-name catalog --table-name book_awards
```

```
                                        \--capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=18000,writeCapacityUnits=6000
```

2. テーブルのステータスがアクティブであり、キャパシティがプロビジョニングされていることを確認します。次のステートメントを使用できます。

```
aws keyspaces get-table --keyspace-name catalog --table-name book_awards
```

3. テーブルのステータスが ACTIVE になっていて、キャパシティがプロビジョニングされている場合は、次のステートメントを使用してスループットモードを PAY_PER_REQUEST に設定し、テーブルのキャパシティモードをオンデマンドモードに変更できます。以下のステートメントは、この例です。

```
aws keyspaces update-table --keyspace-name catalog --table-name book_awards
                                        \--capacity-specification
 throughputMode=PAY_PER_REQUEST
```

4. 次のステートメントを使用して、テーブルがオンデマンドモードになったことを確認し、テーブルのステータスを確認できます。

```
aws keyspaces get-table --keyspace-name catalog --table-name book_awards
```

テーブルがオンデマンドキャパシティモードでアクティブになったら、以前のプロビジョンドキャパシティモードと同様のスループットキャパシティを処理できるようになっています。

Amazon Keyspaces 自動スケーリングでスループットキャパシティを自動的に管理する

多くのデータベースワークロードは本質的に循環的なものであり、そうでない場合は前もって予測することは困難です。例えば、日中の時間帯に大部分のユーザーがアクティブなソーシャルネットワーキングアプリがあるとします。データベースは日中のアクティビティを処理できる必要がありますが、夜間のスループットに同じレベルは必要ありません。

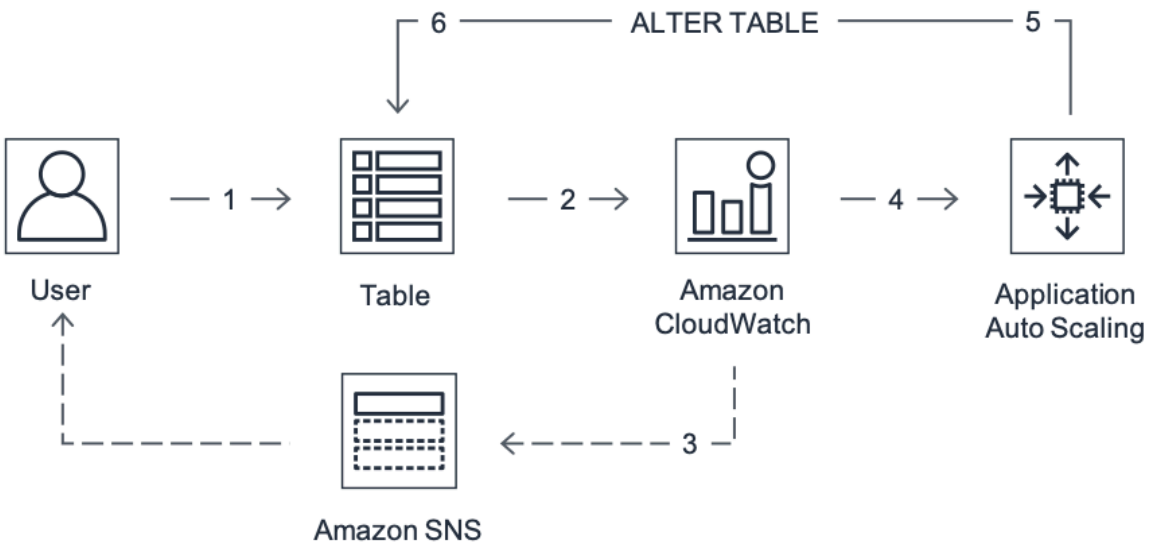
別の例としては、急速に導入された新しいモバイルゲームアプリが挙げられます。ゲームの人気があまりに高まると、利用可能なデータベースリソースを超過し、パフォーマンスが低下して顧客が不満を感じるようになります。この種のワークロードでは多くの場合、手動介入によってデータベースリソースを使用レベルに応じて上下させる必要があります。

Amazon Keyspaces (Apache Cassandra 向け) は、実際のアプリケーショントラフィックに応じてスループットキャパシティを自動的に調整するので、可変ワークロードに対するスループットキャパシティの効率的なプロビジョニングに役立ちます。Amazon Keyspaces では、Applicatino Auto Scaling サービスを使用して、ユーザーに代わってテーブルの読み取りキャパシティと書き込みキャパシティの増減を行います。アプリケーションオートスケーリングの詳細については、「Application Auto Scaling User Guide」(アプリケーションオートスケーリングユーザーガイド) を参照してください。

> **Note**
>
> Amazon Keyspaces のオートスケーリングの簡単な使用方法については「the section called "自動スケーリングポリシーの設定と更新"」を参照してください。

Amazon Keyspaces オートスケーリングの仕組み

次の図は、Amazon Keyspaces オートスケーリングによるテーブルのスループットキャパシティ管理の管理方法について、高レベルの概要を示しています。



テーブルのオートスケーリングを有効にするには、スケーリングポリシーを作成します。スケーリングポリシーは、テーブルの読み込みキャパシティと書き込みキャパシティ (またはそのいずれか) およびプロビジョンドキャパシティユニット設定をスケーリングするかどうかを指定するものです。

スケーリングポリシーによりターゲット使用率も定義されます。ターゲット使用率は、プロビジョンドキャパシティユニットに対する消費キャパシティユニットの割合をパーセンテージで示したもので

す。オートスケーリングでは、ターゲット追跡アルゴリズムを使用して、実際のワークロードに応じてテーブルのプロビジョンドスループットを上下に調整します。これにより実際のキャパシティ使用率が、ターゲット使用率またはその付近で維持されます。

読み取りおよび書き込みキャパシティに対して、オートスケーリングのターゲット使用率の値を 20%～90% の範囲で設定できます。ターゲット使用率のデフォルト値は 70% です。トラフィックが急速に変化し、キャパシティのスケールアップをすばやく開始したい場合は、ターゲット使用率の値を低く設定できます。アプリケーショントラフィックの変化が遅く、スループットのコストを削減したい場合は、ターゲット使用率の値を高く設定することもできます。

自動スケーリングポリシーの詳細については、「Application Auto Scaling ユーザーガイド」の「Target tracking scaling policies for Application Auto Scaling」を参照してください。

スケーリングポリシーを作成すると、Amazon Keyspaces が自動的に Amazon CloudWatch アラームのペアを 2 つ作成します。各ペアは、プロビジョンドスループット設定と消費スループット設定の上下の境界を示します。CloudWatch アラームは、テーブルの実際の使用率が一定期間ターゲット使用率を逸脱したときにトリガーされます。Amazon CloudWatch の詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

いずれかの CloudWatch アラームがトリガーされると、Amazon Simple Notification Service (Amazon SNS) により通知が送信されます (有効にしている場合)。次に、CloudWatch アラームにより アプリケーションオートスケーリングが呼び出されてスケーリングポリシーが評価されます。これにより、Amazon Keyspaces に Alter Table リクエストが発行されて、テーブルのプロビジョンドキャパシティが必要に応じて調整されます。Amazon SNS 通知の詳細については、「Setting up Amazon SNS notifications」(Amazon SNS 通知の設定) を参照してください。

Amazon Keyspaces は Alter Table リクエストを処理してテーブルのプロビジョンドスループット性能を動的に増減し、ターゲット使用率に近づけます。

> **Note**
>
> Amazon Keyspaces の自動スケーリングは、プロビジョニング済みスループットの設定を、実際のワークロードが上回った (または下回った) 状態が数分間続いた場合にのみ変更します。 ターゲットトラッキングアルゴリズムはターゲット使用率を選択した値の付近に長期に渡って維持しようとします。アクティビティの急激かつ短時間の上昇は、テーブルに組み込まれたバーストキャパシティで対応されます。

マルチリージョンテーブルに対する自動スケーリングの仕組み

プロビジョンドキャパシティモードのすべての AWS リージョン マルチリージョンテーブルで、すべてのテーブルレプリカに十分な読み取りおよび書き込みキャパシティを常に確保するには、Amazon Keyspaces の自動スケーリングを設定することをお勧めします。

プロビジョンドモードのマルチリージョンテーブルで自動スケーリングを使用する場合、テーブルレプリカ単体の自動スケーリングを無効にすることはできません。ただし、テーブルの読み取りの自動スケーリング設定はリージョンごとに調整できます。例えば、テーブルのレプリケート先のリージョンごとに、読み取りキャパシティと読み取り自動スケーリングの設定を変えることができます。

指定したリージョンのテーブルレプリカに対して設定した読み取り自動スケーリング設定は、テーブル全般の自動スケーリング設定よりも優先されます。一方で、書き込みキャパシティは、すべてのリージョンで書き込みをレプリケートするのに十分なキャパシティを確保するため、すべてのテーブルレプリカ間で同期させておく必要があります。

Amazon Keyspaces の自動スケーリングでは、各 AWS リージョン のテーブルにプロビジョニングされたキャパシティを、そのリージョンの使用実績に応じて個別に更新します。その結果、自動スケーリングが有効になっている場合は、マルチリージョンテーブルのリージョンごとに、プロビジョニング済みキャパシティが異なる可能性があります。

マルチリージョンテーブルとそのレプリカの自動スケーリング設定は、Amazon Keyspaces コンソール、API AWS CLI、または CQL を使用して設定できます。マルチリージョンテーブルの自動スケーリング設定を作成および更新する方法については、「the section called “マルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定の更新”」を参照してください。

> **Note**
>
> マルチリージョンテーブルで自動スケーリングを使用する場合は、常に Amazon Keyspaces API オペレーションを使用して自動スケーリングを設定する必要があります。Application Auto Scaling API オペレーションを直接使用して自動スケーリング設定を構成する場合、マルチリージョンテーブル AWS リージョン の を指定することはできません。そのため、サポート対象外の設定になってしまうおそれがあります。

使用に関する注意事項

Amazon Keyspaces オートスケーリングの使用を開始する前に、以下を確認する必要があります。

- Amazon Keyspaces オートスケーリングでは、スケーリングポリシーに従って、読み込みキャパシティや書き込みキャパシティを必要に応じて増加させることができます。クォータ で説明されているように、すべての Amazon Keyspaces クォータは有効です。
- Amazon Keyspaces オートスケーリングにより、プロビジョンドスループット設定を手動で変更できなくなることはありません。これらの手動調整は、スケーリングポリシーにアタッチされている既存の CloudWatch アラームに影響しません。
- コンソールを使用してプロビジョンドスループット性能でテーブルを作成する場合、Amazon Keyspaces オートスケーリングはデフォルトで有効になります。オートスケーリングの設定はいつでも変更できます。詳細については、「the section called “テーブルの Amazon Keyspaces 自動スケーリングを無効にする”」を参照してください。
- AWS CloudFormation を使用してスケーリングポリシーを作成する場合は、スタックがスタックテンプレートと同期 AWS CloudFormation するように、 からスケーリングポリシーを管理する必要があります。Amazon Keyspaces からスケーリングポリシーを変更すると、 AWS CloudFormation スタックがリセットされると、スタックテンプレートの元の値で上書きされます。
- CloudTrail を使用して Amazon Keyspaces のオートスケーリングを監視する場合、設定検証プロセスの一部として、アプリケーションオートスケーリングによって行われたコールのアラートが表示されることがあります。検証チェックのための `application-autoscaling.amazonaws.com` が含まれている `invokedBy` フィールドを使用すれば、これらのアラートを除外できます。

Amazon Keyspaces の自動スケーリングポリシーを設定および更新する

コンソール、CQL、または AWS Command Line Interface （ AWS CLI) を使用して、新規および既存のテーブルの Amazon Keyspaces 自動スケーリングを設定できます。自動スケーリングの設定変更や無効化を行うこともできます。

スケールインおよびスケールアウトのクールダウン時間の設定などのより高度な機能については、CQL または AWS CLI を使用して Amazon Keyspaces スケーリングポリシーを管理することをお勧めします。

トピック

- Amazon Keyspaces の自動スケーリング用のアクセス許可を設定する
- 自動スケーリングを有効にして新規テーブルを作成する
- 既存のテーブルに対して自動スケーリングを設定する

- テーブルの Amazon Keyspaces 自動スケーリングの設定を表示する
- テーブルの Amazon Keyspaces 自動スケーリングを無効にする
- Amazon CloudWatch で Amazon Keyspaces テーブルの自動スケーリングアクティビティを表示する

Amazon Keyspaces の自動スケーリング用のアクセス許可を設定する

まず、自動スケーリング設定を作成し、管理するための適切なアクセス許可がプリンシパルに与えられていることを確認してください。 AWS Identity and Access Management (IAM) では、Amazon Keyspaces スケーリングポリシーを管理するために AWS マネージドポリシー`AmazonKeyspacesFullAccess`が必要です。

> **Important**
>
> テーブルの自動スケーリングを無効にするには、`application-autoscaling:*` アクセス許可が必要です。テーブルを削除するには、その前にテーブルの自動スケーリングを無効にしておく必要があります。

Amazon Keyspaces コンソールへのアクセスと Amazon Keyspaces の自動スケーリングで使用する IAM ユーザーまたはロールを設定するには、次のポリシーを追加します。

`AmazonKeyspacesFullAccess` ポリシーをアタッチするには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールダッシュボードで [ユーザー] を選択して、リストから IAM ユーザーまたはロールを選択します。
3. [Summary (概要)] ページで、[Add permissions (許可の追加)] を選択します。
4. [Attach existing policies directly (既存のポリシーを直接アタッチする)] を選択します。
5. ポリシーリストから [AmazonKeyspacesFullAccess] を選択し、次に [Next: Review (次へ: 確認)] を選択します。
6. [Add permissions (許可の追加)] を選択します。

自動スケーリングを有効にして新規テーブルを作成する

新しい Amazon Keyspaces テーブルの作成時に、テーブルの書き込みキャパシティまたは読み取りキャパシティの自動スケーリングを自動的に有効にすることができます。その場合は、Amazon Keyspaces が自動的に Application Auto Scaling と通信してテーブルをスケーラブルターゲットとして登録し、プロビジョニング済みの書き込みキャパシティまたは読み取りキャパシティを調整します。

マルチリージョンテーブルを作成し、テーブルのレプリカごとに異なる自動スケーリング設定を行う方法については、「the section called “プロビジョンドモードのマルチリージョンテーブルの作成”」を参照してください。

> **Note**
>
> Amazon Keyspaces の自動スケーリングでは、ユーザーに代わって自動スケーリングアクションを実行するサービスリンクロール (`AWSServiceRoleForApplicationAutoScaling_CassandraTable`) の存在が必要になります。このロールは自動的に作成されます。詳細については、「the section called “サービスにリンクされたロールの使用”」を参照してください。

Console

自動スケーリングが有効な新規テーブルをコンソールで作成する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで [Tables] (テーブル) を選択して、[Create table (テーブルの作成)] を選択します。
3. [Table details (テーブルの詳細)] セクションの [Create table (テーブルの作成)] ページで、キースペースを選択し、新しいテーブルに名前を付けます。
4. [列] セクションで、テーブルのスキーマを作成します。
5. [プライマリキー] セクションで、テーブルのプライマリキーを定義し、オプションでクラスタリング列を選択します。
6. [Table settings (テーブルの設定)] セクションで、[Customize settings (設定のカスタマイズ)] を選択します。
7. [Read/write capacity settings (読み取り/書き込みキャパシティの設定)] に進みます。

8. [Capacity mode (キャパシティモード)] で、[Provisioned (プロビジョン)] を選択します。
9. [Read capacity] (読み取りキャパシティ) セクションで、[Scale automatically (自動スケーリング)] が選択されているか確認します。

   このステップでは、テーブルの読み取りキャパシティユニットの最小値と最大値、およびターゲット使用率を選択します。

   - 最小キャパシティユニット — テーブルをいつでもすぐにサポートできる状態にするために、最小レベルのスループット値を入力します。この値は、1 から、アカウントの秒単位の最大スループットクォータ (デフォルトは 40,000) までの範囲でなければなりません。
   - 最大キャパシティユニット — テーブルに対してプロビジョニングするスループットの最大量を入力します。この値は、1 から、アカウントの秒単位の最大スループットクォータ (デフォルトは 40,000) までの範囲でなければなりません。
   - ターゲット使用率— ターゲット使用率を 20% ~ 90% の範囲で入力します。定義したターゲット使用率をトラフィックが上回ると、キャパシティが自動的に増加されます。定義したターゲットをトラフィックが下回ると、再び容量が自動的に減少されます。

   > **Note**
   >
   > アカウントのデフォルトクォータの詳細およびクォータを引き上げる方法については、「クォータ」を参照してください。

10. [書き込みキャパシティ] セクションで、前の手順で定義した読み取りキャパシティと同じ設定を選択するか、またはキャパシティの値を手動で設定します。
11. [Create table (テーブルの作成)] を選択します。指定した自動スケーリングパラメータを使用してテーブルが作成されます。

Cassandra Query Language (CQL)

Amazon Keyspaces の自動スケーリングが有効な新規テーブルを CQL を使用して作成する

テーブルの自動スケーリング設定をプログラムによって行うには、`AUTOSCALING_SETTINGS` ステートメントに Amazon Keyspaces 自動スケーリングのパラメータを指定します。これらのパラメータでは、テーブルのプロビジョニング済みスループットを調整するように Amazon Keyspaces に指示する条件を定義し、追加で実行するアクションがあれば、それらも任意で定義します。この例では、mytable の自動スケーリング設定を定義します。

ポリシーには、次の要素が含まれます。

- AUTOSCALING_SETTINGS – スループットキャパシティの自動調整を Amazon Keyspaces に許可するかどうかを指定します。以下の値が必須です。
  - provisioned_write_capacity_autoscaling_update:
    - minimum_units
    - maximum_units
  - provisioned_read_capacity_autoscaling_update:
    - minimum_units
    - maximum_units
  - scaling_policy – Amazon Keyspaces はターゲット追跡ポリシーに対応しています。ターゲット追跡ポリシーを定義するには、次のパラメータを設定します。
    - target_value — Amazon Keyspaces の自動スケーリングでは、プロビジョニング済みのキャパシティに占める実際のキャパシティ消費の割合が、この値と同じか近い値で維持されます。target_value をパーセンテージとして定義します。
    - disableScaleIn: (オプション) テーブルに対して scale-in が無効か有効かを指定する boolean。このパラメータは、デフォルトでは無効になっています。scale-in を有効にするには、boolean 値を FALSE に設定します。その場合は、テーブルのキャパシティが自動的にスケールダウンされます。
    - scale_out_cooldown — スケールアウトアクティビティでテーブルのプロビジョンドスループットを増加させます。スケールアウトアクティビティのクールダウン期間を追加するには、scale_out_cooldown の値を秒単位で指定します。値を指定しない場合、デフォルト値は 0 です。ターゲット追跡とクールダウン期間の詳細については、「Application Auto Scaling ユーザーガイド」の「ターゲット追跡スケーリングポリシー」を参照してください。
    - scale_in_cooldown — スケールインアクティビティでテーブルのプロビジョンドスループットを減少させます。スケールインアクティビティのクールダウン期間を追加するには、scale_in_cooldown の値を秒単位で指定します。値を指定しない場合、デフォルト値は 0 です。ターゲット追跡とクールダウン期間の詳細については、「Application Auto Scaling ユーザーガイド」の「ターゲット追跡スケーリングポリシー」を参照してください。

> **Note**
>
> target_value がどのように機能するかをさらに理解するために、書き込み容量単位が 200 で、プロビジョンされたスループット設定を持つテーブルがあるとします。このテーブルのスケーリングポリシーを作成することにしました。target_value は 70% です。
> ここで、実際の書き込みスループットが 150 容量単位になるように、テーブルへの書き込みトラフィックを駆動し始めたとします。消費とプロビジョンの比率は現在 (150/200)、つまり 75% です。この割合は目標値を上回っているため、自動スケーリングはプロビジョニング済みの書き込みキャパシティを 215 に増やします。これで、割合が (150/215)、つまり 69.77% になります。これは、target_value に可能な限り近く、なおかつ上回らない値です。

mytable では、読み取りキャパシティと書き込みキャパシティの両方の TargetValue を 50% に設定します。Amazon Keyspaces の自動スケーリングは、テーブルのプロビジョニング済みスループットを 5～10 キャパシティユニットの範囲内で調整し、プロビジョニング済みキャパシティに占める実際のキャパシティ消費の割合が 50%、または 50% に近くなるようにします。読み取りキャパシティについては、ScaleOutCooldown と ScaleInCooldown の値を 60 秒に設定します。

次のステートメントを使用して、自動スケーリングを有効にした新しい Amazon Keyspaces テーブルを作成できます。

```
CREATE TABLE mykeyspace.mytable(pk int, ck int, PRIMARY KEY (pk, ck))
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 1,
        'write_capacity_units': 1
    }
} AND AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50
            }
        }
```

```
    },
    'provisioned_read_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50,
                'scale_in_cooldown': 60,
                'scale_out_cooldown': 60
            }
        }
    }
};
```

CLI

を使用して Amazon Keyspaces 自動スケーリングで新しいテーブルを作成する AWS CLI

テーブルの自動スケーリング設定をプログラムによって行うには、`autoScalingSpecification` アクションに Amazon Keyspaces 自動スケーリングのパラメータを指定します。これらのパラメータでは、テーブルのプロビジョニング済みスループットを調整するように Amazon Keyspaces に指示する条件を定義し、追加で実行するアクションがあれば、それらも任意で定義します。この例では、mytable の自動スケーリング設定を定義します。

ポリシーには、次の要素が含まれます。

- `autoScalingSpecification` – キャパシティスループットの自動調整を Amazon Keyspaces に許可するかどうかを指定します。読み取りキャパシティと書き込みキャパシティに対して別々に自動スケーリングを有効にすることができます。`autoScalingSpecification` の次のパラメータを指定する必要があります。
  - `writeCapacityAutoScaling` – 最大および最小の書き込みキャパシティユニット。
  - `readCapacityAutoScaling` – 最大および最小の読み取りキャパシティユニット。
  - `scalingPolicy` – Amazon Keyspaces はターゲット追跡ポリシーに対応しています。ターゲット追跡ポリシーを定義するには、次のパラメータを設定します。
    - `targetValue` — Amazon Keyspaces の自動スケーリングでは、プロビジョニング済みのキャパシティに占める実際のキャパシティ消費の割合が、この値と同じか近い値で維持されます。`targetValue` をパーセンテージとして定義します。

- disableScaleIn: (オプション) テーブルに対して scale-in が無効か有効かを指定する boolean。このパラメータは、デフォルトでは無効になっています。scale-in を有効にするには、boolean 値を FALSE に設定します。その場合は、テーブルのキャパシティが自動的にスケールダウンされます。
- scaleOutCooldown — スケールアウトアクティビティでテーブルのプロビジョンドスループットを増加させます。スケールアウトアクティビティのクールダウン期間を追加するには、ScaleOutCooldown の値を秒単位で指定します。デフォルト値は 0 です。ターゲット追跡とクールダウン期間の詳細については、「Application Auto Scaling ユーザーガイド」の「ターゲット追跡スケーリングポリシー」を参照してください。
- scaleInCooldown — スケールインアクティビティでテーブルのプロビジョンドスループットを減少させます。スケールインアクティビティのクールダウン期間を追加するには、ScaleInCooldown の値を秒単位で指定します。デフォルト値は 0 です。ターゲット追跡とクールダウン期間の詳細については、「Application Auto Scaling ユーザーガイド」の「ターゲット追跡スケーリングポリシー」を参照してください。

> **Note**
>
> TargetValue がどのように機能するかをさらに理解するために、書き込み容量単位が 200 で、プロビジョンされたスループット設定を持つテーブルがあるとします。このテーブルのスケーリングポリシーを作成することにしました。TargetValue は 70% です。ここで、実際の書き込みスループットが 150 容量単位になるように、テーブルへの書き込みトラフィックを駆動し始めたとします。消費とプロビジョンの比率は現在 (150/200)、つまり 75% です。この割合は目標値を上回っているため、自動スケーリングはプロビジョニング済みの書き込みキャパシティを 215 に増やします。これで、割合が (150/215)、つまり 69.77% になります。これは、TargetValue に可能な限り近く、なおかつ上回らない値です。

mytable では、読み取りキャパシティと書き込みキャパシティの両方の TargetValue を 50% に設定します。Amazon Keyspaces の自動スケーリングは、テーブルのプロビジョニング済みスループットを 5 ~ 10 キャパシティユニットの範囲内で調整し、プロビジョニング済みキャパシティに占める実際のキャパシティ消費の割合が 50%、または 50% に近くなるようにします。読み取りキャパシティについては、ScaleOutCooldown と ScaleInCooldown の値を 60 秒に設定します。

テーブルの作成時に複雑な自動スケーリング設定を行う場合は、JSON ファイルから自動スケーリング設定を読み込むと便利です。次の例を使用するには、サンプル JSON ファイルを収録した auto-scaling.zip をダウンロードし、auto-scaling.json を抽出します。ファイルのパスを書き留めておいてください。この例では、JSON ファイルは現在のディレクトリにあります。さまざまなファイルパスオプションについては、「ファイルからパラメーターを読み込む方法」を参照してください。

```
aws keyspaces create-table --keyspace-name mykeyspace --table-name mytable
            \ --schema-definition 'allColumns=[{name=pk,type=int},
{name=ck,type=int}],partitionKeys=[{name=pk},{name=ck}]'
            \ --capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1
            \ --auto-scaling-specification file://auto-scaling.json
```

既存のテーブルに対して自動スケーリングを設定する

既存の Amazon Keyspaces テーブルを更新して、そのテーブルの書き込みキャパシティまたは読み取りキャパシティに対して自動スケーリングを有効にすることができます。現時点でオンデマンドキャパシティモードのテーブルを更新する場合は、まずテーブルのキャパシティモードをプロビジョンドキャパシティモードに変更する必要があります。

マルチリージョンテーブルの自動スケーリング設定を更新する方法については、「the section called “マルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定の更新”」を参照してください。

Amazon Keyspaces の自動スケーリングでは、ユーザーに代わって自動スケーリングアクションを実行するサービスリンクロール (AWSServiceRoleForApplicationAutoScaling_CassandraTable) の存在が必要になります。このロールは自動的に作成されます。詳細については、「the section called “サービスにリンクされたロールの使用”」を参照してください。

Console

既存のテーブルに対して Amazon Keyspaces の自動スケーリングを設定する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. 使用するテーブルを選択してから、[キャパシティ] タブを選択します。
3. [キャパシティ設定] セクションで、[編集] を選択します。

4. [キャパシティモード] で、テーブルが [プロビジョンド] キャパシティモードを使用していることを確認します。
5. [Scale automatically (自動スケーリング)] を選択し、the section called “自動スケーリングを有効にして新規テーブルを作成する” のステップ 6 を参照して読み取り/書き込みキャパシティを編集します。
6. 自動スケーリング設定が定義されたら、[Save (保存)] を選択します。

Cassandra Query Language (CQL)

CQL を使用して既存のテーブルに Amazon Keyspaces 自動スケーリングを設定する

既存の Amazon Keyspaces テーブルに対して ALTER TABLE ステートメントを使用して、そのテーブルの書き込みキャパシティまたは読み取りキャパシティの自動スケーリングを設定できます。現時点でオンデマンドキャパシティモードのテーブルを更新する場合は、capacity_mode を provisioned (プロビジョンド) に設定する必要があります。テーブルが既にプロビジョンドキャパシティモードになっている場合、このフィールドは省略できます。

次の例のステートメントは、オンデマンドキャパシティモードのテーブル mytable を更新します。このステートメントは、テーブルのキャパシティモードを、自動スケーリングが有効なプロビジョンドモードに変更します。

書き込みキャパシティは、5～10 のキャパシティユニット範囲内で、50% の目標値が設定されています。読み取りキャパシティも、5～10 のキャパシティユニット範囲内で、50% の目標値が設定されています。読み取りキャパシティについては、scale_out_cooldown と scale_in_cooldown の値を 60 秒に設定します。

```
ALTER TABLE mykeyspace.mytable
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 1,
        'write_capacity_units': 1
    }
} AND AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
```

```
                'target_value': 50
            }
        }
    },
    'provisioned_read_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50,
                'scale_in_cooldown': 60,
                'scale_out_cooldown': 60
            }
        }
    }
};
```

CLI

を使用して Amazon Keyspaces 自動スケーリングで既存のテーブルを設定する AWS CLI

既存の Amazon Keyspaces テーブルに対して UpdateTable オペレーションを使用して、そのテーブルの書き込みキャパシティまたは読み取りキャパシティに対して自動スケーリングを有効にすることができます。

次のコマンドを使用して、既存のテーブルに対して Amazon Keyspaces の自動スケーリングを有効にすることができます。テーブルの自動スケーリング設定は JSON ファイルから読み込まれます。次の例を使用するには、サンプル JSON ファイルを収録した auto-scaling.zip をダウンロードし、auto-scaling.json を抽出します。ファイルのパスを書き留めておいてください。この例では、JSON ファイルは現在のディレクトリにあります。さまざまなファイルパスオプションについては、「ファイルからパラメーターを読み込む方法」を参照してください。

次の例で使用されている自動スケーリング設定の詳細については、「the section called “自動スケーリングを有効にして新規テーブルを作成する”」を参照してください。

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable
            \ --capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1
            \ --auto-scaling-specification file://auto-scaling.json
```

テーブルの Amazon Keyspaces 自動スケーリングの設定を表示する

コンソール、CQL、または を使用して、テーブルの Amazon Keyspaces 自動スケーリング設定 AWS CLI を表示および更新できます。

Console

コンソールを使用して自動スケーリング設定を表示する

1. 表示対象のテーブルを選択し、[キャパシティ] タブに移動します。
2. [キャパシティ設定] セクションで、[編集] を選択します。[読み込みキャパシティ] セクションまたは [書き込みキャパシティ] セクションで設定を変更できます。これらの設定の詳細については、「the section called “自動スケーリングを有効にして新規テーブルを作成する”」をご参照ください。

Cassandra Query Language (CQL)

CQL を使用してテーブルの Amazon Keyspaces 自動スケーリングポリシーを表示する

テーブルの自動スケーリング設定の詳細を表示するには、次のコマンドを使用します。

```
SELECT * FROM system_schema_mcs.autoscaling WHERE keyspace_name = 'mykeyspace' AND
 table_name = 'mytable';
```

このコマンドの出力は次のようになります。

```
 keyspace_name | table_name | provisioned_read_capacity_autoscaling_update

                                                                                |
 provisioned_write_capacity_autoscaling_update
---------------+------------
+-------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------------------
 mykeyspace    | mytable    | {'minimum_units': 5, 'maximum_units':
 10, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
 {'scale_out_cooldown': 60, 'disable_scale_in': false, 'target_value':
 50, 'scale_in_cooldown': 60}}} | {'minimum_units': 5, 'maximum_units':
 10, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
 {'scale_out_cooldown': 0, 'disable_scale_in': false, 'target_value': 50,
 'scale_in_cooldown': 0}}}
```

CLI

を使用してテーブルの Amazon Keyspaces 自動スケーリングポリシーを表示する AWS CLI

テーブルの自動スケーリング設定を表示するには、`get-table-auto-scaling-settings` オペレーションを使用します。次の CLI コマンドは一例です。

```
aws keyspaces get-table-auto-scaling-settings --keyspace-name mykeyspace --table-
name mytable
```

このコマンドの出力は次のようになります。

```
{
    "keyspaceName": "mykeyspace",
    "tableName": "mytable",
    "resourceArn": "arn:aws:cassandra:us-east-1:5555-5555-5555:/keyspace/mykeyspace/
table/mytable",
    "autoScalingSpecification": {
        "writeCapacityAutoScaling": {
            "autoScalingDisabled": false,
            "minimumUnits": 5,
            "maximumUnits": 10,
            "scalingPolicy": {
                "targetTrackingScalingPolicyConfiguration": {
                    "disableScaleIn": false,
                    "scaleInCooldown": 0,
                    "scaleOutCooldown": 0,
                    "targetValue": 50.0
                }
            }
        },
        "readCapacityAutoScaling": {
            "autoScalingDisabled": false,
            "minimumUnits": 5,
            "maximumUnits": 10,
            "scalingPolicy": {
                "targetTrackingScalingPolicyConfiguration": {
                    "disableScaleIn": false,
                    "scaleInCooldown": 60,
                    "scaleOutCooldown": 60,
                    "targetValue": 50.0
                }
            }
```

```
        }
    }
}
```

テーブルの Amazon Keyspaces 自動スケーリングを無効にする

テーブルの Amazon Keyspaces 自動スケーリングはいつでも無効にすることができます。テーブルの読み取りキャパシティまたは書き込みキャパシティのスケーリングが不要になった場合は、それ以降は Amazon Keyspaces でテーブルの読み取り/書き込みキャパシティの設定が変更されないように、自動スケーリングを無効にすることを検討してください。コンソール、CQL、または AWS CLI を使用して、テーブルを更新することができます。

自動スケーリングを無効にすると、自動的に作成された CloudWatch アラームも削除されます。

Amazon Keyspaces テーブルにアクセスのために Application Auto Scaling により使用されるサービスリンクロールを削除するには、the section called “Amazon Keyspaces のサービスリンクロールの削除” のステップを実行します。

> **Note**
>
> Application Auto Scaling で使用されているサービスリンクロールを削除するには、すべての AWS リージョンにおいてアカウント内のすべてのテーブルで自動スケーリングを無効にする必要があります。

Console

コンソールを使用してテーブルの Amazon Keyspaces 自動スケーリングを無効にする

Amazon Keyspaces コンソールの使用

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. 更新対象のテーブルを選択し、[キャパシティ] タブに移動します。
3. [キャパシティ設定] セクションで、[編集] を選択します。
4. Amazon Keyspaces の自動スケーリングを無効にするには、[自動スケーリング] チェックボックスの選択を解除します。自動スケーリングを無効にすると、Application Auto Scaling のスケーラブルターゲットとしてのテーブルの登録が解除されます。

Cassandra Query Language (CQL)

CQL を使用してテーブルの Amazon Keyspaces 自動スケーリングを無効にする

次のステートメントは、テーブル mytable の書き込みキャパシティの自動スケーリングを無効にします。

```
ALTER TABLE mykeyspace.mytable
WITH AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'autoscaling_disabled': true
    }
};
```

CLI

を使用してテーブルの Amazon Keyspaces 自動スケーリングをオフにする AWS CLI

次のコマンドは、テーブルの読み取りキャパシティの自動スケーリングを無効にします。また、自動的に作成された CloudWatch アラームも削除されます。

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable
            \ --auto-scaling-specification
 readCapacityAutoScaling={autoScalingDisabled=true}
```

Amazon CloudWatch で Amazon Keyspaces テーブルの自動スケーリングアクティビティを表示する

使用状況とパフォーマンスに関するメトリクスを生成する Amazon CloudWatch を使用すれば、Amazon Keyspaces 自動スケーリングによるリソースの使用状況をモニタリングできます。『Application Auto Scaling ユーザーガイド』のステップに従って CloudWatch ダッシュボードを作成します。

Amazon Keyspaces でバーストキャパシティを効果的に使用する

Amazon Keyspaces では、バーストキャパシティを指定することで、パーティションごとのスループットプロビジョニングに柔軟性を与えることができます。パーティションのスループットが十分に利用されていない場合、Amazon Keyspaces では、スループットの利用率がバーストした場合に備えて、未使用のキャパシティーの一部を確保して使用量のスパイクに対応します。

現在 Amazon Keyspaces では、未使用の読み取りキャパシティーと書き込みキャパシティーは、最大 5 分 (300 秒) 確保されます。読み込みアクティビティや書き込みアクティビティが時々バーストする状況では、これらの余分な容量ユニットをすばやく消費して対応できます。これは、テーブルに定義した 1 秒あたりのプロビジョン済みスループットキャパシティよりも高速です。

また、Amazon Keyspaces では、バックグラウンドメンテナンスやその他のタスクのために予告なしにバーストキャパシティーが消費される場合があります。

これらのバーストキャパシティの詳細は将来変更される可能性があります。

Amazon Keyspaces (Apache Cassandra 向け) の機能を使用する

この章では、バックアップと復元、有効期限、マルチリージョンレプリケーションなど、Amazon Keyspaces とさまざまなデータベース機能の使用について詳しく説明します。

- 有効期限 (TTL) – Amazon Keyspaces は、設定した TTL 値に基づいて、テーブルのデータを自動的に期限切れにします。TTL の設定方法と、テーブルでの活用方法について説明します。
- PITR – テーブルのデータのバックアップを継続的に作成して、Amazon Keyspaces テーブルを誤った書き込みや削除操作から保護します。テーブルで PITR を設定する方法、特定の時点の状態にテーブルを復元する方法、誤って削除されたテーブルを復元する方法を説明します。
- マルチリージョンテーブルの使用 – Amazon Keyspaces のマルチリージョンテーブルには、オンデマンドキャパシティモードまたは自動スケーリングが有効なプロビジョンドキャパシティモードのいずれかで、書き込みスループットキャパシティを設定する必要があります。各リージョンに必要な書き込みキャパシティユニット (WCU) を推定してスループットキャパシティのニーズを予測し、書き込みを十分にレプリケートできる分のキャパシティを確保するため、すべてのリージョンの書き込みを合計した分量をプロビジョニングします。
- Amazon Keyspaces 変更データキャプチャ (CDC) – Amazon Keyspaces CDC ストリームは、テーブルからの行レベルの変更イベントをほぼリアルタイムで記録します。Kinesis Client Library (KCL) を使用して Amazon Keyspaces CDC ストリームからデータを使用および処理する方法について説明します。
- クエリとページ分割 – Amazon Keyspaces は、`SELECT` ステートメントでの `IN` 演算子の使用、`ORDER BY` による結果の並べ替え、大きな結果セットの自動ページ分割など、高度なクエリ機能に対応しています。このセクションでは、Amazon Keyspaces がこれらのクエリをどのように処理するかを説明し、例を紹介します。
- パーティショナー – Amazon Keyspaces には、`Murmur3Partitioner` (デフォルト)、`RandomPartitioner`、`DefaultPartitioner` の 3 つのパーティショナーが用意されています。 AWS Management Console または Cassandra クエリ言語 (CQL) を使用して、アカウントレベルでリージョンごとにパーティショナーを変更できます。
- クライアント側のタイムスタンプ – クライアント側のタイムスタンプは Cassandra 互換のタイムスタンプで、Amazon Keyspaces ではテーブルの各セルに保存されます。クライアント側のタイムスタンプは競合の解決や、クライアントアプリケーションが書き込みの順序を判断する上で役立ちます。

- ユーザー定義型 (UDTs) – UDTsを使用すると、実際のデータ階層を表すアプリケーション内のデータ構造を定義できます。
- リソースのタグ付け — タグを使用して、キースペースやテーブルなどの Amazon Keyspaces リソースにラベルを付けることができます。タグは、リソースの分類、コスト追跡の有効化、タグに基づくアクセスコントロールの設定に役立ちます。このセクションでは、Amazon Keyspaces におけるタグ付けの制限、オペレーション、ベストプラクティスについて説明します。
- AWS CloudFormation テンプレート – Amazon Keyspaces のキースペースとテーブルを AWS CloudFormation モデル化してセットアップできるため、リソースとインフラストラクチャの作成と管理に費やす時間を短縮できます。

トピック

- Amazon Keyspaces のシステムキースペース
- Amazon Keyspaces のユーザー定義タイプ (UDTs)
- Amazon Keyspaces で CQL クエリを使用する
- Amazon Keyspaces での変更データキャプチャ (CDC) ストリームの使用
- Amazon Keyspaces でのパーティショナーの操作
- Amazon Keyspaces でのクライアント側のタイムスタンプ
- Amazon Keyspaces のマルチリージョンレプリケーション (Apache Cassandra 用)
- Amazon Keyspaces のポイントインタイムリカバリでデータをバックアップおよび復元する
- Amazon Keyspaces (Apache Cassandra 向け) で有効期限 (TTL) を使用してデータを期限切れにする
- AWS SDK でのこのサービスの使用
- Amazon Keyspaces リソースのタグとラベルを操作する
- を使用して Amazon Keyspaces リソースを作成する AWS CloudFormation
- Amazon Keyspaces (Apache Cassandra 向け) での NoSQL Workbench の使用

Amazon Keyspaces のシステムキースペース

このセクションでは、Amazon Keyspaces (Apache Cassandra 向け) でのシステムキースペースの操作について詳しく説明します。

Amazon Keyspaces は次の 4 つのシステムキースペースを使用します。

- `system`

- system_schema
- system_schema_mcs
- system_multiregion_info

以降のセクションでは、Amazon Keyspaces でサポートされているシステムキースペースとシステムテーブルについて詳しく説明します。

system

これは Cassandra キースペースです。Amazon Keyspaces は以下のテーブルを使用します。

テーブル名	列名	コメント
local	key, bootstrapped, broadcast_address, cluster_name, cql_version, data_center, gossip_generation, host_id, listen_address, native_protocol_version, partitioner, rack, release_version, rpc_address, schema_version, thrift_version, tokens, truncated_at	ローカルキースペースに関する情報。
peers	peer, data_center, host_id, preferred_ip, rack, release_version, rpc_address, schema_version, tokens	このテーブルをクエリして、使用可能なエンドポイントを確認してください。例えば、パブリックエンドポイント経由で接続している場合、使用可能な 9 つの IP アドレスのリストが表示されます。FIPS エンドポイント経由で接続し

テーブル名	列名	コメント
		ている場合は、3 つの IP アドレスのリストが表示されます。 AWS PrivateLink VPC エンドポイント経由で接続している場合は、設定した IP アドレスのリストが表示されます。詳細については、「the section called “インターフェイス VPC エンドポイント情報を含む system.peers テーブルエントリの入力”」を参照してください。
size_estimates	keyspace_name, table_name, range_start, range_end, mean_partition_size, partitions_count	このテーブルは、各テーブルのトークン範囲ごとのパーティションの合計サイズと数を定義しています。これは、推定パーティションサイズを使用して処理を分散する Apache Cassandra Spark コネクタに必要です。
prepared_statements	prepared_id, logged_keyspace, query_string	このテーブルには、保存されたクエリに関する情報が含まれています。

system_schema

これは Cassandra キースペースです。Amazon Keyspaces は以下のテーブルを使用します。

テーブル名	列名	コメント
keyspaces	keyspace_name, durable_writes, replication	特定のキースペースに関する情報。

テーブル名	列名	コメント
tables	keyspace_name, table_name, bloom_filter_fp_chance, caching, comment, compaction, compression, crc_check_chance, dclocal_read_repair_chance, default_time_to_live, extensions, flags, gc_grace_seconds, id, max_index_interval, memtable_flush_period_in_ms, min_index_interval, read_repair_chance, speculative_retry	特定のトリガーに関する情報です。
types	keyspace_name, type_name, field_names, field_types	特定のユーザー定義タイプ (UDT) に関する情報。
columns	keyspace_name, table_name, column_name, clustering_order, column_name_bytes, kind, position, type	特定の列に関する情報。

system_schema_mcs

これは、 AWS または Amazon Keyspaces 固有の設定に関する情報を保存する Amazon Keyspaces キースペースです。

テーブル名	列名	コメント
keyspaces	keyspace_name, durable_writes, replication	このテーブルをクエリして、キースペースが作成されたかどうかをプログラムで調べます。詳細については、「the section called “キースペースの作成ステータスの確認”」を参照してください。
tables	keyspace_name, creation_time, speculative_retry, cdc, gc_grace_seconds, crc_check_chance, min_index_interval, bloom_filter_fp_chance, flags, custom_properties, dclocal_read_repair_chance, table_name, caching, default_time_to_live, read_repair_chance, max_index_interval, extensions, compaction, comment, id, compression, memtable_flush_period_in_ms, cdc_specification, latest_stream_arn status	このテーブルをクエリして、特定のテーブルのステータスを調べます。詳細については、「the section called “テーブルの作成ステータスの確認”」を参照してください。 このテーブルをクエリして、custom_properties として保存されている Amazon Keyspaces 固有の設定を一覧表示することもできます。例: • capacity_mode • cdc_specification • client_side_timestamps • encryption_specification • point_in_time_recovery • ttl
tables_history	keyspace_name, table_name, event_tim	このテーブルをクエリすると、特定のテーブルのスキー

テーブル名	列名	コメント
	e, creation_time, custom_properties, event	マ変更について調べることができます。
columns	keyspace_name, table_name, column_name, clustering_order, column_name_bytes, kind, position, type	このテーブルは、system_schema キースペースの Cassandra テーブルと同じです。
tags	resource_id, keyspace_name, resource_name, resource_type, tags	キースペースのタグの有無を調べるには、このテーブルをクエリしてください。詳細については、「the section called “テーブルのタグの表示”」を参照してください。
types	keyspace_name, type_name, field_names, field_types, max_nesting_depth, last_modified_timestamp, status, direct_referring_tables, direct_parent_types	このテーブルをクエリして、ユーザー定義型 (UDTsに関する情報を確認します。たとえば、このテーブルをクエリして、特定のキースペースのすべての UDTs を一覧表示できます。詳細については、「the section called “ユーザー定義タイプ (UDTs)”」を参照してください。

テーブル名	列名	コメント
autoscaling	keyspace_name, table_name, provisioned_read_capacity_autoscaling_update, provisioned_write_capacity_autoscaling_update	このテーブルをクエリして、プロビジョンドモードのテーブルの自動スケーリング設定を取得します。これらの設定は、テーブルがアクティブになるまでは使用できません。このテーブルをクエリするには、WHERE 句に keyspace_name および table_name を指定する必要があります。詳細については、「the section called “テーブルの Amazon Keyspaces 自動スケーリングの設定を表示する”」を参照してください。

system_multiregion_info

これは、マルチリージョンレプリケーションに関する情報を保存する Amazon Keyspaces キースペースです。

テーブル名	列名	コメント
tables	keyspace_name, table_name, region, status	このテーブルには、テーブル AWS リージョン がレプリケートされる やテーブルのステータスなど、マルチリージョンテーブルに関する情報が含まれています。このテーブルをクエリして、custom_properties として保存されている Amazon Keyspaces 固有の設定を一覧表示することもできます。例:

テーブル名	列名	コメント
		• capacity_mode このテーブルをクエリするには、WHERE 句に keyspace_name および table_name を指定する必要があります。詳細については、「the section called “マルチリージョンキースペースの作成”」を参照してください。
keyspaces	keyspace_name, region, status, tables_replication_progress	このテーブルには、キースペースにレプリカを追加するALTER KEYSPACEオペレーションの進行状況に関する情報が含まれています。たとえば、新しいリージョンで作成済みのテーブルの数、進行中のテーブルの数などです。例については、「」を参照してくださいthe section called “レプリケーションの進行状況を確認する”。

テーブル名	列名	コメント
autoscaling	keyspace_name, table_name, provisioned_read_capacity_autoscaling_update, provisioned_write_capacity_autoscaling_update, region	このテーブルをクエリして、プロビジョンドモードのマルチリージョンテーブルの自動スケーリング設定を取得します。これらの設定は、テーブルがアクティブになるまでは使用できません。このテーブルをクエリするには、WHERE 句に keyspace_name および table_name を指定する必要があります。詳細については、「the section called “マルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定の更新”」を参照してください。
types	keyspace_name, type_name, field_names, field_types, max_nesting_depth, last_modified_timestamp, status, direct_referring_tables, direct_parent_types, region	このテーブルをクエリして、マルチリージョンキースペースのユーザー定義タイプ (UDTs) に関する情報を確認します。たとえば、このテーブルをクエリして、特定のキースペースに UDTs を使用するすべてのテーブルレプリカとそれぞれの AWS リージョンを一覧表示できます。詳細については、「the section called “ユーザー定義タイプ (UDTs)”」を参照してください。

Amazon Keyspaces のユーザー定義タイプ (UDTs)

ユーザー定義型 (UDT) は、Amazon Keyspaces で 1 つの列を定義するために使用できるフィールドとデータ型のグループです。UDTs の有効なデータ型は、同じキースペースで既に作成したコレクションやその他の UDTs を含む、サポートされているすべての Cassandra データ型です。サポートされている Cassandra データ型の詳細については、「」を参照してくださいthe section called “Cassandra データ型サポート”。

Amazon Keyspaces でユーザー定義タイプ (UDTs) を使用して、データをより効率的に整理できます。たとえば、ネストされたコレクションを使用して UDTs を作成できます。これにより、より複雑なデータモデリングをアプリケーションに実装できます。UDTs の定義には、フリーズキーワードを使用することもできます。

UDTs はキースペースにバインドされ、同じキースペース内のすべてのテーブルと UDTsで使用できます。UDTs は、単一リージョンおよびマルチリージョンのキースペースで作成できます。

新しいテーブルを作成したり、既存のテーブルを変更したり、UDT を使用する新しい列を追加したりできます。ネストされた UDT を使用して UDT を作成するには、ネストされた UDT をフリーズする必要があります。

キースペースごとにサポートされている UDTs の数、サポートされているネストレベル、およびUDTs「」を参照してくださいthe section called “Amazon Keyspaces のユーザー定義タイプ (UDTs クォータとデフォルト値”。

UDTs「」を参照してくださいthe section called “データ型に基づいてデータ値のエンコードされたサイズを推定する”。

構文の詳細については、「the section called “型”」を参照してください。

UDTs「」を参照してくださいthe section called “PITR UDTs”。

トピック

- Amazon Keyspaces でユーザー定義タイプ (UDTs) を操作するためのアクセス許可を設定する
- Amazon Keyspaces でユーザー定義タイプ (UDT) を作成する
- Amazon Keyspaces でユーザー定義タイプ (UDTs) を表示する
- Amazon Keyspaces でユーザー定義タイプ (UDT) を削除する

Amazon Keyspaces でユーザー定義タイプ (UDTs) を操作するためのアクセス許可を設定する

テーブルと同様に、UDTsは特定のキースペースにバインドされます。ただし、テーブルとは異なり、UDTs のアクセス許可を直接定義することはできません。UDTsは のリソースとは見なされず AWS 、Amazon リソースネーム (ARN) の形式の一意の識別子はありません。代わりに、UDT で特定のアクションを実行するアクセス許可を IAM プリンシパルに付与するには、UDT がバインドされているキースペースのアクセス許可を定義する必要があります。マルチリージョンキースペースで UDTs を使用するには、追加のアクセス許可が必要です。

UDTs を作成、表示、または削除できるようにするには、プリンシパル、例えば IAM ユーザーやロールには、UDT がバインドされているキースペースで同じアクションを実行するために必要なのと同じアクセス許可が必要です。

詳細については AWS Identity and Access Management、「」を参照してくださいthe section called “AWS Identity and Access Management”。

UDT を作成するアクセス許可

単一リージョンのキースペースに UDT を作成するには、プリンシパルにキースペースの`Create`アクセス許可が必要です。

次の IAM ポリシーはその一例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "cassandra:Create",
            "Resource": [
                "arn:aws:cassandra:aws-region:111122223333:/keyspace/my_keyspace/"
            ]
        }
    ]
}
```

マルチリージョンキースペースに UDT を作成するには、プリンシパルには、アクセス`Create`許可に加えて、指定されたキースペースの アクション`CreateMultiRegionResource`に対するアクセス許可も必要です。

次の IAM ポリシーはその一例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action":  [ "cassandra:Create", "cassandra:CreateMultiRegionResource" ],
            "Resource": [
                "arn:aws:cassandra:aws-region:111122223333:/keyspace/my_keyspace/"
            ]
        }
    ]
}
```

UDT を表示するアクセス許可

単一リージョンのキースペースで UDTs を表示または一覧表示するには、プリンシパルにシステムキースペースの読み取りアクセス許可が必要です。詳細については、「the section called “system_schema_mcs”」を参照してください。

次の IAM ポリシーはその一例です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"cassandra:Select",
         "Resource":[
             "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

マルチリージョンキースペースUDTs を表示または一覧表示するには、プリンシパルにアクションSELECTとSelectMultiRegionResourceシステムキースペースのアクセス許可が必要です。詳細については、「the section called “system_multiregion_info”」を参照してください。

次の IAM ポリシーはその一例です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action": ["cassandra:Select", "cassandra:SelectMultiRegionResource"],
         "Resource":[
             "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

UDT を削除するアクセス許可

単一リージョンのキースペースから UDT を削除するには、プリンシパルに指定されたキースペースの Dropアクションに対するアクセス許可が必要です。

次の IAM ポリシーはその一例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "cassandra:Drop",
            "Resource": [
                "arn:aws:cassandra:aws-region:111122223333:/keyspace/my_keyspace/"
            ]
        }
    ]
}
```

マルチリージョンキースペースから UDT を削除するには、プリンシパルに Dropアクションのアクセス許可と、指定されたキースペースの DropMultiRegionResourceアクションのアクセス許可が必要です。

次の IAM ポリシーはその一例です。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action":  [ "cassandra:Drop", "cassandra:DropMultiRegionResource" ],
            "Resource": [
                "arn:aws:cassandra:aws-region:111122223333:/keyspace/my_keyspace/"
            ]
        }
    ]
}
```

Amazon Keyspaces でユーザー定義タイプ (UDT) を作成する

単一リージョンのキースペースに UDT を作成するには、CQL で CREATE TYPEステートメント、で create-type コマンド AWS CLI、または コンソールを使用できます。

UDT 名は 48 文字以下で、英字で始まり、英数字とアンダースコアのみを含めることができます。Amazon Keyspaces は、大文字を自動的に小文字に変換します。

または、UDT 名を二重引用符で宣言することもできます。二重引用符で UDT 名を宣言する場合、Amazon Keyspaces は大文字を保持し、特殊文字を許可します。

UDT の作成時に名前の一部として二重引用符を使用することもできますが、各二重引用符は追加の二重引用符で工スケープする必要があります。

次の表は、許可された UDT 名の例を示しています。最初の列はタイプの作成時に名前を入力する方法を示し、2 番目の列は Amazon Keyspaces が内部で名前をフォーマットする方法を示しています。Amazon Keyspaces は、 のようなオペレーションにフォーマットされた名前を想定していますGetType。

入力した名前	フォーマットされた名前	メモ
MY_UDT	my_udt	二重引用符を使用しない場合、Amazon Keyspaces はすべての大文字を小文字に変換します。
"MY_UDT"	MY_UDT	二重引用符を使用すると、Amazon Keyspaces は大文字を尊重し、フォーマットされた名前から二重引用符を削除します。

入力した名前	フォーマットされた名前	メモ
`"1234"`	`1234`	二重引用符を使用すると、名前は数字で始まることができ、Amazon Keyspaces はフォーマットされた名前から二重引用符を削除します。
`"Special_Ch@r@cters<>!!"`	`Special_Ch@r@cters<>!!`	二重引用符では、名前に特殊文字を含めることができ、Amazon Keyspaces はフォーマットされた名前から二重引用符を削除します。
`"nested""""""quotes"`	`nested"""quotes`	Amazon Keyspaces は、フォーマットされた名前から外部二重引用符と エスケープ二重引用符を削除します。

Console

Amazon Keyspaces コンソールを使用してユーザー定義タイプ (UDT) を作成する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、キースペースを選択し、リストからキースペースを選択します。
3. UDTsタブを選択します。
4. UDT の作成を選択する
5. UDT の詳細に、UDT の名前を入力します。UDT フィールドで、UDT のスキーマを定義します。
6. 終了するには、UDT の作成を選択します。

Cassandra Query Language (CQL)

CQL を使用してユーザー定義タイプ (UDT) を作成する

この例では、 で使用される書籍報酬テーブルの新しいバージョンを作成しますthe section called “テーブルの作成”。この表では、作成者が特定の書籍に対して受け取るすべての報酬を保存します。ネストされた 2 つの UDTs を作成し、賞を受け取った本に関する情報を含めます。

1. という名前のキースペースを作成しますcatalog。

```
CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

2. 最初のタイプを作成します。このタイプは、本のジャンルを定義するために使用される BISAC コードを保存します。BISAC コードは、英数字コードと最大 4 つのサブジェクト領域で構成されます。

```
CREATE TYPE catalog.bisac (
    bisac_code text,
    subject1 text,
    subject2 text,
    subject3 text,
    subject4 text
);
```

3. 最初の UDT を使用する書籍報酬の 2 番目のタイプを作成します。ネストされた UDT はフリーズする必要があります。

```
CREATE TYPE catalog.book (
    award_title text,
    book_title text,
    publication_date date,
    page_count int,
    ISBN text,
    genre FROZEN <bisac>
);
```

4. 作成者の名前の列を含むテーブルを作成し、書籍の報酬にリストタイプを使用します。リストで使用される UDT はフリーズする必要があります。

```
CREATE TABLE catalog.authors (
    author_name text PRIMARY KEY,
    awards list <FROZEN <book>>
);
```

5. このステップでは、新しいテーブルに 1 行のデータを挿入します。

```
CONSISTENCY LOCAL_QUORUM;
```

```
INSERT INTO catalog.authors (author_name, awards) VALUES (
'John Stiles' ,
```

```
[{
      award_title: 'Wolf',
      book_title: 'Yesterday',
      publication_date: '2020-10-10',
      page_count: 345,
      ISBN: '026204630X',
      genre: { bisac_code:'FIC014090', subject1: 'FICTION', subject2:
 'Historical', subject3: '20th Century', subject4: 'Post-World War II'}
      },
      {award_title: 'Richard Roe',
      book_title: 'Who ate the cake?',
      publication_date: '2019-05-13',
      page_count: 193,
      ISBN: '9780262046305',
      genre: { bisac_code:'FIC022130', subject1: 'FICTION', subject2: 'Mystery &
 Detective', subject3: 'Cozy', subject4: 'Culinary'}
      }]
);
```

6. 最後のステップでは、テーブルからデータを読み取ります。

```
SELECT * FROM catalog.authors;
```

コマンドの出力は次のようになります。

```
 author_name | awards
-------------
+-------------------------------------------------------------------------------------
 John Stiles | [{award_title: 'Wolf', book_title: 'Yesterday', publication_date:
 2020-10-10, page_count: 345, isbn: '026204630X', genre: {bisac_code:
 'FIC014090', subject1: 'FICTION', subject2: 'Historical', subject3: '20th
 Century', subject4: 'Post-World War II'}}, {award_title: 'Richard Roe',
 book_title: 'Who ate the cake?', publication_date: 2019-05-13, page_count: 193,
 isbn: '9780262046305', genre: {bisac_code: 'FIC022130', subject1: 'FICTION',
 subject2: 'Mystery & Detective', subject3: 'Cozy', subject4: 'Culinary'}}]

(1 rows)
```

構文の詳細については、「the section called “CREATE TYPE”」を参照してください。

CLI

を使用してユーザー定義タイプ (UDT) を作成する AWS CLI

1. タイプを作成するには、次の構文を使用できます。

```
aws keyspaces create-type
--keyspace-name 'my_keyspace'
--type-name 'my_udt'
--field-definitions
    '[
        {"name" : "field1", "type" : "int"},
        {"name" : "field2", "type" : "text"}
    ]'
```

2. そのコマンドの出力は、この例のようになります。は UDT のフォーマットされた名前typeNameを返します。

```
{
    "keyspaceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
my_keyspace/",
    "typeName": "my_udt"
}
```

Amazon Keyspaces でユーザー定義タイプ (UDTs) を表示する

単一リージョンのキースペース内のすべての UDTs を表示または一覧表示するには、CQL system_schema_mcs.typesの ステートメントを使用してシステムキースペースのテーブルをクエリするか、 または AWS CLIコンソールで get-typeおよび list-type コマンドを使用します。

どちらのオプションでも、IAM プリンシパルにはシステムキースペースへの読み取りアクセス許可が必要です。詳細については、「the section called “アクセス許可の設定”」を参照してください。

Console

Amazon Keyspaces コンソールでユーザー定義タイプ (UDT) を表示する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、キースペースを選択し、リストからキースペースを選択します。

3. UDTs タブを選択して、キースペース内のすべての UDTsのリストを確認します。
4. 1 つの UDT を詳細に確認するには、リストから UDT を選択します。
5. スキーマタブで、スキーマを確認できます。「Use inUDTs で使用されているかどうかを確認できます。削除できるのは、テーブルまたは他の UDTs で使用されていない UDTsのみです。

Cassandra Query Language (CQL)

CQL を使用して単一リージョンのキースペースのユーザー定義タイプ (UDTs) を表示する

1. 特定のキースペースで使用できるタイプを確認するには、次のステートメントを使用できます。

```
SELECT type_name
FROM system_schema_mcs.types
WHERE keyspace_name = 'my_keyspace';
```

2. 特定のタイプに関する詳細を表示するには、次のステートメントを使用できます。

```
SELECT
    keyspace_name,
    type_name,
    field_names,
    field_types,
    max_nesting_depth,
    last_modified_timestamp,
    status,
    direct_referring_tables,
    direct_parent_types
FROM system_schema_mcs.types
WHERE keyspace_name = 'my_keyspace' AND type_name = 'my_udt';
```

3. を使用して、アカウントに存在するすべての UDTs を一覧表示できますDESC TYPE。

```
DESC TYPES;

 Keyspace my_keyspace
 ---------------------------
 my_udt1  my_udt2

 Keyspace my_keyspace2
```

```
 --------------------------
 my_udt1
```

4. を使用して、現在選択されているキースペース内のすべての UDTs を一覧表示できますDESC TYPE。

```
USE my_keyspace;
my_keyspace DESC TYPES;

my_udt1  my_udt2
```

5. マルチリージョンキースペース内のすべての UDTs を一覧表示するには、system_multiregion_infoキースペースtypesのシステムテーブルをクエリします。次のクエリはその一例です。

```
SELECT keyspace_name, type_name, region, status FROM
 system_multiregion_info.types WHERE keyspace_name = 'mykeyspace' AND table_name
 = 'mytable';
```

このコマンドの出力は次のようになります。

```
keyspace_name     | table_name         | region                | status
mykeyspace        | mytable            | us-east-1             | ACTIVE
mykeyspace        | mytable            | ap-southeast-1        | ACTIVE
mykeyspace        | mytable            | eu-west-1             | ACTIVE
```

CLI

でユーザー定義タイプ (UDTsを表示する AWS CLI

1. キースペースで使用できるタイプを一覧表示するには、 list-types コマンドを使用できます。

```
aws keyspaces list-types
--keyspace-name 'my_keyspace'
```

そのコマンドの出力は、この例のようになります。

```
{
    "types": [
```

```
        "my_udt",
        "parent_udt"
    ]
}
```

2. 特定のタイプに関する詳細を表示するには、 get-type コマンドを使用できます。

```
aws keyspaces get-type
--type-name 'my_udt'
--keyspace-name 'my_keyspace'
```

このコマンドの出力は次の例のようになります。

```
{
    "keyspaceName": "my_keyspace",
    "typeName": "my_udt",
    "fieldDefinitions": [
        {
            "name": "a",
            "type": "int"
        },
        {
            "name": "b",
            "type": "text"
        }
    ],
    "lastModifiedTimestamp": 1721328225776,
    "maxNestingDepth": 3
    "status": "ACTIVE",
    "directReferringTables": [],
    "directParentTypes": [
        "parent_udt"
    ],
    "keyspaceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
my_keyspace/"
}
```

Amazon Keyspaces でユーザー定義タイプ (UDT) を削除する

キースペース内の UDT を削除するには、CQL の DROP TYPEステートメント、 での delete-type コマンド AWS CLI、または コンソールを使用できます。

Console

Amazon Keyspaces コンソールでユーザー定義タイプ (UDT) を削除する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、キースペースを選択し、リストからキースペースを選択します。
3. UDTsタブを選択します。
4. 削除する UDT を選択します。「Use in」で、削除するタイプがテーブルやその他の UDT で現在使用されていないことを確認できます。
5. 概要の上にある「削除」を選択します。
6. 表示されるダイアログDeleteに と入力し、UDT の削除を選択します。

Cassandra Query Language (CQL)

CQL でユーザー定義タイプ (UDT) を削除する

- タイプを削除するには、次のステートメントを使用できます。

```
DROP TYPE my_keyspace.my_udt;
```

構文の詳細については、「the section called “DROP TYPE”」を参照してください。

CLI

を使用してユーザー定義タイプ (UDT) を削除する AWS CLI

1. タイプを削除するには、次のコマンドを使用します。

```
aws keyspaces delete-type
--keyspace-name 'my_keyspace'
--type-name 'my_udt'
```

2. コマンドの出力は次の例のようになります。

```
{
    "keyspaceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
my_keyspace/",
```

```
    "typeName": "my_udt"
}
```

Amazon Keyspaces で CQL クエリを使用する

このセクションでは、Amazon Keyspaces (Apache Cassandra 向け) でのクエリの操作について説明します。データのクエリ、変換、および管理に使用できる CQL ステートメントは、SELECT、INSERT、UPDATE、DELETE です。次のトピックでは、クエリを操作する際に使用できる、より複雑なオプションの概要を説明します。例を含む完全な言語構文については、「the section called “DML ステートメント”」を参照してください。

トピック

- Amazon Keyspaces でクエリの SELECT ステートメントで IN 演算子を使用する
- Amazon Keyspaces で ORDER BY を使用して結果を並べ替える
- Amazon Keyspaces で結果のページを分割する

Amazon Keyspaces でクエリの **SELECT** ステートメントで **IN** 演算子を使用する

SELECT IN

ステートメントを使用してテーブルからデータをクエリできます。SELECT ステートメントは、テーブル内の 1 つ以上の行の 1 つ以上の列を読み取り、リクエストに一致する行を含む結果セットを返します。SELECT ステートメントには、結果セットで読み取る列と返す列を決定する select_clause が含まれています。この句には、データを返す前に変換する命令を含めることができます。オプションの WHERE 句は、どの行をクエリする必要があるかを指定するもので、主キーの一部である列のリレーションで構成されます。Amazon Keyspaces は、WHERE 句内の IN キーワードをサポートしています。このセクションでは、例を使用して、Amazon Keyspaces が IN キーワードを含む SELECT ステートメントを処理する方法を示します。

この例は、Amazon Keyspaces が IN キーワードを含む SELECT ステートメントをサブクエリに分割する方法を示しています。この例では、my_keyspace.customers という名前のテーブルを使用しています。このテーブルには、プライマリキー列が 1 つ (department_id)、クラスタリング列とが 2 つ (sales_region_id および sales_representative_id)、customer_name 列に顧客の名前を含む列が 1 つあります。

```
SELECT * FROM my_keyspace.customers;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  0            |        0        |            0            |    a
  0            |        0        |            1            |    b
  0            |        1        |            0            |    c
  0            |        1        |            1            |    d
  1            |        0        |            0            |    e
  1            |        0        |            1            |    f
  1            |        1        |            0            |    g
  1            |        1        |            1            |    h
```

このテーブルを使用して、次の SELECT ステートメントを実行して、WHERE 句内の IN キーワードで関心のある部門や販売地域の顧客を検索できます。以下のステートメントは、この例です。

```
SELECT * FROM my_keyspace.customers WHERE department_id IN (0, 1) AND sales_region_id
 IN (0, 1);
```

Amazon Keyspaces は、以下の出力に示しているように、このステートメントを 4 つのサブクエリに分割しています。

```
SELECT * FROM my_keyspace.customers WHERE department_id = 0 AND sales_region_id = 0;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  0            |        0        |           0             |    a
  0            |        0        |           1             |    b

SELECT * FROM my_keyspace.customers WHERE department_id = 0 AND sales_region_id = 1;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  0            |        1        |          0              |    c
  0            |        1        |          1              |    d

SELECT * FROM my_keyspace.customers WHERE department_id = 1 AND sales_region_id = 0;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  1            |        0        |          0              |    e
  1            |        0        |          1              |    f
```

```
SELECT * FROM my_keyspace.customers WHERE department_id = 1 AND sales_region_id = 1;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  1            |        1        |           0             |    g
  1            |        1        |           1             |    h
```

IN キーワードを使用すると、Amazon Keyspaces は、以下のいずれかの場合に結果を自動的にページ分割します。

- 10 回目ごとにサブクエリが処理されます。
- 1 MB の論理 IO を処理した後。
- PAGE SIZE を設定した場合、Amazon Keyspaces は、PAGE SIZE セットに基づいて処理するクエリの数を読み取った後にページ分割します。
- LIMIT キーワードを使用して返される行の数を減らすと、Amazon Keyspaces は、セット LIMIT に基づいて処理するクエリの数を読み取った後にページ分割します。

以下の表では、これを例を挙げて説明しています。

ページ分割の詳細については、「the section called “結果のページ分割”」を参照してください。

```
SELECT * FROM my_keyspace.customers;

         department_id | sales_region_id | sales_representative_id | customer_name
        ---------------+-----------------+-------------------------+--------------
          2            |        0        |           0             |    g
          2            |        1        |           1             |    h
          2            |        2        |           2             |    i
          0            |        0        |           0             |    a
          0            |        1        |           1             |    b
          0            |        2        |           2             |    c
          1            |        0        |           0             |    d
          1            |        1        |           1             |    e
          1            |        2        |           2             |    f
          3            |        0        |           0             |    j
          3            |        1        |           1             |    k
          3            |        2        |           2             |    l
```

このテーブルに対して次のステートメントを実行すると、ページ分割の仕組みを確認できます。

```
SELECT * FROM my_keyspace.customers WHERE department_id IN (0, 1, 2, 3) AND
 sales_region_id IN (0, 1, 2) AND sales_representative_id IN (0, 1);
```

Amazon Keyspaces は、このステートメントを 24 個のサブクエリとして処理します。これは、このクエリに含まれるすべての IN 用語のデカルト積の基数が 24 であるためです。

```
 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  0            |        0        |          0              |    a
  0            |        1        |          1              |    b
  1            |        0        |          0              |    d
  1            |        1        |          1              |    e

---MORE---
 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  2            |        0        |          0              |    g
  2            |        1        |          1              |    h
  3            |        0        |          0              |    j

---MORE---
 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  3            |        1        |          1              |    k
```

この例は、IN キーワードを含む SELECT ステートメントで ORDER BY 句を使用する方法を示しています。

```
SELECT * FROM my_keyspace.customers WHERE department_id IN (3, 2, 1) ORDER BY
 sales_region_id DESC;

         department_id | sales_region_id | sales_representative_id | customer_name
        ---------------+-----------------+-------------------------+--------------
          3            |        2        |          2              |    l
          3            |        1        |          1              |    k
          3            |        0        |          0              |    j
          2            |        2        |          2              |    i
          2            |        1        |          1              |    h
          2            |        0        |          0              |    g
          1            |        2        |          2              |    f
          1            |        1        |          1              |    e
```

```
          1            |        0        |          0              |    d
```

サブクエリは、パーティションキー列とクラスタリングキー列がクエリに表示される順序で処理されます。以下の例では、パーティションキー値「2」のサブクエリが最初に処理され、続いてパーティションキー値「3」と「1」のサブクエリが処理されます。特定のサブクエリの結果は、クエリの順序付け句 (存在する場合) またはテーブル作成時に定義されたテーブルのクラスタリング順序に従って順序付けられます。

```
SELECT * FROM my_keyspace.customers WHERE department_id IN (2, 3, 1) ORDER BY
 sales_region_id DESC;

         department_id | sales_region_id | sales_representative_id | customer_name
        ---------------+-----------------+-------------------------+--------------
          2            |        2        |          2              |    i
          2            |        1        |          1              |    h
          2            |        0        |          0              |    g
          3            |        2        |          2              |    l
          3            |        1        |          1              |    k
          3            |        0        |          0              |    j
          1            |        2        |          2              |    f
          1            |        1        |          1              |    e
          1            |        0        |          0              |    d
```

Amazon Keyspaces で **ORDER BY** を使用して結果を並べ替える

ORDER BY 句は、SELECT ステートメントで返された結果のソート順序を指定するものです。このステートメントでは列名のリストが引数として解釈されるので、各列に対してデータのソート順を指定できます。クラスタリング列は順序句でしか指定できず、非クラスタリング列は許容されません。

返された結果に対するソート順序のオプションは、昇順の ASC と降順の DESC です。

```
SELECT * FROM my_keyspace.my_table ORDER BY (col1 ASC, col2 DESC, col3 ASC);

         col1 | col2 | col3
        ------+------+------
          0   |  6   |  a
          1   |  5   |  b
          2   |  4   |  c
          3   |  3   |  d
          4   |  2   |  e
          5   |  1   |  f
```

```
 6   |  0   |  g
```

```
SELECT * FROM my_keyspace.my_table ORDER BY (col1 DESC, col2 ASC, col3 DESC);

 col1 | col2 | col3
------+------+------
 6   |  0   |  g
 5   |  1   |  f
 4   |  2   |  e
 3   |  3   |  d
 2   |  4   |  c
 1   |  5   |  b
 0   |  6   |  a
```

クエリステートメントでソート順序を指定しない場合、クラスタリング列のデフォルトの順序が使用されます。

順序句で使用可能なソート順序は、テーブル作成時に各クラスタリング列に割り当てられたソート順序に依存します。クエリ結果は、テーブル作成時にすべてのクラスタリング列に対して定義された順序、または定義されたソート順序とは逆の順序でのみソートできます。その他の組み合わせは使用できません。

例えば、テーブルの CLUSTERING ORDER が (col1 ASC, col2 DESC, col3 ASC) である場合、ORDER BY の有効なパラメータは (col1 ASC, col2 DESC, col3 ASC) または (col1 DESC, col2 ASC, col3 DESC) のいずれかになります。CLUSTERING ORDER に関する詳細については、the section called “CREATE TABLE” の table_options を参照してください。

Amazon Keyspaces で結果のページを分割する

Amazon Keyspaces では、SELECT ステートメントを処理するために読み取られたデータが 1 MB を超えると、SELECT ステートメントの結果のページ分割が自動で行われます。ページ割りを行うことで SELECT ステートメント結果が 1 MB サイズ (またはそれ以下) のデータの「ページ」に分割されます。アプリケーションは結果の最初のページ、次に 2 ページと処理できます。クライアントでは、複数の行を返す SELECT クエリを処理する際に必ずページ分割トークンのチェックが行われます。

クライアントにより、1 MB を超えるデータの読み取りが必要になる PAGE SIZE が提供されると、Amazon Keyspaces では、1 MB のデータ読み取り増分に基づいて結果が複数のページに自動的に分割されます。

例えば、行の平均サイズが 100 KB である場合に PAGE SIZE を 20 に指定すると、Amazon Keyspaces では 10 行 (読み取られるデータは 1000 KB) が読み取られ、その後、自動的にデータのページ分割が行われます。

Amazon Keyspaces では、結果セットで返された行数ではなく、リクエストを処理するために読み取られた行数に基づいて結果のページ分割が行われるため、フィルタリングされたクエリを実行している場合は、一部のページに行が含まれていないことがあります。

例えば、PAGE SIZE を 10 に設定し、Keyspaces で SELECT クエリの処理のために 30 行が評価された場合、Amazon Keyspaces から 3 ページが返されます。行のサブセットのみがクエリに一致した場合、一部のページの行数が 10 行未満になることがあります。LIMIT クエリの PAGE SIZE による読み取りキャパシティへの影響を示した例については、「the section called “限定クエリの読み取りキャパシティ消費量を推定する”」を参照してください。

Apache Cassandra のページ分割との比較については、「the section called “Pagination (ページ分割)”」を参照してください。

Amazon Keyspaces での変更データキャプチャ (CDC) ストリームの使用

Amazon Keyspaces 変更データキャプチャ (CDC) は、Amazon Keyspaces テーブルからの行レベルの変更イベントをほぼリアルタイムで記録します。

Amazon Keyspaces CDC を使用すると、産業用 IoT や不正検出などのイベント駆動型のユースケースや、全文検索やデータアーカイブなどのデータ処理のユースケースが可能になります。Amazon Keyspaces CDC がストリームでキャプチャする変更イベントは、データ分析、テキスト検索、ML トレーニング/推論、アーカイブのための継続的なデータバックアップなどのビジネスクリティカルな機能を実行するダウンストリームアプリケーションで消費できます。例えば、ストリームデータを Amazon OpenSearch Service、Amazon Redshift、Amazon S3 などの AWS 分析およびストレージサービスに転送して、さらに処理できます。

Amazon Keyspaces CDC は、テーブルの時系列および重複排除された変更レコードを提供し、データスループットと保持時間の自動スケーリングは最大 24 時間です。

Amazon Keyspaces CDC ストリームは完全にサーバーレスであり、変更イベントをキャプチャするためにデータインフラストラクチャを管理する必要はありません。さらに、Amazon Keyspaces CDC はコンピューティングまたはストレージのテーブル容量を消費しません。詳細については、「the section called “CDC ストリームの仕組み”」を参照してください。

Amazon Keyspaces Streams API を使用して、Amazon Keyspaces CDC ストリームを消費し、コンテンツに基づいてアクションを実行するアプリケーションを構築できます。使用可能なエンドポイントについては、「」を参照してくださいthe section called “CDC ストリームエンドポイントにアクセスする”。

Streams API で Amazon Keyspaces で使用できるすべてのオペレーションの完全なリストについては、「Amazon Keyspaces Streams API リファレンス」を参照してください。

トピック

- Amazon Keyspaces での変更データキャプチャ (CDC) ストリームの仕組み
- Amazon Keyspaces で変更データキャプチャ (CDC) ストリームを使用する方法

Amazon Keyspaces での変更データキャプチャ (CDC) ストリームの仕組み

このセクションでは、Amazon Keyspaces での変更データキャプチャ (CDC) ストリームの仕組みの概要を説明します。

Amazon Keyspaces 変更データキャプチャ (CDC) は、行レベルの変更の順序付けられたシーケンスを Amazon Keyspaces テーブルに記録し、この情報をストリームと呼ばれる口グに最大 24 時間保存します。行レベルを変更するたびに、プライマリキー列情報と、すべての列を含む行の「前」状態と「後」状態を保持する新しい CDC レコードが生成されます。アプリケーションはストリームにアクセスし、ミューテーションをほぼリアルタイムで表示できます。

テーブルで CDC を有効にすると、Amazon Keyspaces は新しい CDC ストリームを作成し、テーブル内のすべての変更に関する情報をキャプチャし始めます。CDC ストリームには、次の形式の Amazon リソースネーム (ARN) があります。

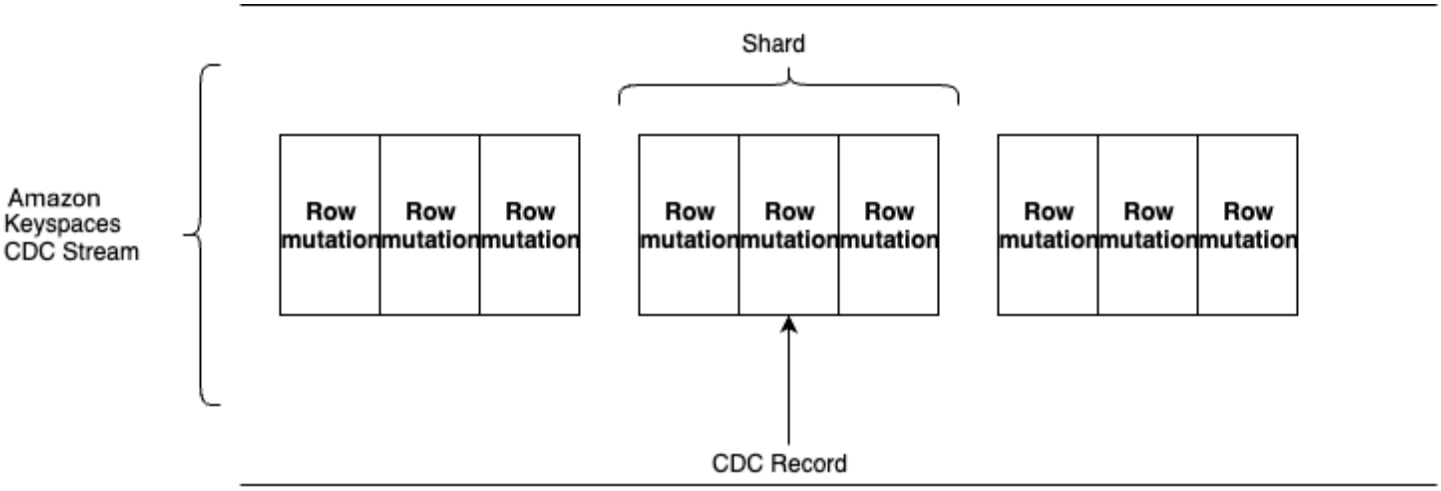
```
arn:${Partition}:cassandra:{Region}:${Account}:/keyspace/${keyspaceName}/table/
${tableName}/stream/${streamLabel}
```

最初に CDC ストリームを有効にするときに、CDC ストリームが各レコードに対して収集する情報のタイプまたはビュータイプを選択できます。その後、ストリームのビュータイプを変更することはできません。Amazon Keyspaces では、次のビュータイプがサポートされています。

- NEW_AND_OLD_IMAGES – ミューテーションの前と後に行のバージョンをキャプチャします。これがデフォルトです。
- NEW_IMAGE – ミューテーション後の行のバージョンをキャプチャします。
- OLD_IMAGE – ミューテーションの前に行のバージョンをキャプチャします。

- KEYS_ONLY – 変更された行のパーティションキーとクラスタリングキーをキャプチャします。

すべての CDC ストリームはレコードで構成されます。各レコードは、Amazon Keyspaces テーブルの 1 行の変更を表します。レコードは、シャードと呼ばれるグループに論理的に整理されます。これらのグループは、プライマリキーの範囲 (パーティションキーとクラスタリングキーの範囲の組み合わせ) によって論理的に整理され、Amazon Keyspaces の内部構造です。各シャードは複数のレコードのコンテナとして機能し、これらのレコードにアクセスして反復するために必要な情報が含まれています。



各 CDC レコードには、シャード内でレコードが発行された順序を反映したシーケンス番号が割り当てられます。シーケンス番号は、各シャード内で増加し、一意であることが保証されます。

Amazon Keyspaces はシャードを自動的に作成および削除します。トラフィック負荷に基づいて、Amazon Keyspaces は時間の経過とともにシャードを分割またはマージすることもできます。例えば、Amazon Keyspaces は 1 つのシャードを複数の新しいシャードに分割したり、シャードを新しい 1 つのシャードにマージしたりできます。Amazon Keyspaces APIsシャードと CDC ストリーム情報を公開し、シャードの系統グラフ全体にアクセスして、消費するアプリケーションが正しい順序でレコードを処理できるようにします。

Amazon Keyspaces CDC は、アプリケーションを構築するときに信頼できる以下の原則に基づいています。

- 各行レベルのミューテーションレコードは、CDC ストリームに 1 回だけ表示されます。
- 系統順にシャードを使用すると、各行レベルのミューテーションレコードは、プライマリキーの実際のミューテーション順序と同じ順序で表示されます。

トピック

- Amazon Keyspaces の CDC ストリームのデータ保持の仕組み
- Amazon Keyspaces の CDC ストリームで有効期限 (TTL) データの有効期限がどのように機能するか
- Amazon Keyspaces での CDC ストリームのバッチオペレーションの仕組み
- Amazon Keyspaces の CDC ストリームでの静的列の仕組み
- Amazon Keyspaces の CDC ストリームの保管時の暗号化の仕組み
- Amazon Keyspaces の CDC ストリームでのマルチリージョンレプリケーションの仕組み
- CDC ストリームと AWS サービスとの統合

Amazon Keyspaces の CDC ストリームのデータ保持の仕組み

Amazon Keyspaces は、CDC ストリームのレコードを 24 時間保持します。保持期間を変更することはできません。テーブルで CDC を無効にすると、ストリーム内のデータは 24 時間読み取ることができます。この時間が経過すると、データの有効期限が切れ、レコードは自動的に削除されます。

Amazon Keyspaces の CDC ストリームで有効期限 (TTL) データの有効期限がどのように機能するか

Amazon Keyspaces は、CDC 変更レコード`expirationTime`の というメタデータフィールドの列/セルレベルでの有効期限と行レベルを表示します。Amazon Keyspaces TTL がセルの有効期限を検出すると、CDC は TTL を変更のオリジンとして表示する新しい変更レコードを作成します。TTL の詳細については、「the section called “TTL によるデータの期限切れ”」を参照してください。

Amazon Keyspaces での CDC ストリームのバッチオペレーションの仕組み

バッチオペレーションは、内部的に個々の行レベルの変更に分割されます。Amazon Keyspaces は、変更がバッチオペレーションで発生した場合でも、CDC ストリーム内のすべてのレコードを行レベルで保持します。Amazon Keyspaces は、CDC ストリーム内のレコードの順序を、行レベルまたはプライマリキーで発生したミューテーション順序と同じ順序で維持します。

Amazon Keyspaces の CDC ストリームでの静的列の仕組み

静的列値は、Cassandra のパーティション内のすべての行間で共有されます。この動作により、Amazon Keyspaces は静的列への更新を CDC ストリーム内の別のレコードとしてキャプチャします。次の例は、静的列ミューテーションの動作をまとめたものです。

- 静的列のみが更新されると、CDC ストリームには、静的列の行変更が行内の唯一の列として含まれます。

- 静的列を変更せずに行が更新されると、CDC ストリームには静的列を除くすべての列を含む行変更が含まれます。
- 行が静的列とともに更新されると、CDC ストリームには 2 つの個別の行変更が含まれます。1 つは静的列用、もう 1 つは残りの行用です。

Amazon Keyspaces の CDC ストリームの保管時の暗号化の仕組み

CDC 順序付きログに保管中のデータを暗号化するために、Amazon Keyspaces はテーブルに既に使用されているのと同じ暗号化キーを使用します。保管時の暗号化の詳細については、「the section called “保管データ暗号化”」を参照してください。

Amazon Keyspaces の CDC ストリームでのマルチリージョンレプリケーションの仕組み

`update-table` API または CQL コマンドを使用して、マルチリージョンテーブルの個々のレプリカの CDC `ALTER TABLE` ストリームを有効または無効にできます。非同期レプリケーションと競合の解決により、マルチリージョンテーブルの CDC ストリームは全体で一貫性がありません AWS リージョン。したがって、Amazon Keyspaces がストリームでキャプチャするレコードは、異なるリージョンで異なる順序で表示される可能性があります。

マルチリージョンレプリケーションの詳細については、「」を参照してくださいthe section called “マルチリージョンレプリケーション”。

CDC ストリームと AWS サービスとの統合

Amazon Keyspaces で CDC ストリームの VPC エンドポイントを操作する方法

VPC エンドポイントを使用して Amazon Keyspaces CDC ストリームにアクセスできます。ストリームの VPC エンドポイントを作成してアクセスする方法については、「」を参照してくださいthe section called “ストリームでのインターフェイス VPC エンドポイントの使用”。

Amazon Keyspaces の CDC ストリームに対する CloudWatch によるモニタリングの仕組み

Amazon CloudWatch を使用して、Amazon Keyspaces CDC エンドポイントに対して行われた API コールをモニタリングできます。使用可能なメトリクスの詳細については、「」を参照してくださいthe section called “Amazon Keyspaces 変更データキャプチャ (CDC) のメトリクス”。

Amazon Keyspaces の CDC ストリームでの CloudTrail を使用したログ記録の仕組み

Amazon Keyspaces CDC は AWS CloudTrail、Amazon Keyspaces のユーザー、ロール、または のサービス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 では Amazon Keyspaces のデータ定義言語 (DDL) API コールとデータ操作言語 (DML) API コールがイベントとしてキャプチャされます。キャプチャされたコールには、Amazon Keyspaces コンソールからのコールと、Amazon Keyspaces API オペレーションへのプログラムによるコールが含まれます。

CloudTrail でキャプチャされた CDC イベントの詳細については、「」を参照してくださいthe section called “CloudTrail による口グ記録”。

Amazon Keyspaces の CDC ストリームでのタグ付けの仕組み

Amazon Keyspaces CDC ストリームはタグ付け可能なリソースです。CQL、 AWS SDK、または を使用してプログラムでテーブルを作成するときに、ストリームにタグを付けることができます AWS CLI。既存のストリームにタグ付けしたり、タグを削除したり、ストリームのタグを表示したりすることもできます。詳細については、「the section called “キースペース、テーブル、ストリームにタグを付ける”」を参照してください。

Amazon Keyspaces で変更データキャプチャ (CDC) ストリームを使用する方法

トピック

- Amazon Keyspaces で CDC ストリームを操作するためのアクセス許可を設定する
- Amazon Keyspaces で CDC ストリームエンドポイントにアクセスする方法
- Amazon Keyspaces で新しいテーブルを作成するときに CDC ストリームを有効にする
- Amazon Keyspaces で既存のテーブルの CDC ストリームを有効にする
- Amazon Keyspaces で CDC ストリームを無効にする
- Amazon Keyspaces で CDC ストリームを表示する
- Amazon Keyspaces の CDC ストリームのレコードにアクセスする
- Kinesis Client Library (KCL) を使用して Amazon Keyspaces ストリームを処理する

Amazon Keyspaces で CDC ストリームを操作するためのアクセス許可を設定する

CDC ストリームを有効にするには、IAM ユーザーやロールなどのプリンシパルに次のアクセス許可が必要です。

詳細については AWS Identity and Access Management、「」を参照してくださいthe section called “AWS Identity and Access Management”。

テーブルの CDC ストリームを有効にするアクセス許可

Amazon Keyspaces テーブルの CDC ストリームを有効にするには、プリンシパルにまずテーブルを作成するアクセス許可が必要であり、次にサービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesCDC を作成するアクセス許可が必要です。Amazon Keyspaces は、サービスにリンクされたロールを使用して、ユーザーに代わって CloudWatch メトリクスをアカウントに発行します。

次の IAM ポリシーがその例です。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":[
                // Enable CDC stream during table creation
                "cassandra:Create",
                "cassandra:CreateMultiRegionResource",
                // Enable CDC stream after creation
                "cassandra:Alter",
                "cassandra:AlterMultiRegionResource"
            ],
            "Resource":[
                "arn:aws:cassandra:aws-region:555555555555:/keyspace/my_keyspace/*",
                "arn:aws:cassandra:aws-region:555555555555:/keyspace/system*"
            ]
        },
        {
            "Sid": "KeyspacesCDCServiceLinkedRole",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/cassandra-
streams.amazonaws.com/AWSServiceRoleForAmazonKeyspacesCDC",
            "Condition": {
              "StringLike": {
                "iam:AWSServiceName": "cassandra-streams.amazonaws.com"
              }
            }
        }
```

```
    ]
}
```

ストリームを無効にするには、ALTER TABLEアクセス許可のみが必要です。

CDC ストリームを表示するアクセス許可

CDC ストリームを表示または一覧表示するには、プリンシパルにシステムキースペースの読み取りアクセス許可が必要です。詳細については、「the section called “system_schema_mcs”」を参照してください。

次の IAM ポリシーがその例です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"cassandra:Select",
         "Resource":[
             "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

または Amazon Keyspaces API を使用して CDC ストリームを表示 AWS CLI または一覧表示するには、プリンシパルにアクション cassandra:ListStreamsおよび のアクセス許可が必要ですcassandra:GetStream。

次の IAM ポリシーがその例です。

```
{
            "Effect": "Allow",
            "Action": [
                "cassandra:Select",
                "cassandra:ListStreams",
                "cassandra:GetStream"
            ],
            "Resource": "*"
}
```

CDC ストリームを読み取るアクセス許可

CDC ストリームを読み取るには、プリンシパルに次のアクセス許可が必要です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:GetStream",
            "cassandra:GetShardIterator",
            "cassandra:GetRecords"
         ],
         "Resource":[
            // CDC Stream ARN
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/my_keyspace/
table/my_table/stream/stream_label"
         ]
      }
   ]
}
```

Kinesis Client Library (KCL) を使用して Amazon Keyspaces CDC ストリームを処理するアクセス許可

KCL で Amazon Keyspaces CDC ストリームを処理するには、IAM プリンシパルに次のアクセス許可が必要です。

- `Amazon Keyspaces` – 指定された Amazon Keyspaces CDC ストリームへの読み取り専用アクセス。
- `DynamoDB` – KCL ストリーム処理に必要な`shard lease`テーブルの作成、テーブルへの読み取りおよび書き込みアクセス、インデックスへの読み取りアクセスを行うためのアクセス許可。
- `CloudWatch` – KCL を使用した Amazon Keyspaces CDC ストリーム処理から CloudWatch アカウントの KCL クライアントアプリケーションの名前空間にメトリクスデータを発行するアクセス許可。モニタリングの詳細については、Amazon CloudWatch で Kinesis Client Library をモニタリングする」を参照してください。

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:GetStream",
            "cassandra:GetShardIterator",
            "cassandra:GetRecords"
         ],
         "Resource":[
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/my_keyspace/
table/my_table/stream/stream_label"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:CreateTable",
            "dynamodb:DescribeTable",
            "dynamodb:UpdateTable",
            "dynamodb:GetItem",
            "dynamodb:UpdateItem",
            "dynamodb:PutItem",
            "dynamodb:DeleteItem",
            "dynamodb:Scan"
         ],
         "Resource":[
            "arn:aws:dynamodb:aws-region:111122223333:table/KCL_APPLICATION_NAME"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:CreateTable",
            "dynamodb:DescribeTable",
            "dynamodb:GetItem",
            "dynamodb:UpdateItem",
            "dynamodb:PutItem",
            "dynamodb:DeleteItem",
            "dynamodb:Scan"
         ],
         "Resource":[
            "arn:aws:dynamodb:aws-region:111122223333:table/KCL_APPLICATION_NAME-
WorkerMetricStats",
```

```
            "arn:aws:dynamodb:aws-region:111122223333:table/KCL_APPLICATION_NAME-
CoordinatorState"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:aws-region:111122223333:table/KCL_APPLICATION_NAME/index/
*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "cloudwatch:PutMetricData"
         ],
         "Resource":"*"
      }
   ]
}
```

Amazon Keyspaces で CDC ストリームエンドポイントにアクセスする方法

Amazon Keyspaces は、Amazon Keyspaces AWS リージョン が利用可能な各 のキースペース/テーブルとストリーム用に個別のエンドポイントを維持します。ストリームにアクセスするには、次の例に示すように、テーブルのリージョンを選択し、エンドポイントamazonaws.comapi.awsでcassandraプレフィックスを cassandra-streamsと に置き換えます。

リージョン名	リージョン	キースペースとテーブルのエンドポイント	ストリームのエンドポイント
米国東部 (バージニア北部)	us-east-1	cassandra.us-east-1.amazonaws.com	cassandra-streams.us-east-1.api.aws

次の表に、ストリームエンドポイントの完全なリストを示します。

リージョン名	リージョン	エンドポイント	プロトコル	
米国東部 (オハイオ)	us-east-2	cassandra-streams.us-east-2.api.aws	HTTPS	
米国東部 (バージニア北部)	us-east-1	cassandra-streams.us-east-1.api.aws	HTTPS	
米国西部 (北カリフォルニア)	us-west-1	cassandra-streams.us-west-1.api.aws	HTTPS	
米国西部 (オレゴン)	us-west-2	cassandra-streams.us-west-2.api.aws	HTTPS	
アフリカ (ケープタウン)	af-south-1	cassandra-streams.af-south-1.api.aws	HTTPS	
アジアパシフィック (香港)	ap-east-1	cassandra-streams.ap-east-1.api.aws	HTTPS	
アジアパシフィック (ムンバイ)	ap-south-1	cassandra-streams.ap-south-1.api.aws	HTTPS	
アジアパシフィック (ソウル)	ap-northeast-2	cassandra-streams.ap-northeast-2.api.aws	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
アジアパシフィック (シンガポール)	ap-southeast-1	cassandra-streams.ap-southeast-1.api.aws	HTTPS	
アジアパシフィック (シドニー)	ap-southeast-2	cassandra-streams.ap-southeast-2.api.aws	HTTPS	
アジアパシフィック (東京)	ap-northeast-1	cassandra-streams.ap-northeast-1.api.aws	HTTPS	
カナダ (中部)	ca-central-1	cassandra-streams.ca-central-1.api.aws	HTTPS	
欧州 (フランクフルト)	eu-central-1	cassandra-streams.eu-central-1.api.aws	HTTPS	
欧州 (アイルランド)	eu-west-1	cassandra-streams.eu-west-1.api.aws	HTTPS	
欧州 (ロンドン)	eu-west-2	cassandra-streams.eu-west-2.api.aws	HTTPS	
欧州 (パリ)	eu-west-3	cassandra-streams.eu-west-3.api.aws	HTTPS	
欧州 (ストックホルム)	eu-north-1	cassandra-streams.eu-north-1.api.aws	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
中東 (バーレーン)	me-south-1	cassandra-streams.me-south-1.api.aws	HTTPS	
南米 (サンパウロ)	sa-east-1	cassandra-streams.sa-east-1.api.aws	HTTPS	
AWS GovCloud (米国東部)	us-gov-east-1	cassandra-streams.us-gov-east-1.api.aws	HTTPS	
AWS GovCloud (米国西部)	us-gov-west-1	cassandra-streams.us-gov-west-1.api.aws	HTTPS	

Amazon Keyspaces で新しいテーブルを作成するときに CDC ストリームを有効にする

テーブルの作成時に CDC ストリームを有効にするには、CQL で `CREATE TABLE`ステートメントを使用するか、 で `create-table` コマンドを使用します AWS CLI。

テーブル内の変更された行ごとに、Amazon Keyspaces は`cdc_specification`選択した `view_type`の に基づいて次の変更をキャプチャできます。

- `NEW_AND_OLD_IMAGES` – 変更前と変更後の両方のバージョンの行。これがデフォルトです。
- `NEW_IMAGE` – 変更後の行のバージョン。
- `OLD_IMAGE` – 変更前の行のバージョン。
- `KEYS_ONLY` – 変更された行のパーティションキーとクラスタリングキー。

ストリームにタグを付ける方法については、「」を参照してくださいthe section called “新しいテーブルでタグを使用してストリームを作成する”。

> **Note**
>
> Amazon Keyspaces CDC には、ユーザーに代わって Amazon Keyspaces CDC ストリームから CloudWatch アカウントの `"cloudwatch:namespace": "AWS/Cassandra"` にメトリクスデータを発行するサービスにリンクされたロール (AWSServiceRoleForAmazonKeyspacesCDC) が必要です。このロールは自動的に作成されます。詳細については、「the section called “CDC ストリーム”」を参照してください。

Cassandra Query Language (CQL)

CQL でテーブルを作成するときに CDC ストリームを有効にする

1.
```
CREATE TABLE mykeyspace.mytable (a text, b text, PRIMARY KEY(a))
WITH CUSTOM_PROPERTIES={'cdc_specification': {'view_type': 'NEW_IMAGE'}} AND CDC
 = TRUE;
```

2. ストリーム設定を確認するには、次のステートメントを使用します。

```
SELECT keyspace_name, table_name, cdc, custom_properties FROM
 system_schema_mcs.tables WHERE keyspace_name = 'mykeyspace' AND table_name =
 'mytable';
```

そのステートメントの出力は次のようになります。

```
SELECT keyspace_name, table_name, cdc, custom_properties FROM
 system_schema_mcs.tables WHERE keyspace_name = 'mykeyspace' AND table_name =
 'mytable';keyspace_name | table_name | cdc  | custom_properties
--------------+------------+------
+-------------------------------------------------------------------------------------------------
    mykeyspace |   mytable  | True | {'capacity_mode':
 {'last_update_to_pay_per_request_timestamp': '1741383893782',
 'throughput_mode': 'PAY_PER_REQUEST'}, 'cdc_specification':
 {'latest_stream_arn': 'arn:aws:cassandra:aws-region:111122223333:/keyspace/
mykeyspace/table/mytable/stream/2025-03-07T21:44:53.783', 'status': 'ENABLED',
 'view_type': 'NEW_IMAGE'}, 'encryption_specification': {'encryption_type':
 'AWS_OWNED_KMS_KEY'}, 'point_in_time_recovery': {'status': 'disabled'}}>
```

CLI

でテーブルを作成するときに CDC ストリームを有効にする AWS CLI

1. ストリームを作成するには、次の構文を使用できます。

```
aws keyspaces create-table \
--keyspace-name 'mykeyspace' \
--table-name 'mytable' \
--schema-definition 'allColumns=[{name=a,type=text},
{name=b,type=text}],partitionKeys=[{name=a}]' \
--cdc-specification status=ENABLED,viewType=NEW_IMAGE
```

2. そのコマンドの出力は標準`create-table`レスポンスを示し、この例のようになります。

```
{ "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/mykeyspace/
table/mytable" }
```

Amazon Keyspaces で既存のテーブルの CDC ストリームを有効にする

既存のテーブルの CDC ストリームを有効にするには、CQL で `ALTER TABLE`ステートメントを使用するか、 で `update-table` コマンドを使用するか AWS CLI、 コンソールを使用できます。

テーブル内の変更された行ごとに、Amazon Keyspaces は`cdc_specification`選択した `view_type`の に基づいて次の変更をキャプチャできます。

- `NEW_AND_OLD_IMAGES` – 変更前と変更後の両方のバージョンの行。これがデフォルトです。
- `NEW_IMAGE` – 変更後の行のバージョン。
- `OLD_IMAGE` – 変更前の行のバージョン。
- `KEYS_ONLY` – 変更された行のパーティションキーとクラスタリングキー。

ストリームにタグを付ける方法については、「」を参照してくださいthe section called “ストリームに新しいタグを追加する”。

> **Note**
>
> Amazon Keyspaces CDC には、ユーザーに代わって Amazon Keyspaces CDC ストリームから CloudWatch アカウントの `"cloudwatch:namespace": "AWS/`

Cassandra" にメトリクスデータを発行するサービスにリンクされたロール (AWSServiceRoleForAmazonKeyspacesCDC) が必要です。このロールは自動的に作成されます。詳細については、「the section called “CDC ストリーム”」を参照してください。

Cassandra Query Language (CQL)

CQL でストリーム (CDC ストリーム) を有効にする

を使用してALTER TABLE、既存のテーブルのストリームを有効にできます。

1. 次の例では、変更された行のパーティションキーとクラスタリングキーの変更のみをキャプチャするストリームを作成します。

```
ALTER TABLE mykeyspace.mytable
WITH cdc = TRUE
WITH CUSTOM_PROPERTIES={'cdc_specification': {'view_type': 'KEYS_ONLY'}};
```

2. ストリーム設定を確認するには、次のステートメントを使用できます。

```
SELECT keyspace_name, table_name, cdc, custom_properties FROM
 system_schema_mcs.tables WHERE keyspace_name = 'mykeyspace' AND table_name =
 'mytable';
```

ステートメントの出力は次のようになります。

```
 keyspace_name | table_name | cdc  | custom_properties
---------------+------------+------
+-------------------------------------------------------------------------------------------
    mykeyspace |    mytable | True | {'capacity_mode':
 {'last_update_to_pay_per_request_timestamp': '1741385897045',
 'throughput_mode': 'PAY_PER_REQUEST'}, 'cdc_specification':
 {'latest_stream_arn': 'arn:aws:cassandra:aws-region:111122223333:/keyspace/
mykeyspace/table/mytable/stream/2025-03-07T22:20:10.454', 'status': 'ENABLED',
 'view_type': 'KEYS_ONLY'}, 'encryption_specification': {'encryption_type':
 'AWS_OWNED_KMS_KEY'}, 'point_in_time_recovery': {'status': 'disabled'}}
```

CLI

を使用して CDC ストリームを作成する AWS CLI

1. 既存のテーブルのストリームを作成するには、次の構文を使用できます。

```
aws keyspaces update-table \
--keyspace-name 'mykeyspace' \
--table-name 'mytable' \
--cdc-specification status=ENABLED,viewType=NEW_AND_OLD_IMAGES
```

2. そのコマンドの出力は標準`create-table`レスポンスを示し、この例のようになります。

```
{ "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/mykeyspace/
table/mytable" }
```

Console

Amazon Keyspaces コンソールで CDC ストリームを有効にする

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、テーブルを選択し、リストからテーブルを選択します。
3. ストリームタブを選択します。
4. 編集 を選択してストリームを有効にします。
5. ストリームをオンにする を選択します。
6. ストリームの表示タイプを選択します。以下のオプションが利用できます。作成後にストリームのビュータイプを変更することはできません。

   - 新しいイメージと古いイメージ – Amazon Keyspaces は、変更前と変更後の両方のバージョンの行をキャプチャします。これがデフォルトです。
   - 新しいイメージ – Amazon Keyspaces は、変更後の行のバージョンのみをキャプチャします。
   - 古いイメージ – Amazon Keyspaces は、変更前の行のバージョンのみをキャプチャします。
   - プライマリキーのみ – Amazon Keyspaces は、変更された行のパーティションとクラスタリングキー列のみをキャプチャします。

7. 終了するには、変更の保存を選択します。

Amazon Keyspaces で CDC ストリームを無効にする

キースペースで CDC ストリームを無効にするには、CQL で ALTER TABLEステートメント、 で update-table コマンド AWS CLI、または コンソールを使用できます。

Cassandra Query Language (CQL)

CQL でストリーム (CDC ストリーム) を無効にする

1. ストリームを無効にするには、次のステートメントを使用します。

```
ALTER TABLE mykeyspace.mytable
WITH cdc = FALSE;
```

2. ストリームが無効になっていることを確認するには、次のステートメントを使用できます。

```
SELECT keyspace_name, table_name, cdc, custom_properties FROM
 system_schema_mcs.tables WHERE keyspace_name = 'mykeyspace' AND table_name =
 'mytable';
```

そのステートメントの出力は次のようになります。

```
 keyspace_name | table_name | cdc   | custom_properties
---------------+------------+-------
+-----------------------------------------------------------------------------------------------------
   mykeyspace  |   mytable  | False | {'capacity_mode':
 {'last_update_to_pay_per_request_timestamp': '1741385668642',
 'throughput_mode': 'PAY_PER_REQUEST'}, 'encryption_specification':
 {'encryption_type': 'AWS_OWNED_KMS_KEY'}, 'point_in_time_recovery': {'status':
 'disabled'}}
```

CLI

でストリーム (CDC ストリーム) を無効にする AWS CLI

1. ストリームを無効にするには、次のコマンドを使用します。

```
aws keyspaces update-table \
```

```
--keyspace-name 'mykeyspace' \
--table-name 'mytable' \
--cdc-specification status=DISABLED
```

2. コマンドの出力は次の例のようになります。

```
{
    "keyspaceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
my_keyspace/",
    "streamName": "my_stream"
}
```

Console

Amazon Keyspaces コンソールでストリーム (CDC ストリーム) を無効にする

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、テーブルを選択し、リストからテーブルを選択します。
3. ストリームタブを選択します。
4. [編集] を選択します。
5. 選択解除 ストリームをオンにします。
6. 変更を保存を選択してストリームを無効にします。

Amazon Keyspaces で CDC ストリームを表示する

キースペース内のすべてのストリームを表示または一覧表示するには、CQL `system_schema_mcs.streams`の ステートメントを使用してシステムキースペースのテーブルをクエリするか、 AWS CLIまたは コンソールで `get-stream`および `list-stream` コマンドを使用します。

必要なアクセス許可については、「the section called “アクセス許可の設定”」を参照してください。

Cassandra Query Language (CQL)

CQL で CDC ストリームを表示する

- テーブルの CDC ステータスをモニタリングするには、次のステートメントを使用できます。

```
SELECT custom_properties
FROM system_schema_mcs.tables
WHERE keyspace_name='my_keyspace' and table_name='my_table';
```

コマンドの出力は次のようになります。

```
...
custom_properties
--------------------------------------------------------------------------------
{'cdc_specification':{'status': 'Enabled', 'view_type': 'NEW_IMAGE',
 'latest_stream_arn': 'arn:aws:cassandra:aws-region:111122223333:/
keyspace/my_keyspace/table/my_table/stream/stream_label''}}
...
```

CLI

で CDC ストリームを表示する AWS CLI

1. この例では、テーブルのストリーム情報を表示する方法を示します。

```
aws keyspaces get-table \
--keyspace-name 'my_keyspace' \
--table-name 'my_table'
```

コマンドの出力は次のようになります。

```
{
    "keyspaceName": "my_keyspace",
    "tableName": "my_table",
    ... Other fields ...,
    "latestStreamArn": "arn:aws:cassandra:aws-region:111122223333:/
keyspace/my_keyspace/table/my_table/stream/stream_label",
    "cdcSpecification": {
```

```
        "status": "ENABLED",
        "viewType": "NEW_AND_OLD_IMAGES"
    }
}
```

2. 指定した のアカウント内のすべてのストリームを一覧表示できます AWS リージョン。次のコマンドは、その一例です。

```
aws keyspacesstreams list-streams --region aws-region
```

コマンドの出力は次のようになります。

```
{
    "Streams": [
        {
            "StreamArn": "arn:aws:cassandra:aws-region:111122223333:/keyspace/
ks_1/table/t1/stream/2023-05-11T21:21:33.291",
            "StreamLabel": "2023-05-11T21:21:33.291",
            "KeyspaceName": "ks_1"
            "TableName": "t1",
        },
        {
            "StreamArn": "arn:aws:cassandra:aws-region:111122223333:/keyspace/
ks_1/table/t2/stream/2023-05-11T21:21:33.291",
            "StreamLabel": "2023-05-11T21:21:33.291",
            "KeyspaceName": "ks_1"Create a keyspace with the name catalog. Note
                                that streams are not supported in multi-Region
 keyspaces.
            "TableName": "t2",
        },
        {
            "StreamArn": "arn:aws:cassandra:aws-region:111122223333:/keyspace/
ks_2/table/t1/stream/2023-05-11T21:21:33.291",
            "StreamLabel": "2023-05-11T21:21:33.291",
            "KeyspaceName": "ks_3"
            "TableName": "t1",
        }
    ]
}
```

3. 次のパラメータを使用して、特定のキースペースの CDC ストリームを一覧表示することもできます。

```
aws keyspacesstreams list-streams --keyspace-name ks_1 --region aws-region
```

コマンドの出力は次のようになります。

```
{
    "Streams": [
        {
            "StreamArn": "arn:aws:cassandra:aws-region:111122223333:/keyspace/
ks_1/table/t1/stream/2023-05-11T21:21:33.291",
            "StreamLabel": "2023-05-11T21:21:33.291",
            "KeyspaceName": "ks_1"
            "TableName": "t1",
        },
        {
            "StreamArn": "arn:aws:cassandra:aws-region:111122223333:/keyspace/
ks_1/table/t2/stream/2023-05-11T21:21:33.291",
            "StreamLabel": "2023-05-11T21:21:33.291",
            "KeyspaceName": "ks_1"
            "TableName": "t2",
        }
    ]
}
```

4. 次のパラメータを使用して、特定のテーブルの CDC ストリームを一覧表示することもできます。

```
aws keyspacesstreams list-streams --keyspace-name ks_1 --table-name t2 --
region aws-region
```

コマンドの出力は次のようになります。

```
{
    "Streams": [
        {
            "StreamArn": "arn:aws:cassandra:aws-region:111122223333:/keyspace/
ks_1/table/t2/stream/2023-05-11T21:21:33.291",
            "StreamLabel": "2023-05-11T21:21:33.291",
            "KeyspaceName": "ks_1"
            "TableName": "t2",
        }
    ]
```

```
}
```

Console

Amazon Keyspaces コンソールで CDC ストリームを表示する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインでテーブルを選択し、リストからテーブルを選択します。
3. ストリームタブを選択して、ストリームの詳細を確認します。

Amazon Keyspaces の CDC ストリームのレコードにアクセスする

ストリーム内のレコードにアクセスするには、Amazon Keyspaces Streams API を使用します。次のセクションでは、 を使用してレコードにアクセスする方法の例を示します AWS CLI。

必要なアクセス許可については、「the section called “アクセス許可の設定”」を参照してください。

を使用してストリーム内のレコードにアクセスする AWS CLI

1. Amazon Keyspaces Streams API を使用して、ストリームの変更レコードにアクセスできます。詳細については、「Amazon Keyspaces Streams API リファレンス」を参照してください。ストリーム内のシャードを取得するには、次の例に示すように get-stream API を使用できます。

```
aws keyspacesstreams get-stream \
--stream-arn 'arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/table/
mytable/stream/STREAM_LABEL'
```

次は出力の例です。

```
{
   "StreamArn": "arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/
table/mytable/stream/2023-05-11T21:21:33.291",
   "StreamStatus": "ENABLED",
   "StreamViewType": "NEW_AND_OLD_IMAGES",
   "CreationRequestDateTime": "<CREATION_TIME>",
   "KeyspaceName": "mykeyspace",
   "TableName": "mytable",
   "StreamLabel": "2023-05-11T21:21:33.291",
```

```
    "Shards": [
        {
            "SequenceNumberRange": {
                "EndingSequenceNumber": "<END_SEQUENCE_NUMBER>",
                "StartingSequenceNumber": "<START_SEQUENCE_NUMBER>"
            },
            "ShardId": "<SHARD_ID>"
        },
    ]
}
```

2. ストリームからレコードを取得するには、まずレコードにアクセスするための開始点を提供するイテレーターを取得します。これを行うには、前のステップで API によって返された CDC ストリーム内のシャードを使用できます。イテレーターを収集するには、 `get-shard-iterator` API を使用できます。この例では、シャードの最後のトリミングポイントまたは先頭から が取得TRIM_HORIZONするタイプのイテレーターを使用します)。

```
aws keyspacesstreams get-shard-iterator \
--stream-arn 'arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/table/
mytable/stream/STREAM_LABEL' \
--shard-id 'SHARD_ID' \
--shard-iterator-type 'TRIM_HORIZON'
```

コマンドの出力は次の例のようになります。

```
{
    "ShardIterator": "<SHARD_ITERATOR>"
}
```

3. `get-records` API を使用して CDC レコードを取得するには、最後のステップで返されたイテレーターを使用できます。次のコマンドは、その一例です。

```
aws keyspacesstreams get-records \
--shard-iterator 'SHARD_ITERATOR' \
--limit 100
```

Kinesis Client Library (KCL) を使用して Amazon Keyspaces ストリームを処理する

このトピックでは、Kinesis Client Library (KCL) を使用して Amazon Keyspaces 変更データキャプチャ (CDC) ストリームのデータを使用および処理する方法について説明します。

Amazon Keyspaces Streams API を直接使用する代わりに、Kinesis Client Library (KCL) を使用すると、次のような多くの利点があります。

- シャード系統の追跡とイテレーター処理が組み込まれています。
- ワーカー間の自動負荷分散。
- ワーカーの障害に対する耐障害性と復旧。
- 処理の進行状況を追跡するためのチェックポイント。
- ストリーム容量の変化への適応。
- CDC レコードを処理するための簡素化された分散コンピューティング。

次のセクションでは、Kinesis Client Library (KCL) を使用してストリームを処理する理由と方法について説明します。また、KCL を使用して Amazon Keyspaces CDC ストリームを処理する例を示します。

料金については、「Amazon Keyspaces (for Apache Cassandra) pricing (Amazon Keyspaces (Apache Cassandra 向け) の料金)」を参照してください。

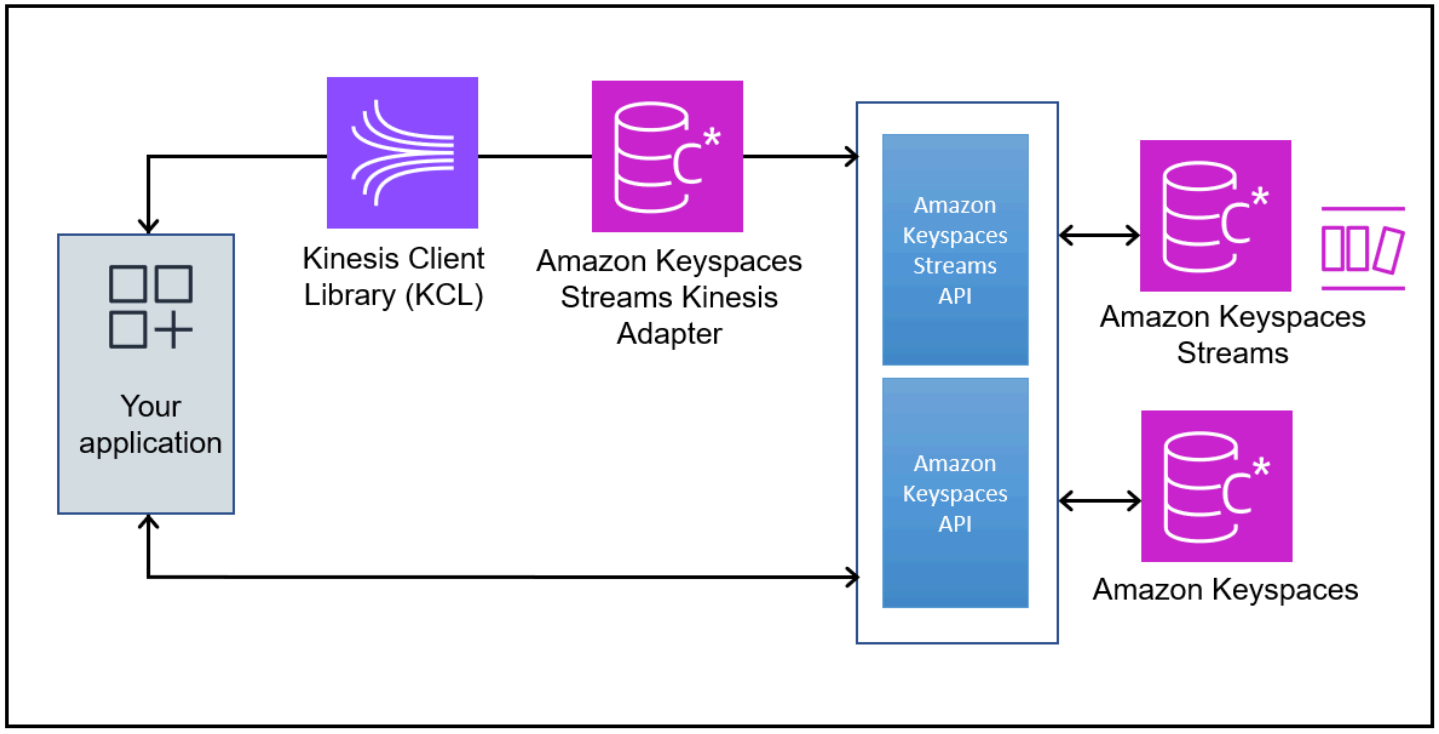
Kinesis Client Library (KCL) とは何ですか?

Kinesis Client Library (KCL) は、ストリームからのデータの消費と処理のプロセスを簡素化するために設計されたスタンドアロンの Java ソフトウェアライブラリです。KCL は、分散コンピューティングに関連する複雑なタスクの多くを処理し、ストリームデータを処理する際にビジネスロジックの実装に集中できます。KCL は、複数のワーカー間の負荷分散、ワーカーの障害への対応、処理されたレコードのチェックポイント作成、ストリーム内のシャード数の変化への対応などのアクティビティを管理します。

Amazon Keyspaces CDC ストリームを処理するには、KCL にある設計パターンを使用して、ストリームシャードとストリームレコードを操作できます。KCL は、低レベルの Kinesis Data Streams API の上で役に立つ抽象化を提供することによりコーディングを簡素化します。KCL の詳細については、Amazon Kinesis Data Streams デベロッパーガイド」の「KCL を使用したコンシューマーの開発」を参照してください。

KCL を使用してアプリケーションを記述するには、Amazon Keyspaces Streams Kinesis Adapter を使用します。Kinesis Adapter は Kinesis Data Streams インターフェイスを実装しているため、KCL を使用して Amazon Keyspaces ストリームからのレコードを消費および処理できます。Amazon Keyspaces ストリーム Kinesis Adapter をセットアップしてインストールする方法については、GitHub リポジトリを参照してください。

次の図は、これらのライブラリが相互にどのように相互作用するかを示しています。



基盤となるライブラリの新しいバージョン、セキュリティの改善、バグ修正を組み込むために KCL が頻繁に更新されます。既知の問題を回避し、すべての最新の改善点を活用するには、最新バージョンの KCL を使用することをお勧めします。最新の KCL バージョンを確認するには、「KCL GitHub リポジトリ」を参照してください。

KCL の概念

KCL を使用してコンシューマーアプリケーションを実装する前に、次の概念を理解しておく必要があります。

KCL コンシューマーアプリケーション

KCL コンシューマーアプリケーションは、Amazon Keyspaces CDC ストリームからのデータを処理するプログラムです。KCL は、コンシューマーアプリケーションコードと Amazon Keyspaces CDC ストリームの間の仲介として機能します。

ワーカー

ワーカーは、Amazon Keyspaces CDC ストリームからのデータを処理する KCL コンシューマーアプリケーションの実行ユニットです。アプリケーションは、複数のインスタンスに分散された複数のワーカーを実行できます。

レコードプロセッサ

レコードプロセッサは、Amazon Keyspaces CDC ストリームのシャードからのデータを処理するアプリケーション内のロジックです。レコードプロセッサは、管理するシャードごとにワーカーによってインスタンス化されます。

リース

リースは、シャードの処理責任を表します。ワーカーはリースを使用して、どのワーカーがどのシャードを処理しているかを調整します。KCL は、Amazon DynamoDB のテーブルにリースデータを保存します。

チェックポイント

チェックポイントは、レコードプロセッサがレコードを正常に処理したシャード内の位置のレコードです。チェックポイントを使用すると、ワーカーが失敗した場合にアプリケーションが中断した場所から処理を再開できます。

Amazon Keyspaces Kinesis アダプターを使用すると、Amazon Keyspaces ストリームエンドポイントにシームレスに送信される API コールを使用して、KCL インターフェイスに対して開発を開始できます。利用可能なエンドポイントのリストについては、「the section called “CDC ストリームエンドポイントにアクセスする”」を参照してください。

アプリケーションは起動時に KCL をコールしてワーカーをインスタンス化します。ストリーム記述子や AWS 認証情報、指定したレコードプロセッサクラスの名前など、アプリケーションの設定情報をワーカーに提供する必要があります。レコードプロセッサでコードを実行すると、ワーカーは次のタスクを実行します。

- ストリームに接続する
- ストリーミング内のシャードを列挙します。
- シャードと他のワーカー (存在する場合) の関連付けを調整する
- レコードプロセッサで管理する各シャードのレコードプロセッサをインスタンス化する
- ストリーミングからレコードを取得します。
- 対応するレコードプロセッサにレコードを送信する
- 処理されたレコードのチェックポイントを作成する
- ワーカーのインスタンス数が変化したときに、シャードとワーカーの関連付けを調整する
- シャードが分割されたときに、シャードとワーカーの関連付けを調整します。

Amazon Keyspaces CDC ストリーム用の KCL コンシューマーアプリケーションの実装

このトピックでは、Amazon Keyspaces CDC ストリームを処理する KCL コンシューマーアプリケーションを実装するためのstep-by-stepガイドを提供します。

1. 前提条件: 開始する前に、以下を確認してください。
   - CDC ストリームを含む Amazon Keyspaces テーブル
   - IAM プリンシパルが Amazon Keyspaces CDC ストリームにアクセスし、KCL ストリーム処理用の DynamoDB テーブルを作成してアクセスするために必要な IAM アクセス許可、および CloudWatch にメトリクスを発行するためのアクセス許可。詳細とポリシーの例については、「」を参照してくださいthe section called “KCL のアクセス許可”。
   - 有効な AWS 認証情報がローカル設定で設定されていることを確認します。詳細については、「the section called “ アクセスキーを管理する”」を参照してください。
   - Java Development Kit (JDK) 8 以降
   - Github の Readme に記載されている要件。
2. このステップでは、KCL 依存関係をプロジェクトに追加します。Maven の場合は、pom.xml に以下を追加します。

```
<dependencies>
        <dependency>
            <groupId>software.amazon.kinesis</groupId>
            <artifactId>amazon-kinesis-client</artifactId>
            <version>3.1.0</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.keyspaces</groupId>
            <artifactId>keyspaces-streams-kinesis-adapter</artifactId>
            <version>1.0.0</version>
        </dependency>
    </dependencies>
```

> **Note**
>
> 常に KCL の最新バージョンを KCL GitHub リポジトリで確認してください。

3. レコードプロセッサインスタンスを生成するファクトリクラスを作成します。

```
import software.amazon.awssdk.services.keyspacesstreams.model.Record;
```

```
import
 software.amazon.keyspaces.streamsadapter.adapter.KeyspacesStreamsClientRecord;
import
 software.amazon.keyspaces.streamsadapter.model.KeyspacesStreamsProcessRecordsInput;
import
 software.amazon.keyspaces.streamsadapter.processor.KeyspacesStreamsShardRecordProcessor;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.RecordProcessorCheckpointer;

public class RecordProcessor implements KeyspacesStreamsShardRecordProcessor {
    private String shardId;

    @Override
    public void initialize(InitializationInput initializationInput) {
        this.shardId = initializationInput.shardId();
        System.out.println("Initializing record processor for shard: " + shardId);
    }

    @Override
    public void processRecords(KeyspacesStreamsProcessRecordsInput
 processRecordsInput) {
        try {
            for (KeyspacesStreamsClientRecord record :
 processRecordsInput.records()) {
                Record keyspacesRecord = record.getRecord();
                System.out.println("Received record: " + keyspacesRecord);
            }

            if (!processRecordsInput.records().isEmpty()) {
                RecordProcessorCheckpointer checkpointer =
 processRecordsInput.checkpointer();
                try {
                    checkpointer.checkpoint();
                    System.out.println("Checkpoint successful for shard: " +
 shardId);
                } catch (Exception e) {
                    System.out.println("Error while checkpointing for shard: " +
 shardId + " " + e);
                }
            }
        } catch (Exception e) {
```

```
            System.out.println("Error processing records for shard: " + shardId + "
 " + e);
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        System.out.println("Lease lost for shard: " + shardId);
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        System.out.println("Shard ended: " + shardId);
        try {
            // This is required. Checkpoint at the end of the shard
            shardEndedInput.checkpointer().checkpoint();
            System.out.println("Final checkpoint successful for shard: " +
 shardId);
        } catch (Exception e) {
            System.out.println("Error while final checkpointing for shard: " +
 shardId + " " + e);
            throw new RuntimeException("Error while final checkpointing", e);
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        System.out.println("Shutdown requested for shard " + shardId);
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (Exception e) {
            System.out.println("Error while checkpointing on shutdown for shard: "
 + shardId + " " + e);
        }
    }
}
```

4. 次の例に示すように、レコードファクトリを作成します。

```
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;
```

```
public class RecordProcessorFactory implements ShardRecordProcessorFactory {
    private final Queue<RecordProcessor> processors = new
 ConcurrentLinkedQueue<>();

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        System.out.println("Creating new RecordProcessor");
        RecordProcessor processor = new RecordProcessor();
        processors.add(processor);
        return processor;
    }
}
```

5. このステップでは、KCLv3 と Amazon Keyspaces アダプターを設定するベースクラスを作成します。

```
import com.example.KCLExample.utils.RecordProcessorFactory;
import
 software.amazon.keyspaces.streamsadapter.AmazonKeyspacesStreamsAdapterClient;
import software.amazon.keyspaces.streamsadapter.StreamsSchedulerFactory;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ExecutionException;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableResponse;
import software.amazon.awssdk.services.keyspacesstreams.KeyspacesStreamsClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
import software.amazon.kinesis.coordinator.CoordinatorConfig;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.leases.LeaseManagementConfig;
import software.amazon.kinesis.processor.ProcessorConfig;
import software.amazon.kinesis.processor.StreamTracker;
import software.amazon.kinesis.retrieval.polling.PollingConfig;

public class KCLTestBase {
```

```
    protected KeyspacesStreamsClient streamsClient;
    protected KinesisAsyncClient adapterClient;
    protected DynamoDbAsyncClient dynamoDbAsyncClient;
    protected CloudWatchAsyncClient cloudWatchClient;
    protected Region region;
    protected RecordProcessorFactory recordProcessorFactory;
    protected Scheduler scheduler;
    protected Thread schedulerThread;

    public void baseSetUp() {
        recordProcessorFactory = new RecordProcessorFactory();
        setupKCLBase();
    }

    protected void setupKCLBase() {
        region = Region.US_EAST_1;

        streamsClient = KeyspacesStreamsClient.builder()
                .region(region)
                .build();
        adapterClient = new AmazonKeyspacesStreamsAdapterClient(
                streamsClient,
                region);
        dynamoDbAsyncClient = DynamoDbAsyncClient.builder()
                .region(region)
                .build();
        cloudWatchClient = CloudWatchAsyncClient.builder()
                .region(region)
                .build();
    }

    protected void startScheduler(Scheduler scheduler) {
        this.scheduler = scheduler;
        schedulerThread = new Thread(() -> scheduler.run());
        schedulerThread.start();
    }

    protected void shutdownScheduler() {
        if (scheduler != null) {
            scheduler.shutdown();
            try {
                schedulerThread.join(30000);
            } catch (InterruptedException e) {
```

```
                System.out.println("Error while shutting down scheduler " + e);
            }
        }
    }

    protected Scheduler createScheduler(String streamArn, String leaseTableName) {
        String workerId = "worker-" + System.currentTimeMillis();

        // Create ConfigsBuilder
        ConfigsBuilder configsBuilder = createConfigsBuilder(streamArn, workerId,
leaseTableName);

        // Configure retrieval config for polling
        PollingConfig pollingConfig = new PollingConfig(streamArn, adapterClient);

        // Create the Scheduler
        return StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),

configsBuilder.retrievalConfig().retrievalSpecificConfig(pollingConfig),
                streamsClient,
                region
        );
    }

    private ConfigsBuilder createConfigsBuilder(String streamArn, String workerId,
String leaseTableName) {
        ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamArn,
                leaseTableName,
                adapterClient,
                dynamoDbAsyncClient,
                cloudWatchClient,
                workerId,
                recordProcessorFactory);

        configureCoordinator(configsBuilder.coordinatorConfig());
        configureLeaseManagement(configsBuilder.leaseManagementConfig());
        configureProcessor(configsBuilder.processorConfig());
```

```
        configureStreamTracker(configsBuilder, streamArn);

        return configsBuilder;
    }

    private void configureCoordinator(CoordinatorConfig config) {
        config.skipShardSyncAtWorkerInitializationIfLeasesExist(true)
                .parentShardPollIntervalMillis(1000)
                .shardConsumerDispatchPollIntervalMillis(500);
    }

    private void configureLeaseManagement(LeaseManagementConfig config) {
        config.shardSyncIntervalMillis(0)
                .leasesRecoveryAuditorInconsistencyConfidenceThreshold(0)
                .leasesRecoveryAuditorExecutionFrequencyMillis(5000)
                .leaseAssignmentIntervalMillis(1000L);
    }

    private void configureProcessor(ProcessorConfig config) {
        config.callProcessRecordsEvenForEmptyRecordList(true);
    }

    private void configureStreamTracker(ConfigsBuilder configsBuilder, String
streamArn) {
        StreamTracker streamTracker =
StreamsSchedulerFactory.createSingleStreamTracker(
                streamArn,

InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON)
        );
        configsBuilder.streamTracker(streamTracker);
    }

    public void deleteAllDdbTables(String baseTableName) {
        List<String> tablesToDelete = Arrays.asList(
                baseTableName,
                baseTableName + "-CoordinatorState",
                baseTableName + "-WorkerMetricStats"
        );

        for (String tableName : tablesToDelete) {
            deleteTable(tableName);
        }
    }
```

```
    private void deleteTable(String tableName) {
        DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            DeleteTableResponse response =
 dynamoDbAsyncClient.deleteTable(deleteTableRequest).get();
            System.out.println("Table deletion response " + response);
        } catch (InterruptedException | ExecutionException e) {
            System.out.println("Error deleting table: " + tableName + " " + e);
        }
    }
}
```

6. このステップでは、アプリケーションのレコードプロセッサクラスを実装して、変更イベントの処理を開始します。

```
 import software.amazon.kinesis.coordinator.Scheduler;

public class KCLTest {

    private static final int APP_RUNTIME_SECONDS = 1800;
    private static final int SLEEP_INTERNAL_MS = 60*1000;

    public static void main(String[] args) {
        KCLTestBase kclTestBase;

        kclTestBase = new KCLTestBase();
        kclTestBase.baseSetUp();

        // Create and start scheduler
        String leaseTableName = generateUniqueApplicationName();

        // Update below to your Stream ARN
        String streamArn = "arn:aws:cassandra:us-east-1:759151643516:/keyspace/
cdc_sample_test/table/test_kcl_bool/stream/2025-07-01T15:52:57.529";
        Scheduler scheduler = kclTestBase.createScheduler(streamArn,
 leaseTableName);
        kclTestBase.startScheduler(scheduler);
```

```
        // Wait for specified time before shutting down - KCL applications are
 designed to run forever, however in this
        // example we will shut it down after APP_RUNTIME_SECONDS
        long startTime = System.currentTimeMillis();
        long endTime = startTime + (APP_RUNTIME_SECONDS * 1000);
        while (System.currentTimeMillis() < endTime) {
            try {
                // Print and sleep every minute
                Thread.sleep(SLEEP_INTERNAL_MS);
                System.out.println("Application is running");
            } catch (InterruptedException e) {
                System.out.println("Interrupted while waiting for records");
                Thread.currentThread().interrupt();
                break;
            }
        }

        // Stop the scheduler
        kclTestBase.shutdownScheduler();
        kclTestBase.deleteAllDdbTables(leaseTableName);
    }

    public static String generateUniqueApplicationName() {
        String timestamp = String.valueOf(System.currentTimeMillis());
        String randomString = java.util.UUID.randomUUID().toString().substring(0,
 8);
        return String.format("KCL-App-%s-%s", timestamp, randomString);
    }
}
```

ベストプラクティス

Amazon Keyspaces CDC ストリームで KCL を使用する場合は、次のベストプラクティスに従ってください。

エラー処理

- レコードプロセッサに堅牢なエラー処理を実装して、例外を適切に処理します。一時的な障害に対して再試行ロジックを実装することを検討してください。

チェックポイントの頻度

チェックポイントの頻度を調整して、適切な進行状況の追跡を確保しながら、重複処理を最小限に抑えます。チェックポイントが頻繁すぎるとパフォーマンスに影響する可能性がありますが、チェックポイントが頻繁すぎると、ワーカーが失敗した場合に再処理が増える可能性があります。

ワーカースケーリング

CDC ストリーム内のシャードの数に基づいてワーカーの数をスケールします。まず、シャードごとにワーカーを 1 人配置することをお勧めしますが、処理要件に基づいて調整する必要がある場合があります。

モニタリング

KCL が提供する CloudWatch メトリクスを使用して、コンシューマーアプリケーションのヘルスとパフォーマンスをモニタリングします。主なメトリクスには、処理レイテンシー、チェックポイント経過時間、リース数が含まれます。

テスト

ワーカーの障害、ストリームのリシャーディング、さまざまな負荷条件などのシナリオを含め、コンシューマーアプリケーションを徹底的にテストします。

Java 以外の言語で KCL を使用する

KCL は主に Java ライブラリですが、MultiLangDaemon を通じて他のプログラミング言語で使用できます。MultiLangDaemon は、非 Java レコードプロセッサと KCL 間のインタラクションを管理する Java ベースのデーモンです。

KCL では、次の言語がサポートされています。

- Python
- Ruby
- Node.js
- .NET

Java 以外の言語で KCL を使用する方法の詳細については、KCL MultiLangDaemon 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

このセクションでは、Amazon Keyspaces CDC ストリームで KCL を使用する際に発生する可能性がある一般的な問題の解決策を示します。

処理が遅い

コンシューマーアプリケーションでレコードの処理が遅い場合は、次の点を考慮してください。

- ワーカーインスタンスの数を増やす
- レコード処理ロジックの最適化
- ダウンストリームシステムのボトルネックの確認

重複処理

レコードの処理が重複している場合は、チェックポイントロジックを確認してください。レコードを正常に処理した後、チェックポイントに入っていることを確認します。

ワーカーの障害

ワーカーが頻繁に失敗する場合は、以下を確認してください。

- リソース制約 (CPU、メモリ)
- ネットワーク接続の問題
- アクセス許可の問題

リーステーブルの問題

KCL リーステーブルで問題が発生した場合：

- アプリケーションに Amazon Keyspaces テーブルにアクセスするための適切なアクセス許可があることを確認します。
- テーブルに十分なプロビジョニングされたスループットがあることを確認する

Amazon Keyspaces でのパーティショナーの操作

Apache Cassandra では、パーティショナーがクラスター内のどのノードにデータを保存するかを制御します。パーティショナーは、パーティションキーのハッシュ値を使用して数値トークンを作成します。Cassandra はこのトークンを使用してデータをノードに分散します。クライアントはこれらのトークンを `SELECT` 操作や `WHERE` 句に使用して、読み取り操作と書き込み操作を最適化することもできます。たとえば、クライアントは、各並列ジョブでクエリする個別のトークン範囲を指定することで、大きなテーブルに対して効率的に並列クエリを実行できます。

Amazon Keyspaces には 3 つの異なるパーティショナーがあります。

Murmur3Partitioner (デフォルト)

Apache Cassandra 対応 `Murmur3Partitioner`。`Murmur3Partitioner` は Amazon Keyspaces と Cassandra 1.2 以降のバージョンのデフォルトの Cassandra パーティショナーです。

RandomPartitioner

Apache Cassandra 対応 `RandomPartitioner`。`RandomPartitioner` は Cassandra 1.2 より前のバージョンのデフォルトの Cassandra パーティショナーです。

Keyspaces デフォルトパーティショナー

`DefaultPartitioner` は、`RandomPartitioner` と同じ `token` 関数結果を返します。

パーティショナー設定は、アカウントレベルでリージョンごとに適用されます。たとえば、米国東部 (バージニア北部) のパーティショナーを変更すると、その変更はこのリージョンの同じアカウントのすべてのテーブルに適用されます。パーティショナーはいつでも安全に変更できます。設定の変更は約 10 分で完了します。パーティショナー設定を変更するときに Amazon Keyspaces データをリロードする必要はありません。クライアントは次回の接続時に自動的に新しいパーティショナー設定を使用します。

Amazon Keyspaces でパーティショナーを変更する方法

パーティショナーは、 AWS Management Console または Cassandra クエリ言語 (CQL) を使用して変更できます。

AWS Management Console

Amazon Keyspaces コンソールを使用してパーティショナーを変更するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[設定] を選択します。
3. [設定] ページで [パーティショナーを編集] に移動します。
4. Cassandra のバージョンと互換性のあるパーティショナーを選択します。パーティショナーの変更が約 10 分で反映されます。

> **Note**
>
> 設定の変更が完了したら、Amazon Keyspaces への接続を切断して再接続し、新しいパーティショナーの使用をリクエストする必要があります。

Cassandra Query Language (CQL)

1. アカウントにどのパーティショナーが設定されているかを確認するには、次のクエリを使用できます。

```
SELECT partitioner from system.local;
```

パーティショナーが変更されていない場合、クエリの出力は次のようになります。

```
partitioner
-------------------------------------------
com.amazonaws.cassandra.DefaultPartitioner
```

2. パーティショナーを Murmur3 パーティショナーに更新するには、次のステートメントを使用できます。

```
UPDATE system.local set
 partitioner='org.apache.cassandra.dht.Murmur3Partitioner' where key='local';
```

3. この設定の変更は約 10 分で完了します。パーティショナーが設定されたことを確認するには、SELECT クエリをもう一度実行できます。最終的な読み込みの一貫性により、最近のパーティショナーの変更の結果がまだ反映されていないことがあります。少し時間がたってから読み込みリクエストを繰り返すと、応答で最新のデータが返されます。

```
SELECT partitioner from system.local;
```

> **Note**
>
> リクエストが新しいパーティショナーを使用するように、Amazon Keyspaces への接続を切断して再接続する必要があります。

Amazon Keyspaces でのクライアント側のタイムスタンプ

Amazon Keyspaces では、クライアントサイドのタイムスタンプは Cassandra 互換のタイムスタンプです。テーブル内の各セルに保存されます。クライアントアプリケーションがクライアントサイドのタイムスタンプで書き込みの順序を決定できるようにして競合を解決します。たとえば、グローバルに分散しているアプリケーションのクライアントが同じデータを更新する場合、クライアントサイドのタイムスタンプにはクライアントサイドで更新が行われた順序が残ります。Amazon Keyspaces はこれらのタイムスタンプを使用して書き込みを処理します。

Amazon Keyspaces のクライアント側のタイムスタンプは全面的に管理されています。コンパクション戦略やクリーンアップ戦略などの低レベルのシステム設定は管理する必要はありません。

データを削除すると、行は削除対象としてトゥームストーンのマークが付きます。Amazon Keyspaces では、アプリケーションのパフォーマンスや可用性に影響を与えることなく、トゥームストーン化されたデータは (通常は 10 日以内に) 自動的に削除されます。トゥームストーン化されたデータは、データ操作言語 (DML) ステートメントでは使用できなくなります。トゥームストーン化されたデータがある行に対して引き続き読み取りと書き込みを実行すると、トゥームストーン化されたデータは、ストレージから削除されるまで、ストレージ、読み取りキャパシティユニット (RCU) および書き込みキャパシティユニット (WCU) にカウントされ続けます。

テーブルのクライアントサイドのタイムスタンプを有効にすると、データ操作言語 (DML) CQL クエリの `USING TIMESTAMP` 句でタイムスタンプを指定できます。詳細については、「the section called “クエリでのクライアント側のタイムスタンプの使用”」を参照してください。CQL クエリでタイムスタンプを指定しない場合、Amazon Keyspaces はクライアントドライバーから渡されたタイムスタンプを使用します。クライアントドライバーでタイムスタンプが提供されない場合、タイムスタンプを `NULL` にすることはできないため、Amazon Keyspaces はセルごとにタイムスタンプを自動的に割り当てます。タイムスタンプをクエリするには、DML ステートメント内の `WRITETIME` 関数を使用できます。

Amazon Keyspaces では、クライアントサイドのタイムスタンプを有効にしても追加料金はかかりません。ただし、クライアントサイドのタイムスタンプでは、行内の値ごとに追加のデータを保存して書き込みます。これにより、ストレージの使用量が増え、場合によってはスループットの使用量が増える可能性があります。料金の詳細については、「Amazon Keyspaces (Apache Cassandra 向け) の料金」を参照してください。

Amazon Keyspaces でクライアント側のタイムスタンプをオンにすると、すべての行のすべての列にタイムスタンプが格納されます。これらのタイムスタンプで約 20 ~ 40 バイト (データによって異なります) が使用され、行のストレージとスループットのコストに影響します。これらのメタデー

タバイトはまた、1 MB の行サイズクォータにカウントされます。(行のサイズが 1 MB 未満になるように) ストレージスペース全体の増加量を決定するときは、テーブルの列数と各行のコレクション要素の数を考慮してください。たとえば、テーブルに 20 列あり、各列に 40 バイトのデータが格納されている場合、行のサイズは 800 バイトから 1200 バイトに増加します。行のサイズを見積もる方法の詳細については、「the section called “行サイズの推定”」を参照してください。この例では、ストレージ用の余分な 400 バイトに加えて、1 回の書き込みで消費される書き込みキャパシティーユニット (WCU) の数が 1 WCU から 2 WCU に増えます。読み取りキャパシティと書き込みキャパシティの計算方法の詳細については、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。

テーブルのクライアント側のタイムスタンプをオンにすると、オフにすることはできません。

クエリでクライアント側のタイムスタンプを使用する方法については、「the section called “クエリでのクライアント側のタイムスタンプの使用”」を参照してください。

トピック

- Amazon Keyspaces のクライアント側のタイムスタンプを AWS サービスに統合する方法
- クライアント側のタイムスタンプが有効な新規テーブルを Amazon Keyspaces で作成する
- Amazon Keyspaces でテーブルに対してクライアント側のタイムスタンプを設定する
- Amazon Keyspaces のクエリでクライアント側のタイムスタンプを使用する

Amazon Keyspaces のクライアント側のタイムスタンプを AWS サービスに統合する方法

次のクライアント側のメトリクスは Amazon CloudWatch で継続的モニタリングを有効にするために使用できます。

- `SystemReconciliationDeletes` — トゥームストーン化したデータを削除するために必要な削除操作の数。

CloudWatch のメトリクスのモニタリング方法については、「the section called “CloudWatch によるモニタリング”」を参照してください。

を使用すると AWS CloudFormation、Amazon Keyspaces テーブルの作成時にクライアント側のタイムスタンプを有効にできます。詳細については、AWS CloudFormation ユーザーガイドをご参照ください。

クライアント側のタイムスタンプが有効な新規テーブルを Amazon Keyspaces で作成する

Amazon Keyspaces、Cassandra クエリ言語 (CQL) AWS Management Console、または を使用して、クライアント側のタイムスタンプを有効にした新しい Amazon Keyspaces テーブルを作成するには、次の例に従います。 AWS Command Line Interface

Console

クライアント側のタイムスタンプが有効な新規テーブルを作成する (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで [Tables] (テーブル) を選択して、[Create table (テーブルの作成)] を選択します。
3. [Table details (テーブルの詳細)] セクションの [Create table (テーブルの作成)] ページで、キースペースを選択し、新しいテーブルに名前を付けます。
4. [Schema (スキーマ)] セクションで、テーブルのスキーマを作成します。
5. [Table settings (テーブルの設定)] セクションで、[Customize settings (設定のカスタマイズ)] を選択します。
6. [クライアント側のタイムスタンプ] に進みます。

   [クライアント側のタイムスタンプを有効にする] を選択して、テーブルのクライアント側のタイムスタンプを有効にします。
7. [Create table (テーブルの作成)] を選択します。テーブルはクライアント側のタイムスタンプが有効の状態で作成されます。

Cassandra Query Language (CQL)

CQL を使用して新規テーブルを作成する

1. クライアント側のタイムスタンプが有効な新規テーブルを CQL で作成するには、次の例を使用します。

```
CREATE TABLE my_keyspace.my_table (
   userid uuid,
   time timeuuid,
```

```
    subject text,
    body text,
    user inet,
    PRIMARY KEY (userid, time)
) WITH CUSTOM_PROPERTIES = {'client_side_timestamps': {'status': 'enabled'}};
```

2. 新しいテーブルのクライアント側のタイムスタンプ設定を確認するには、次の例に示すように、SELECT ステートメントで、custom_properties を見直します。

```
SELECT custom_properties from system_schema_mcs.tables where keyspace_name =
 'my_keyspace' and table_name = 'my_table';
```

このステートメントの出力には、クライアント側のタイムスタンプのステータスが表示されます。

```
'client_side_timestamps': {'status': 'enabled'}
```

AWS CLI

を使用して新しいテーブルを作成する AWS CLI

1. クライアント側のタイムスタンプが有効な新規テーブルを作成するには、次の例を使用します。

```
./aws keyspaces create-table \
--keyspace-name my_keyspace \
--table-name my_table \
--client-side-timestamps 'status=ENABLED' \
--schema-definition 'allColumns=[{name=id,type=int},{name=date,type=timestamp},
{name=name,type=text}],partitionKeys=[{name=id}]'
```

2. 新しいテーブルのクライアント側のタイムスタンプが有効になっていることを確認するには、次のコードを実行します。

```
./aws keyspaces get-table \
--keyspace-name my_keyspace \
--table-name my_table
```

この例のような出力が得られます。

```
{
    "keyspaceName": "my_keyspace",
    "tableName": "my_table",
    "resourceArn": "arn:aws:cassandra:us-east-2:555555555555:/keyspace/
my_keyspace/table/my_table",
    "creationTimestamp": 1662681206.032,
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "id",
                "type": "int"
            },
            {
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": 1662681206.032
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "clientSideTimestamps": {
        "status": "ENABLED"
    },
```

```
    "ttl": {
        "status": "ENABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    }
}
```

Amazon Keyspaces でテーブルに対してクライアント側のタイムスタンプを設定する

Amazon Keyspaces、Cassandra クエリ言語 (CQL) AWS Management Console、または を使用して、既存のテーブルのクライアント側のタイムスタンプを有効にするには、次の例に従います AWS Command Line Interface。

Console

既存のテーブル (コンソール) のクライアント側のタイムスタンプを有効にするには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. 更新するテーブルを選択し、次に [Additional settings (追加設定)] タブを選択します。
3. [追加設定] タブの [クライアント側のタイムスタンプを変更] に移動し、[クライアント側のタイムスタンプを有効にする] を選択します。
4. [変更を保存] を選択してテーブルの設定を変更します。

Cassandra Query Language (CQL)

CQL ステートメントを使用する

1. CQL の ALTER TABLE ステートメントを使用して、既存のテーブルに対してクライアント側のタイムスタンプを有効にします。

```
ALTER TABLE my_table WITH custom_properties = {'client_side_timestamps':
 {'status': 'enabled'}};;
```

2. 新しいテーブルのクライアント側のタイムスタンプ設定を確認するには、次の例に示すように、SELECT ステートメントで、custom_properties を見直します。

```
SELECT custom_properties from system_schema_mcs.tables where keyspace_name =
 'my_keyspace' and table_name = 'my_table';
```

このステートメントの出力には、クライアント側のタイムスタンプのステータスが表示されます。

```
'client_side_timestamps': {'status': 'enabled'}
```

AWS CLI

の使用 AWS CLI

1. AWS CLI 次の例を使用して、 を使用して既存のテーブルのクライアント側のタイムスタンプを有効にできます。

```
./aws keyspaces update-table \
--keyspace-name my_keyspace \
--table-name my_table \
--client-side-timestamps 'status=ENABLED'
```

2. テーブルのクライアント側のタイムスタンプが有効になっていることを確認するには、次のコードを実行します。

```
./aws keyspaces get-table \
--keyspace-name my_keyspace \
--table-name my_table
```

この例のような出力が表示され、クライアント側のタイムスタンプのステータスが ENABLED となっているはずです。

```
{
    "keyspaceName": "my_keyspace",
    "tableName": "my_table",
    "resourceArn": "arn:aws:cassandra:us-east-2:555555555555:/keyspace/
my_keyspace/table/my_table",
    "creationTimestamp": 1662681312.906,
```

```
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "id",
                "type": "int"
            },
            {
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": 1662681312.906
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "clientSideTimestamps": {
        "status": "ENABLED"
    },
    "ttl": {
        "status": "ENABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    }
```

```
}
```

Amazon Keyspaces のクエリでクライアント側のタイムスタンプを使用する

クライアント側のタイムスタンプを有効にすると、USING TIMESTAMP 節の INSERT、UPDATE、および DELETE ステートメントにそのタイムスタンプを句とともに渡すことができます。

タイムスタンプ値は、epoch という標準基準時刻 1970 年 1 月 1 日 00:00:00 GMT からのマイクロ秒数をを表すbigint です。クライアントから提供されるタイムスタンプは、現在のウォールクロックタイムから過去 2 日間と未来の 5分間の範囲とします。

Amazon Keyspaces は、データの存続期間中、タイムスタンプのメタデータを保持します。WRITETIME 関数で、数年前に発生したタイムスタンプを検索できます。構文の詳細については、「the section called “DML ステートメント”」を参照してください。

次の CQL ステートメントは、タイムスタンプを update_parameter として使用する方法の例です。

```
INSERT INTO catalog.book_awards (year, award, rank, category, book_title, author,
 publisher)
   VALUES (2022, 'Wolf', 4, 'Non-Fiction', 'Science Update', 'Ana Carolina Silva',
 'SomePublisher')
   USING TIMESTAMP 1669069624;
```

CQL クエリでタイムスタンプを指定しない場合、Amazon Keyspaces はクライアントドライバーから渡されたタイムスタンプを使用します。クライアントドライバーからタイムスタンプが提供されない場合、Amazon Keyspaces は書き込み操作にサーバー側のタイムスタンプを割り当てます。

特定の列に保存されているタイムスタンプ値を確認するには、次の例のように WRITETIME 関数を SELECT ステートメントで使用できます。

```
SELECT year, award, rank, category, book_title, author, publisher, WRITETIME(year),
 WRITETIME(award), WRITETIME(rank),
  WRITETIME(category), WRITETIME(book_title), WRITETIME(author), WRITETIME(publisher)
 from catalog.book_awards;
```

Amazon Keyspaces のマルチリージョンレプリケーション (Apache Cassandra 用)

Amazon Keyspaces マルチリージョンレプリケーションを使用して、選択した 全体で自動、フルマネージド、アクティブ/アクティブレプリケーションを使用してデータをレプリケート AWS リージョン できます。アクティブ-アクティブレプリケーションでは、各リージョンが個別に読み取りと書き込みを行うことができます。リージョンレベルでパフォーマンスが低下した場合に可用性と回復力の両方を復旧でき、グローバルアプリケーションの場合はローカルでの低レイテンシーの読み取りと書き込みというメリットもあります。

マルチリージョンレプリケーションでは、Amazon Keyspaces はリージョン間でデータを非同期的にレプリケートし、通常、データは 1 秒以内にリージョン間で伝播されます。また、マルチリージョンレプリケーションでは、競合を解決し、データ分散の問題を修正するという難しい作業がなくなり、アプリケーションに集中できるようになります。

デフォルトでは、Amazon Keyspaces は耐久性と高可用性 AWS リージョン のために、同じ 内の 3 つのアベイラビリティーゾーンにデータをレプリケートします。マルチリージョンレプリケーションを使用すると、 AWS リージョン 選択したさまざまな地理的にテーブルをレプリケートするマルチリージョンキースペースを作成できます。

トピック

- マルチリージョンレプリケーションを使用する利点
- キャパシティモードと料金
- Amazon Keyspaces でのマルチリージョンレプリケーションの仕組み
- Amazon Keyspaces マルチリージョンレプリケーションの使用に関する注意事項
- Amazon Keyspaces のマルチリージョンレプリケーションを設定する (Apache Cassandra 用)

マルチリージョンレプリケーションを使用する利点

マルチリージョンレプリケーションには以下の利点があります。

- 1 桁ミリ秒のレイテンシーでグローバルな読み取りと書き込みを実行 – Amazon Keyspaces では、レプリケーションはアクティブ-アクティブで行われます。どのような規模でも、1 桁ミリ秒のレイテンシーで、顧客に最も近いリージョンからの読み取りと書き込みの両方をローカルで処理できます。世界中のどこでも、Amazon Keyspaces のマルチリージョンテーブルは、高速な応答時間を必要とするグローバルアプリケーションに使用できます。

- ビジネス継続性の向上と単一リージョンの劣化からの保護 – マルチリージョンレプリケーションでは、アプリケーションをマルチリージョンキースペース内の別のリージョンにリダイレクト AWS リージョン することで、単一リージョンの劣化から回復できます。Amazon Keyspaces はアクティブ-アクティブレプリケーションを提供するため、読み取りと書き込みに影響はありません。

  Amazon Keyspaces は、マルチリージョンキースペースに対して実行されても、すべてのレプリカリージョンにまだ反映されていない書き込みを追跡します。リージョンがオンラインに復帰した際に、欠落した変更があれば自動的に同期されるため、アプリケーションに影響を及ぼすことなく復旧できます。
- リージョン間の高速レプリケーション – マルチリージョンレプリケーションは、リージョン間のデータのストレージベースの高速物理レプリケーションを使用し、レプリケーションの遅延は通常 1 秒未満です。

  Amazon Keyspaces でのレプリケーションは、コンピューティングリソースをアプリケーションと共有しないため、データベースクエリにほとんどまたはまったく影響がありません。そのため、アプリケーションに影響を及ぼすことなく、書き込みスループットが高いユースケースや、スループットが急増する (バースト性が高い) ユースケースに対応できます。
- 整合性と競合の解決 – いずれかのリージョンのデータに加えられた変更は、マルチリージョンキースペース内の他のリージョンにレプリケートされます。アプリケーションが異なるリージョンにある同一データをほぼ同時に更新すると、競合が発生する可能性があります。

  最終的な一貫性を保つために、Amazon Keyspaces はセルレベルのタイムスタンプを使用し、同時更新間の調整は最後のライターが優先します。競合の解決は全面的に管理され、アプリケーションに影響を与えることなくバックグラウンドで処理されます。

サポート対象の設定の詳細については、「the section called “使用に関する注意事項”」を参照してください。

キャパシティモードと料金

マルチリージョンキースペースでは、オンデマンドキャパシティモードまたはプロビジョンドキャパシティモードを使用できます。詳細については、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。

オンデマンドモードでは、単一リージョンテーブルの場合と同様に、行ごとに最大 1 KB のデータを書き込むように 1 つの書き込みリクエストユニット (WRU) に課金されます。ただし、マルチリージョンキースペースの各リージョンでの書き込みに対して課金されます。たとえば、2 つのリー

ジョンを持つマルチリージョンキースペースに 3 KB のデータ行を書き込むには、3 * 2 = 6 WRUs 6 WRUsが必要です。さらに、静的データと非静的データの両方がある書き込みには、追加の書き込みオペレーションが必要です。

プロビジョニングモードでは、単一リージョンテーブルの場合と同様に、行ごとに最大 1 KB のデータを書き込むように 1 つの書き込みキャパシティーユニット (WCU) に課金されます。ただし、マルチリージョンキースペースの各リージョンでの書き込みに対して課金されます。たとえば、2 つのリージョンを持つマルチリージョンキースペースで 1 秒あたり 3 KB のデータ行を書き込むには、3 x 2 = 6 WCUs 6 WCUsが必要です。さらに、静的データと非静的データの両方がある書き込みには、追加の書き込みオペレーションが必要です。

料金の詳細については、「Amazon Keyspaces (for Apache Cassandra) pricing (Amazon Keyspaces (Apache Cassandra 向け) の料金)」を参照してください。

Amazon Keyspaces でのマルチリージョンレプリケーションの仕組み

このセクションでは、Amazon Keyspaces マルチリージョンレプリケーションの仕組みの概要を説明します。料金の詳細については、「Amazon Keyspaces (for Apache Cassandra) pricing (Amazon Keyspaces (Apache Cassandra 向け) の料金)」を参照してください。

トピック

- Amazon Keyspaces でのマルチリージョンレプリケーションの仕組み
- マルチリージョンレプリケーションの競合の解決
- マルチリージョンレプリケーションディザスタリカバリ
- のマルチリージョンレプリケーションはデフォルトで AWS リージョン 無効になっています
- マルチリージョンレプリケーションとpoint-in-timeリカバリ (PITR) との統合
- マルチリージョンレプリケーションと AWS サービスとの統合

Amazon Keyspaces でのマルチリージョンレプリケーションの仕組み

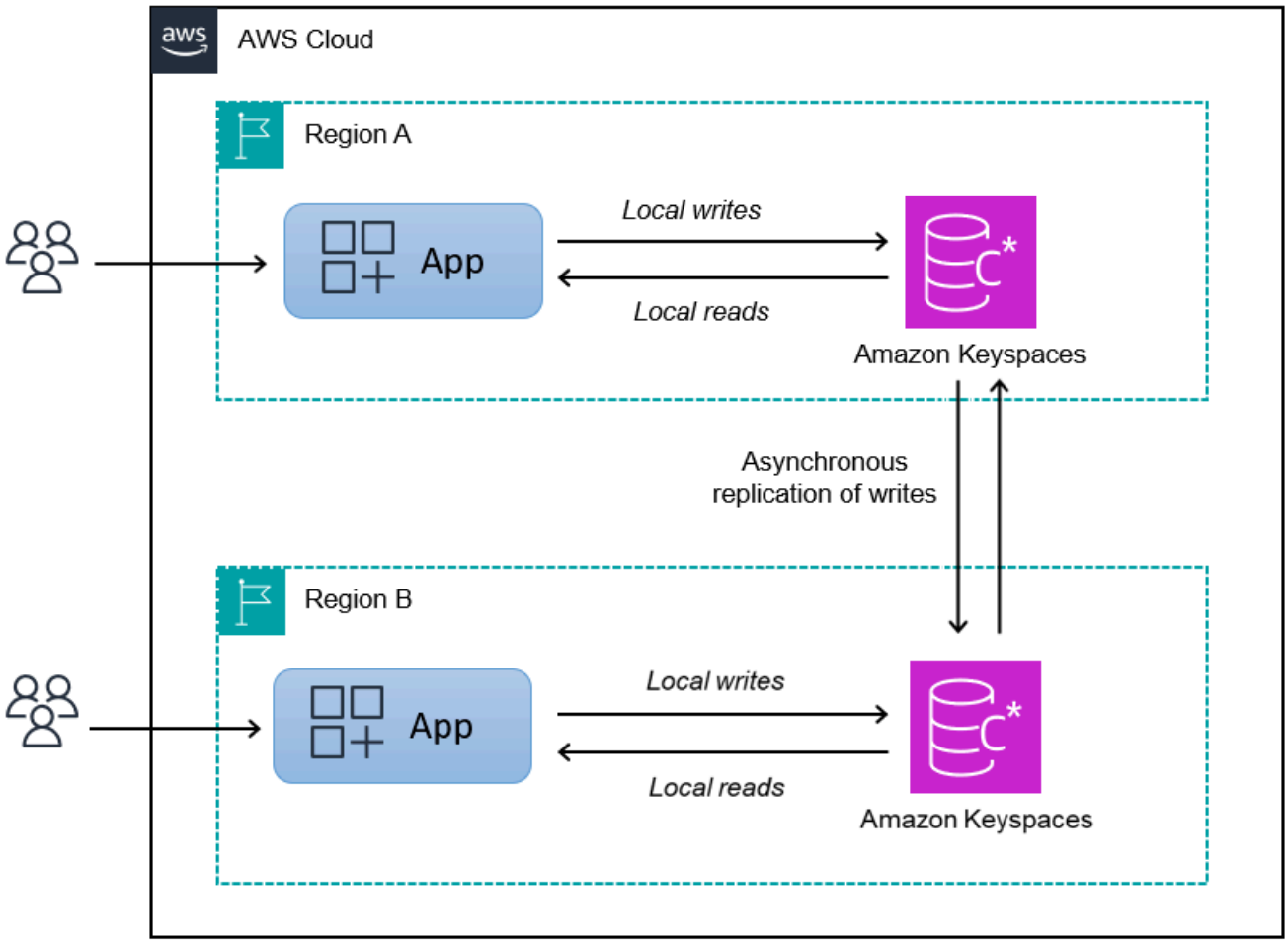
Amazon Keyspaces マルチリージョンレプリケーションは、独立した地理的分散型にデータを分散するデータ耐障害性アーキテクチャを実装します AWS リージョン。アクティブ-アクティブレプリケーションを使用しているため、ローカルで低レイテンシーを実現し、各リージョンが個別に読み取りと書き込みを実行できます。

Amazon Keyspaces マルチリージョンキースペースを作成するときに、データがレプリケートされる追加のリージョンを選択できます。マルチリージョンのキー空間に作成するテーブルは、Amazon

Keyspaces に 1 つの単位と認識される複数のレプリカテーブル (リージョンごとに 1 つ) で構成されます。

すべてのレプリカは、同じテーブル名と同じプライマリキーのスキーマを持っています。アプリケーションによって 1 つのリージョンのローカルテーブルにデータが書き込まれると、データはその LOCAL_QUORUM 整合性レベルを使用して永続的に書き込まれます。Amazon Keyspaces は、データを他のレプリケーションリージョンに自動的に非同期で複製します。リージョン間のレプリケーション遅延は通常 1 秒未満なので、アプリケーションのパフォーマンスやスループットには影響はありません。

データが書き込まれると、LOCAL_ONE/LOCAL_QUORUM 整合性レベルが設定された別のレプリケーションリージョンのマルチリージョンテーブルからデータを読み取ることができます。サポートされる設定の詳細については、「the section called “使用に関する注意事項”」をご参照ください。



マルチリージョンレプリケーションの競合の解決

Amazon Keyspaces マルチリージョンレプリケーションはフルマネージド型であるため、データ同期の問題をクリーンアップするために定期的に修復オペレーションを実行するなど、レプリケーションタスクを実行する必要はありません。Amazon Keyspaces は、競合を検出して修復 AWS リージョン することで、異なる のテーブル間のデータ整合性をモニタリングし、レプリカを自動的に同期します。

Amazon Keyspaces では、最終ライターが勝者となる方法を使用してデータ調整を行います。この競合解決メカニズムでは、マルチリージョンキー空間内のすべてのリージョンが最新の更新を受け入れ、すべてのリージョンで同一のデータが保存される状態に収束します。調整プロセスはアプリケーションのパフォーマンスには影響しません。競合解決をサポートするため、マルチリージョンテーブルではクライアント側のタイムスタンプが自動的にオンになります。オフにすることはできません。詳細については、「the section called “クライアントサイドのタイムスタンプ”」を参照してください。

マルチリージョンレプリケーションディザスタリカバリ

Amazon Keyspaces マルチリージョンレプリケーションでは、書き込みは各リージョン間で非同期的にレプリケートされます。まれに単一リージョンの劣化や障害が発生した場合、マルチリージョンレプリケーションを使用すると、アプリケーションにほとんどまたはまったく影響することなく、災害から回復できます。ディザスターからの回復は、通常、目標復旧時間 (RTO) と目標復旧時点 (RPO) の値を使用して測定されます。

目標復旧時間 - ディザスター後にシステムが稼働状態に戻るまでにかかる時間。RTO は、ワークロードが許容できるダウンタイムを時間単位で測定します。マルチリージョンレプリケーションを使用して影響を受けていないリージョンにフェイルオーバーするディザスタリカバリプランの場合、RTO はほぼゼロになる可能性があります。RTO は、アプリケーションが障害状態をどれだけ早く検出してトラフィックを別のリージョンにリダイレクトできるかによって制限されます。

目標復旧時点 - 損失するおそれのあるデータの量 (時間単位)。マルチリージョンレプリケーションを使用して影響を受けていないリージョンにフェイルオーバーするディザスタリカバリプランの場合、RPO は通常 1 桁の秒です。RPO は、フェールオーバーターゲットレプリカへのレプリケーション遅延によって制限されます。

Amazon Keyspaces のレプリケーションはアクティブ-アクティブなので、リージョンの障害または機能低下が発生しても、セカンダリリージョンの昇格や、データベースのフェイルオーバー手順の実行は必要ありません。代わりに、Amazon Route 53 で、最も近くにある正常なリージョンにアプリ

ケーションをルーティングできます。Route 53 の詳細については、「Amazon Route 53 とは?」を参照してください。 。

単一の AWS リージョン が分離または低下した場合、アプリケーションは Route 53 を使用してトラフィックを別のリージョンにリダイレクトし、別のレプリカテーブルに対して読み取りと書き込みを実行できます。また、カスタムビジネスロジックを適用すれば、リクエストを他のリージョンにリダイレクトするタイミングを決定できます。一例として、使用可能な複数のエンドポイントをアプリケーションに認識させるケースが考えられます。

リージョンがオンラインに戻ると、Amazon Keyspaces はそのリージョンから他のリージョンのレプリカテーブルへの保留中の書き込みの伝播を再開します。また、他のレプリカテーブルから現在オンラインに戻っているリージョンへの書き込みの伝播も再開します。

のマルチリージョンレプリケーションはデフォルトで AWS リージョン 無効になっています

Amazon Keyspaces マルチリージョンレプリケーションは、デフォルトで無効 AWS リージョン になっている以下の でサポートされています。

- アフリカ (ケープタウン) リージョン

Amazon Keyspaces マルチリージョンレプリケーションでデフォルトで無効になっているリージョンを使用するには、まずリージョンを有効にする必要があります。詳細については、「 AWS Organizations ユーザーガイド」の「アカウント AWS リージョン で を有効または無効にする」を参照してください。

リージョンを有効にしたら、そのリージョンに新しい Amazon Keyspaces リソースを作成し、そのリージョンをマルチリージョンキースペースに追加できます。

Amazon Keyspaces マルチリージョンレプリケーションで使用されるリージョンを無効にすると、Amazon Keyspaces は 24 時間の猶予期間を開始します。この時間枠では、次の動作が期待できます。

- Amazon Keyspaces は、有効なリージョンで引き続きデータ操作言語 (DML) オペレーションを実行します。
- Amazon Keyspaces は、有効なリージョンから無効なリージョンへのデータ更新のレプリケーションを一時停止します。
- Amazon Keyspaces は、無効なリージョン内のすべてのデータ定義言語 (DDL) リクエストをブロックします。

リージョンを誤って無効にした場合は、24 時間以内にリージョンを再度有効にできます。24 時間の猶予期間中にリージョンを再度有効にすると、Amazon Keyspaces は次のアクションを実行します。

- 再有効化されたリージョンへのすべてのレプリケーションを自動的に再開します。
- データの整合性を確保するために、リージョンが無効になっている間に有効なリージョンで発生したデータ更新をレプリケートします。
- 追加のすべてのマルチリージョンレプリケーションオペレーションを自動的に続行します。

24 時間のウィンドウが閉じた後もリージョンが無効のままの場合、Amazon Keyspaces は次のアクションを実行して、マルチリージョンレプリケーションからリージョンを完全に削除します。

- すべてのマルチリージョンレプリケーションキースペースから無効なリージョンを削除します。
- 無効なリージョンのマルチリージョンレプリケーションテーブルレプリカを単一リージョンのキースペースとテーブルに変換します。
- Amazon Keyspaces は、無効なリージョンからリソースを削除しません。

Amazon Keyspaces が無効化されたリージョンをマルチリージョンキースペースから完全に削除した後は、無効化されたリージョンを再度追加することはできません。

マルチリージョンレプリケーションとpoint-in-timeリカバリ (PITR) との統合

Point-in-timeリカバリは、マルチリージョンテーブルでサポートされています。PITR を使用してマルチリージョンテーブルを正常に復元するには、次の条件を満たす必要があります。

- ソーステーブルとターゲットテーブルは、マルチリージョンのテーブルとして設定します。
- ソーステーブルのキー空間とターゲットテーブルのキー空間のレプリケーションリージョンは同じリージョンであることとします。
- PITR は、ソーステーブルのすべてのレプリカで有効にする必要があります。

restore ステートメントは、ソーステーブルが使用可能であれば、どのリージョンからでも実行できます。Amazon Keyspaces は、各リージョンのターゲットテーブルを自動的に復元します。PITR の詳細については「the section called “仕組み”」を参照してください。

マルチリージョンテーブルを作成すると、作成プロセス中に定義した PITR 設定がすべてのリージョンのすべてのテーブルに自動的に適用されます。を使用して PITR 設定を変更するとALTER

TABLE、Amazon Keyspaces は更新をローカルテーブルにのみ適用し、他のリージョンのレプリカには適用されません。既存のマルチリージョンテーブルに対して PITR を有効にするには、すべてのレプリカに対して ALTER TABLEステートメントを繰り返す必要があります。

マルチリージョンレプリケーションと AWS サービスとの統合

Amazon CloudWatch メトリクスを使用して AWS リージョン 、異なる のテーブル間のレプリケーションパフォーマンスをモニタリングできます。次のメトリクスでは、マルチリージョンキー空間を継続的にモニタリングできます。

- ReplicationLatency — このメトリクスは、マルチリージョンキー空間内のあるレプリカテーブルから別のレプリカテーブルへ updates、inserts、または deletes を複製するときの所用時間を測定します。

CloudWatch のメトリクスのモニタリング方法については、「the section called “CloudWatch によるモニタリング”」を参照してください。

Amazon Keyspaces マルチリージョンレプリケーションの使用に関する注意事項

Amazon Keyspaces でマルチリージョンレプリケーションを使用する場合は、次の点を考慮してください。

- 使用可能な任意のパブリック AWS リージョンを選択できます。 AWS リージョン デフォルトでは無効になっている の詳細については、「」を参照してくださいthe section called “デフォルトでは無効になっているリージョン”。
- AWS GovCloud (US) Regions および中国リージョンはサポートされていません。
- 各機能が利用できるようになるまでは、次の対策を検討してください。

  マルチリージョンテーブルの作成時に Time to Live (TTL) を設定します。後から TTL の有効/無効を切り替えることや、TTL 値を調整することはできません。詳細については、「the section called “TTL によるデータの期限切れ”」を参照してください。

  - 保管時の暗号化には、 AWS 所有キーを使用します。カスタマーマネージドキーは、現時点では、マルチリージョンテーブルではサポートされていません。詳細については「」を参照してください。

    the section called “動作”.

- ALTER KEYSPACE を使用して、リージョンを単一リージョンまたはマルチリージョンのキースペースに追加できます。詳細については、「the section called “キースペースにリージョンを追加する”」を参照してください。
  - リージョンを単一リージョンのキースペースに追加する前に、キースペースの下にカスタマーマネージドキーでテーブルが設定されていないことを確認してください。
  - キースペースまたはテーブル用に設定された既存のタグは、新しいリージョンにレプリケートされません。
- プロビジョニングされるキャパシティを Amazon Keyspaces の自動スケーリングで管理する場合は、Amazon Keyspaces の API オペレーションを使用してマルチリージョンテーブルを作成および設定してください。Amazon Keyspaces で自動的に呼び出される基盤の Application Auto Scaling API オペレーションには、マルチリージョンの機能がありません。

  詳細については、「the section called “マルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定の更新”」を参照してください。プロビジョンドモードのマルチリージョンテーブルの書き込みキャパシティスループットを推定する方法については、「the section called “マルチリージョンテーブルのキャパシティの推定”」を参照してください。
- データはマルチリージョンテーブルの選択したリージョン間で自動的に複製されますが、クライアントが 1 つのリージョンのエンドポイントに接続して system.peers テーブルをクエリすると、クエリはローカル情報のみを返します。クエリ結果は、クライアントには単一のデータセンターのクラスターのように見えます。
- Amazon Keyspaces マルチリージョンレプリケーションは非同期であり、書き込みのLOCAL_QUORUM一貫性をサポートします。 のLOCAL_QUORUM一貫性を保つには、クライアントに成功を返す前に、行の更新をローカルリージョンの 2 つのレプリカに永続的に保持する必要があります。レプリケート先のリージョンへの書き込みの伝播は、その後非同期で実行されます。

  Amazon Keyspaces マルチリージョンレプリケーションは、同期レプリケーションまたはQUORUM整合性をサポートしていません。
- マルチリージョンのキースペースまたはテーブルを作成すると、作成プロセス中に定義したタグは、すべてのリージョンのすべてのキースペースとテーブルに自動的に適用されます。ALTER KEYSPACE または ALTER TABLE を使用して既存のタグを変更した場合、変更を行ったリージョン内の該当するキースペースまたはテーブルのみが更新されます。
- Amazon CloudWatch は、複製した各リージョンに ReplicationLatency メトリクスを提供します。このメトリクスは、到着する行を追跡し、到着時間と最初の書き込み時間を比較して、平均値を計算することで算出されます。タイミングはソースリージョンの CloudWatch 内に保存されます。詳細については、「the section called “CloudWatch によるモニタリング”」を参照してください。

タイミングの平均値と最大値を表示すると、レプリケーション遅延の平均値や最悪のケースを把握するのに役立ちます。このレイテンシーには SLA はありません。

- マルチリージョンテーブルをオンデマンドモードで使用する場合は、テーブルレプリカが新たなトラフィックピークを迎えると、書き込みの非同期レプリケーションのレイテンシーが増加する可能性があります。Amazon Keyspaces は受信したアプリケーショントラフィックに応じて単一リージョンのオンデマンドテーブルのキャパシティを自動調整しますが、それと同様に、マルチリージョンのオンデマンドテーブルレプリカのキャパシティも、受信したトラフィックに応じて自動的に調整します。Amazon Keyspaces はトラフィック量が増えれば、その分割り当てるキャパシティを自動的に増量させるため、レプリケーションレイテンシーの増加は一時的な現象です。すべてのレプリカがトラフィック量に合わせて調整されれば、レプリケーションレイテンシーは正常に戻ります。詳細については、「the section called “ピークトラフィックとスケーリングプロパティ”」を参照してください。
- マルチリージョンテーブルをプロビジョンドモードで使用する場合は、アプリケーションの消費量がプロビジョニング済みのスループットキャパシティを上回ると、キャパシティ不足のエラーが発生し、レプリケーションレイテンシーが増加する可能性があります。すべての AWS リージョンマルチリージョンテーブルのすべてのテーブルレプリカに十分な読み取りおよび書き込み容量を常に確保するには、Amazon Keyspaces 自動スケーリングを設定することをお勧めします。Amazon Keyspaces の自動スケーリングは、実際のアプリケーショントラフィックに応じてスループットキャパシティを自動的に調整するので、変動的なワークロードに対してスループットキャパシティを効率的にプロビジョニングできます。詳細については、「the section called “マルチリージョンテーブルに対する自動スケーリングの仕組み”」を参照してください。

Amazon Keyspaces のマルチリージョンレプリケーションを設定する (Apache Cassandra 用)

コンソール、Cassandra クエリ言語 (CQL)、または を使用して AWS Command Line Interface 、Amazon Keyspaces でマルチリージョンのキースペースとテーブルを作成および管理できます。

このセクションでは、マルチリージョンのキースペースとテーブルを作成し、管理する方法の例を紹介します。マルチリージョンキースペース内に作成したテーブルはすべて、そのキースペースのマルチリージョン設定を自動的に継承します。

サポートされる設定の詳細については、「the section called “使用に関する注意事項”」をご参照ください。

トピック

- マルチリージョンのキースペースとテーブルの作成に必要な IAM アクセス許可を設定する
- キースペースに を追加するために必要な AWS リージョン IAM アクセス許可を設定する
- Amazon Keyspaces でマルチリージョンキースペースを作成する
- Amazon Keyspaces のキースペース AWS リージョン に を追加する
- キースペースに新しいリージョンを追加するときにレプリケーションの進行状況を確認する
- Amazon Keyspaces でデフォルト設定のマルチリージョンテーブルを作成する
- Amazon Keyspaces で自動スケーリングが有効なプロビジョンドモードのマルチリージョンテーブルを作成する
- Amazon Keyspaces でマルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定を更新する
- Amazon Keyspaces でマルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定を表示する
- Amazon Keyspaces でテーブルの自動スケーリングを無効にする
- Amazon Keyspaces でマルチリージョンテーブルにプロビジョニングされるキャパシティを手動で設定する

マルチリージョンのキースペースとテーブルの作成に必要な IAM アクセス許可を設定する

マルチリージョンのキースペースとテーブルを正常に作成するには、IAM プリンシパルがサービスにリンクされたロールを作成できることが条件になります。このサービスにリンクされたロールは、Amazon Keyspaces によって事前に定義された特殊なタイプの IAM ロールです。このロールには、Amazon Keyspaces がお客様に代わってアクションを実行するために必要なすべての権限が含まれます。サービスにリンクされたロールの詳細については、「the section called “マルチリージョンレプリケーション”」を参照してください。

マルチリージョンレプリケーション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 プリンシパルのポリシーに次の要素が必要です。

- `iam:CreateServiceLinkedRole` – プリンシパルが実行できるアクション。
- `arn:aws:iam::*:role/aws-service-role/replication.cassandra.amazonaws.com/AWSServiceRoleForKeyspacesReplication` – このアクションを実行できるリソース。

- iam:AWSServiceName": "replication.cassandra.amazonaws.com – このロールをアタッチできる唯一の AWS サービスは Amazon Keyspaces です。

マルチリージョンキースペースとテーブルを作成するために必要な最小限の権限をユーザーに与えるポリシーの例を以下に示します。

```
{
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
replication.cassandra.amazonaws.com/AWSServiceRoleForKeyspacesReplication",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "replication.cassandra.amazonaws.com"}}
}
```

マルチリージョンのキースペースとテーブル用のその他の IAM アクセス許可については、「サービス認可リファレンス」の「Actions, resources, and condition keys for Amazon Keyspaces (for Apache Cassandra)」を参照してください。

キースペースに を追加するために必要な AWS リージョン IAM アクセス許可を設定する

リージョンをキースペースに追加するには、IAM プリンシパルに次のアクセス許可が必要です。

- cassandra:Alter
- cassandra:AlterMultiRegionResource
- cassandra:Create
- cassandra:CreateMultiRegionResource
- cassandra:Select
- cassandra:SelectMultiRegionResource
- cassandra:Modify
- cassandra:ModifyMultiRegionResource

テーブルがプロビジョニングモードで自動スケーリングを有効にして設定されている場合は、次の追加のアクセス許可が必要です。

- application-autoscaling:RegisterScalableTarget

- application-autoscaling:DeregisterScalableTarget
- application-autoscaling:DescribeScalableTargets
- application-autoscaling:PutScalingPolicy
- application-autoscaling:DescribeScalingPolicies

リージョンを単一リージョンのキースペースに正常に追加するには、IAM プリンシパルもサービスにリンクされたロールを作成できる必要があります。このサービスにリンクされたロールは、Amazon Keyspaces によって事前に定義された特殊なタイプの IAM ロールです。このロールには、Amazon Keyspaces がお客様に代わってアクションを実行するために必要なすべての権限が含まれます。サービスにリンクされたロールの詳細については、「the section called “マルチリージョンレプリケーション”」を参照してください。

マルチリージョンレプリケーション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 プリンシパルのポリシーに次の要素が必要です。

- iam:CreateServiceLinkedRole – プリンシパルが実行できるアクション。
- arn:aws:iam::*:role/aws-service-role/replication.cassandra.amazonaws.com/AWSServiceRoleForKeyspacesReplication – このアクションを実行できるリソース。
- iam:AWSServiceName": "replication.cassandra.amazonaws.com – このロールをアタッチできる唯一の AWS サービスは Amazon Keyspaces です。

以下は、キースペースにリージョンを追加するために必要な最小限のアクセス許可をプリンシパルに付与するポリシーの例です。

```
{
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
replication.cassandra.amazonaws.com/AWSServiceRoleForKeyspacesReplication",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "replication.cassandra.amazonaws.com"}}
}
```

Amazon Keyspaces でマルチリージョンキースペースを作成する

このセクションでは、マルチリージョンキースペースの作成方法の例を紹介します。この作業は、Amazon Keyspaces コンソール、CQL、または AWS CLIを使用して行うことができます。マル

チリージョンキースペース内に作成したテーブルはすべて、そのキースペースのマルチリージョン設定を自動的に継承します。

> **Note**
>
> マルチリージョンキースペースを作成すると、Amazon Keyspaces はアカウント内の名前 `AWSServiceRoleForAmazonKeyspacesReplication` でサービスにリンクされたロールを作成します。このロールにより、Amazon Keyspaces はユーザーに代わってマルチリージョンテーブルのすべてのレプリカへの書き込みを複製できます。詳細についてはthe section called “マルチリージョンレプリケーション”を参照してください。

Console

マルチリージョンキースペースを作成する (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで [Keyspaces (キースペース)] を選択し、次に [Create keyspace (キースペースの作成)] を選択します。
3. [Keyspace name (キースペース名)] でキースペースの名前を入力します。
4. マルチリージョンレプリケーションセクションでは、リストで使用できるリージョンを追加できます。
5. 終了するには、[ロールを作成] を選択します。

Cassandra Query Language (CQL)

CQL を使用してマルチリージョンキースペースを作成する

1. マルチリージョンキースペースを作成するには、 `NetworkTopologyStrategy`を使用して、キースペースをレプリケート AWS リージョン する を指定します。現在のリージョン以外に、追加リージョンを 1 つ以上、含める必要があります。

   キースペース内のすべてのテーブルは、そのキースペースのレプリケーション戦略を継承します。テーブルレベルでレプリケーション戦略を変更することはできません。

NetworkTopologyStrategy – Amazon Keyspaces はデフォルトで同じ 内の 3 つのアベイラビリティーゾーンにデータをレプリケートするため AWS リージョン、各リージョンのレプリケーション係数は 3 です。

例を、次の CQL ステートメントで示します。

```
CREATE KEYSPACE mykeyspace
WITH REPLICATION = {'class':'NetworkTopologyStrategy', 'us-east-1':'3', 'ap-
southeast-1':'3','eu-west-1':'3' };
```

2. CQL ステートメントで system_multiregion_info キースペース内の tables テーブルをクエリすると、指定したマルチリージョンテーブルのリージョンとステータスをプログラムによって一覧表示できます。次のコードは、この例です。

```
SELECT * from system_multiregion_info.tables WHERE keyspace_name = 'mykeyspace'
 AND table_name = 'mytable';
```

このステートメントの出力は、次のようになります。

```
 keyspace_name  | table_name     | region         | status
----------------+----------------+----------------+--------
 mykeyspace     | mytable        | us-east-1      | ACTIVE
 mykeyspace     | mytable        | ap-southeast-1 | ACTIVE
 mykeyspace     | mytable        | eu-west-1      | ACTIVE
```

CLI

を使用して新しいマルチリージョンキースペースを作成する AWS CLI

- マルチリージョンキースペースは、次の CLI ステートメントで作成できます。現在のリージョンと regionList に 1 つ以上の追加リージョンを指定します。

```
aws keyspaces create-keyspace --keyspace-name mykeyspace \
--replication-specification replicationStrategy=MULTI_REGION,regionList=us-
east-1,eu-west-1
```

マルチリージョンテーブルを作成するには、「the section called “デフォルト設定のマルチリージョンテーブルの作成”」および「the section called “プロビジョンドモードのマルチリージョンテーブルの作成”」を参照してください。

Amazon Keyspaces のキースペース AWS リージョン に を追加する

単一のキースペースまたはマルチリージョンのキースペースのいずれかであるキースペース AWS リージョン に新しい を追加できます。新しいレプリカリージョンは、キースペース内のすべてのテーブルに適用されます。

シングルリージョンをマルチリージョンキースペースに変更するには、キースペース内のすべてのテーブルに対してクライアント側のタイムスタンプを有効にする必要があります。詳細については、「the section called “クライアントサイドのタイムスタンプ”」を参照してください。

マルチリージョンキースペースにリージョンを追加する場合、Amazon Keyspaces は、既存のテーブルごとに 1 回限りのクロスリージョン復元を使用して、既存のテーブル (複数可) を新しいリージョンにレプリケートする必要があります。各テーブルの復元料金は GB ごとに請求されます。詳細については、Amazon Keyspaces (Apache Cassandra の場合) の料金ページの「バックアップと復元」を参照してください。この復元オペレーションでは、リージョン間のデータ転送に料金はかかりません。データに加えて、タグを除くすべてのテーブルプロパティが新しいリージョンにレプリケートされます。

CQL の `ALTER KEYSPACE`ステートメント、 の `update-keyspace` コマンド AWS CLI、またはコンソールを使用して、Amazon Keyspaces の単一のキースペースまたはマルチリージョンキースペースに新しいリージョンを追加できます。ステートメントを正常に実行するには、使用しているアカウントが、キースペースが既に利用可能なリージョンのいずれかにある必要があります。レプリカの追加中は、更新およびレプリケートされるリソースに対して他のデータ定義言語 (DDL) オペレーションを実行することはできません。

リージョンの追加に必要なアクセス許可の詳細については、「」を参照してくださいthe section called “リージョンを追加するための IAM アクセス許可を設定する”。

> **Note**
>
> 単一リージョンのキースペースにリージョンを追加すると、Amazon Keyspaces は`AWSServiceRoleForAmazonKeyspacesReplication`アカウントに という名前のサービスにリンクされたロールを作成します。このロールにより、Amazon Keyspaces はテーブルを新しいリージョンにレプリケートし、ユーザーに代わって 1 つのテーブルからマルチ

リージョンテーブルのすべてのレプリカに書き込みをレプリケートできます。詳細についてはthe section called “マルチリージョンレプリケーション”を参照してください。

Console

Amazon Keyspaces コンソールを使用してキースペースにリージョンを追加するには、次の手順に従います。

キースペースにリージョンを追加する (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、キースペースを選択し、リストからキースペースを選択します。
3. [AWS リージョン] タブを選択します。
4. AWS リージョン タブで、リージョンの追加を選択します。
5. リージョンの追加ダイアログで、キースペースに追加する追加のリージョンを選択します。
6. 終了するには、追加 を選択します。

Cassandra Query Language (CQL)

CQL を使用してキースペースにリージョンを追加する

- キースペースに新しいリージョンを追加するには、次のステートメントを使用できます。この例では、キースペースは米国東部 (バージニア北部) リージョンと米国西部 (オレゴン) リージョンで既に利用可能であり、CQL ステートメントは米国西部 (北カリフォルニア) リージョンを追加しています。

```
ALTER KEYSPACE my_keyspace
WITH REPLICATION = {
    'class': 'NetworkTopologyStrategy',
    'us-east-1': '3',
    'us-west-2': '3',
    'us-west-1': '3'
} AND CLIENT_SIDE_TIMESTAMPS = {'status': 'ENABLED'};
```

CLI

を使用してキースペースにリージョンを追加する AWS CLI

- CLI を使用してキースペースに新しいリージョンを追加するには、次の例を使用できます。client-side-timestamps のデフォルト値は DISABLED です。update-keyspace コマンドでは、値を に変更する必要がありますENABLED。

```
aws keyspaces update-keyspace \
--keyspace-name my_keyspace \
--replication-specification '{"replicationStrategy": "MULTI_REGION",
 "regionList": ["us-east-1", "eu-west-1", "eu-west-3"] }' \
--client-side-timestamps '{"status": "ENABLED"}'
```

キースペースに新しいリージョンを追加するときにレプリケーションの進行状況を確認する

Amazon Keyspaces キースペースへの新しいリージョンの追加は、長時間実行されるオペレーションです。進行状況を追跡するには、このセクションに示すクエリを使用できます。

Cassandra Query Language (CQL)

CQL を使用してリージョンの追加の進行状況を確認する

- 特定のキースペースで新しいテーブルレプリカの作成の進行状況を確認するには、system_multiregion_info.keyspacesテーブルをクエリできます。例を、次の CQL ステートメントで示します。

```
SELECT keyspace_name, region, status, tables_replication_progress
FROM system_multiregion_info.keyspaces
WHERE keyspace_name = 'my_keyspace';
```

レプリケーションオペレーションの進行中、ステータスは新しいリージョンでのテーブル作成の進行状況を示します。これは、10 個のテーブルのうち 5 個が新しいリージョンにレプリケートされた例です。

```
 keyspace_name | region    | status    | tables_replication_progress
---------------+-----------+-----------+-------------------------
   my_keyspace | us-east-1 | Updating  |
```

```
   my_keyspace | us-west-2 | Updating  |
   my_keyspace | eu-west-1 | Creating  | 50%
```

レプリケーションプロセスが正常に完了すると、出力は次の例のようになります。

```
 keyspace_name | region    | status
---------------+-----------+-----------
   my_keyspace | us-east-1 | Active
   my_keyspace | us-west-2 | Active
   my_keyspace | eu-west-1 | Active
```

CLI

を使用してリージョンの追加の進行状況 AWS CLI を確認する

- 特定のキースペースのテーブルレプリカ作成のステータスを確認するには、次の例を使用します。

```
aws keyspaces get-keyspace \
--keyspace-name my_keyspace
```

この例のような出力が得られます。

```
{
    "keyspaceName": "my_keyspace",
    "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
my_keyspace/",
    "replicationStrategy": "MULTI_REGION",
    "replicationRegions": [
        "us-east-1",
        "eu-west-1"
    ]
    "replicationGroupStatus": [
        {
            "RegionName": "us-east-1",
            "KeyspaceStatus": "Active"
        },
        {
            "RegionName": "eu-west-1",
            "KeyspaceStatus": "Creating",
            "TablesReplicationProgress": "50.0%"
```

```
        }
    ]
}
```

Amazon Keyspaces でデフォルト設定のマルチリージョンテーブルを作成する

このセクションでは、オンデマンドモードのマルチリージョンテーブルをすべてデフォルトの設定で作成する方法の例を紹介します。この作業は、Amazon Keyspaces コンソール、CQL、または AWS CLIを使用して行うことができます。マルチリージョンキースペース内に作成したテーブルはすべて、そのキースペースのマルチリージョン設定を自動的に継承します。

マルチリージョンキースペースを作成するには、「the section called “マルチリージョンキースペースの作成”」を参照してください。

Console

マルチリージョンテーブルをデフォルト設定で作成する (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. マルチリージョンキー空間を選択します。
3. [テーブル] タブで [テーブルを作成] を選択します。
4. [テーブル名] にテーブルの名前を入力します。このテーブルが複製される AWS リージョンが情報ボックスに表示されます。
5. テーブルスキーマを続行します。
6. [テーブル設定] で、[デフォルト設定] オプションに進みます。マルチリージョンテーブルの次のデフォルト設定に注意してください。

   - キャパシティモード – デフォルトのキャパシティモードは [オンデマンド] です。プロビジョンドモードの設定の詳細については、「the section called “プロビジョンドモードのマルチリージョンテーブルの作成”」を参照してください。
   - 暗号化キー管理 — AWS 所有のキー オプションのみがサポートされます。
   - クライアント側のタイムスタンプ — この機能はマルチリージョンテーブルに必要です。
   - テーブルとそのすべてのレプリカの Time to Live (TTL) を有効にする必要がある場合は、[設定をカスタマイズ] を選択します。

> **Note**
>
> 既存のマルチリージョンテーブルの TTL 設定は変更できません。

7. 終了するには、[テーブルを作成] を選択します。

Cassandra Query Language (CQL)

オンデマンドモードのマルチリージョンテーブルをデフォルト設定で作成する

- マルチリージョンテーブルをデフォルト設定で作成するには、次の CQL ステートメントを使用します。

```
CREATE TABLE mykeyspace.mytable(pk int, ck int, PRIMARY KEY (pk, ck))
    WITH CUSTOM_PROPERTIES = {
 'capacity_mode':{
  'throughput_mode':'PAY_PER_REQUEST'
 },
 'point_in_time_recovery':{
  'status':'enabled'
 },
 'encryption_specification':{
  'encryption_type':'AWS_OWNED_KMS_KEY'
 },
 'client_side_timestamps':{
  'status':'enabled'
 }
};
```

CLI

の使用 AWS CLI

1. マルチリージョンテーブルをデフォルト設定で作成するには、スキーマのみを指定する必要があります。次の例を使用できます。

```
aws keyspaces create-table --keyspace-name mykeyspace --table-name mytable \
--schema-definition 'allColumns=[{name=pk,type=int}],partitionKeys={name= pk}'
```

このコマンドの出力は、次のようになります。

```
{
    "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
mykeyspace/table/mytable"
}
```

2. テーブルの設定を確認するには、次のステートメントを使用します。

```
aws keyspaces get-table --keyspace-name mykeyspace --table-name mytable
```

出力には、マルチリージョンテーブルのデフォルト設定がすべて表示されます。

```
{
    "keyspaceName": "mykeyspace",
    "tableName": "mytable",
    "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
mykeyspace/table/mytable",
    "creationTimestamp": "2023-12-19T16:50:37.639000+00:00",
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "pk",
                "type": "int"
            }
        ],
        "partitionKeys": [
            {
                "name": "pk"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2023-12-19T16:50:37.639000+00:00"
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
```

```
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    },
    "clientSideTimestamps": {
        "status": "ENABLED"
    },
    "replicaSpecifications": [
        {
            "region": "us-east-1",
            "status": "ACTIVE",
            "capacitySpecification": {
                "throughputMode": "PAY_PER_REQUEST",
                "lastUpdateToPayPerRequestTimestamp": 1702895811.469
            }
        },
        {
            "region": "eu-north-1",
            "status": "ACTIVE",
            "capacitySpecification": {
                "throughputMode": "PAY_PER_REQUEST",
                "lastUpdateToPayPerRequestTimestamp": 1702895811.121
            }
        }
    ]
}
```

Amazon Keyspaces で自動スケーリングが有効なプロビジョンドモードのマルチリージョンテーブルを作成する

このセクションでは、自動スケーリングが有効なプロビジョンドモードのマルチリージョンテーブルを作成する方法の例を紹介します。この作業は、Amazon Keyspaces コンソール、CQL、または AWS CLIを使用して行うことができます。

サポートされている設定とマルチリージョンレプリケーション機能の詳細については、「」を参照してくださいthe section called “使用に関する注意事項”。

マルチリージョンキースペースを作成するには、「the section called “マルチリージョンキースペースの作成”」を参照してください。

自動スケーリング設定を使用してプロビジョンドモードの新しいマルチリージョンテーブルを作成する場合、そのテーブルのレプリケート先のすべての AWS リージョン に適用される汎用設定を指定できます。その上で、読み取りキャパシティ設定と読み取りの自動スケーリング設定をレプリカごとに上書きできます。一方、書き込みキャパシティについては、すべてのリージョン間で書き込みをレプリケートするためのキャパシティを十分に確保するため、すべてのレプリカ間で同期されます。

> **Note**
>
> Amazon Keyspaces の自動スケーリングでは、ユーザーに代わって自動スケーリングアクションを実行するサービスリンクロール (`AWSServiceRoleForApplicationAutoScaling_CassandraTable`) の存在が必要になります。このロールは自動的に作成されます。詳細については、「the section called “サービスにリンクされたロールの使用”」を参照してください。

Console

自動スケーリングが有効な新しいマルチリージョンテーブルを作成する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. マルチリージョンキー空間を選択します。
3. [テーブル] タブで [テーブルを作成] を選択します。
4. [Table details (テーブルの詳細)] セクションの [Create table (テーブルの作成)] ページで、キースペースを選択し、新しいテーブルに名前を付けます。
5. [列] セクションで、テーブルのスキーマを作成します。
6. [プライマリキー] セクションで、テーブルのプライマリキーを定義し、オプションでクラスタリング列を選択します。
7. [Table settings (テーブルの設定)] セクションで、[Customize settings (設定のカスタマイズ)] を選択します。
8. [Read/write capacity settings (読み取り/書き込みキャパシティの設定)] に進みます。
9. [Capacity mode (キャパシティモード)] で、[Provisioned (プロビジョン)] を選択します。
10. [Read capacity] (読み取りキャパシティ) セクションで、[Scale automatically (自動スケーリング)] が選択されているか確認します。

テーブルのレプリケート先のすべての AWS リージョン で同じ読み取りキャパシティユニットを設定するチェックボックスを選択できます。または、そのチェックボックスの選択を解除して、リージョンごとに別々の読み取りキャパシティを設定できます。

リージョンごとに別々に設定する選択肢を選んだ場合は、各テーブルレプリカの最小と最大の読み取りキャパシティユニットとターゲット使用率を併せて選択します。

- 最小キャパシティユニット — テーブルをいつでもすぐにサポートできる状態にするために、最小レベルのスループット値を入力します。この値は、1 から、アカウントの秒単位の最大スループットクォータ (デフォルトは 40,000) までの範囲でなければなりません。
- 最大キャパシティユニット — テーブルに対してプロビジョニングするスループットの最大量を入力します。この値は、1 から、アカウントの秒単位の最大スループットクォータ (デフォルトは 40,000) までの範囲でなければなりません。
- ターゲット使用率— ターゲット使用率を 20% ~ 90% の範囲で入力します。定義したターゲット使用率をトラフィックが上回ると、キャパシティが自動的に増加されます。定義したターゲットをトラフィックが下回ると、再び容量が自動的に減少されます。

- テーブルの読み取りキャパシティを手動でプロビジョニングする場合は、[自動スケーリング] チェックボックスの選択を解除します。この設定は、テーブルのすべてのレプリカに適用されます。

> **Note**
>
> すべてのレプリカに十分な読み取りキャパシティを確保するために、プロビジョンドモードのマルチリージョンテーブルでは Amazon Keyspaces の自動スケーリングを使用することをお勧めします。

> **Note**
>
> アカウントのデフォルトクォータの詳細およびクォータを引き上げる方法については、「クォータ」を参照してください。

11. [読み込みキャパシティ] セクションで、[自動スケーリング] が選択されていることを確認します。その上で、テーブルのキャパシティユニットを設定します。書き込みキャパシティユニットは、すべてのリージョン間で書き込みイベントをレプリケートするためのキャパシティを十分に確保するため、すべての AWS リージョン 間で同期されます。

- テーブルの書き込みキャパシティを手動でプロビジョニングする場合は、[自動スケーリング] の選択を解除します。この設定は、テーブルのすべてのレプリカに適用されます。

> **Note**
>
> すべてのレプリカに十分な書き込みキャパシティを確保するために、プロビジョンドモードのマルチリージョンテーブルでは Amazon Keyspaces の自動スケーリングを使用することをお勧めします。

12. [Create table (テーブルの作成)] を選択します。指定した自動スケーリングパラメータを使用してテーブルが作成されます。

Cassandra Query Language (CQL)

自動スケーリングが有効なプロビジョンドキャパシティモードのマルチリージョンテーブルを CQL を使用して作成する

- 自動スケーリングが有効なプロビジョンドモードのマルチリージョンテーブルを作成するには、まず、テーブルの CUSTOM_PROPERTIES を定義し、キャパシティモードを指定する必要があります。プロビジョンド (provisioned) キャパシティモードを指定した後で、AUTOSCALING_SETTINGS を使用して、テーブルの自動スケーリングを設定できます。

  自動スケーリング設定、ターゲット追跡ポリシー、ターゲット値、オプション設定の詳細については、「the section called “自動スケーリングを有効にして新規テーブルを作成する”」を参照してください。

  特定のリージョン内のテーブルレプリカの読み取りキャパシティを定義するには、テーブルの replica_updates の一部として次のパラメータを設定できます。

  - リージョン
  - プロビジョニングされた読み取りキャパシティユニット (オプション)
  - 読み取りキャパシティの自動スケーリング設定 (オプション)

  次の例は、プロビジョンドモードのマルチリージョンテーブルの CREATE TABLE ステートメントです。書き込みキャパシティと読み取りキャパシティの自動スケーリングの汎用設定は同じです。ただし、読み取りの自動スケーリング設定では、テーブルの読み取りキャパシ

ティをスケールアップまたはスケールダウンする前のクールダウン期間として 60 秒が追加で指定されています。さらに、米国東部 (バージニア北部) リージョンの読み取りキャパシティの自動スケーリング設定は、他のレプリカと比べて高めの設定になっています。また、ターゲット値の設定も 50% ではなく 70% です。

```
CREATE TABLE mykeyspace.mytable(pk int, ck int, PRIMARY KEY (pk, ck))
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 5,
        'write_capacity_units': 5
    }
} AND AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50
            }
        }
    },
    'provisioned_read_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50,
                'scale_in_cooldown': 60,
                'scale_out_cooldown': 60
            }
        }
    },
    'replica_updates': {
        'us-east-1': {
            'provisioned_read_capacity_autoscaling_update': {
                'maximum_units': 20,
                'minimum_units': 5,
                'scaling_policy': {
                    'target_tracking_scaling_policy_configuration': {
                        'target_value': 70
                    }
                }
```

```
            }
        }
    }
};
```

CLI

を使用して、自動スケーリングを使用してプロビジョンドモードで新しいマルチリージョンテーブルを作成する AWS CLI

- 自動スケーリング設定を使用してプロビジョンドモードのマルチリージョンテーブルを作成するには、 AWS CLIを使用します。マルチリージョンの自動スケーリングの設定には、Amazon Keyspaces CLI の `create-table` コマンドを使用する必要があります。その理由は、Application Auto Scaling (Amazon Keyspaces が自動スケーリングを実行するために使用するサービス) がマルチリージョンに対応していないからです。

  自動スケーリング設定、ターゲット追跡ポリシー、ターゲット値、オプション設定の詳細については、「the section called “自動スケーリングを有効にして新規テーブルを作成する”」を参照してください。

  特定のリージョン内のテーブルレプリカの読み取りキャパシティを定義するには、テーブルの `replicaSpecifications` の一部として次のパラメータを設定できます。

  - リージョン
  - プロビジョニングされた読み取りキャパシティユニット (オプション)
  - 読み取りキャパシティの自動スケーリング設定 (オプション)

  自動スケーリング設定が複雑で、テーブルレプリカごとに設定が異なるマルチリージョンテーブルを作成する場合は、テーブルの自動スケーリング設定とレプリカ設定を JSON ファイルからロードすると便利です。

  次のコード例を使用するには、サンプル JSON ファイルを収録した auto-scaling.zip をダウンロードし、`auto-scaling.json` と `replication.json` を抽出します。ファイルのパスを書き留めておいてください。

  この例では、JSON ファイルは現在のディレクトリにあります。さまざまなファイルパスオプションについては、「ファイルからパラメーターを読み込む方法」を参照してください。

```
aws keyspaces create-table --keyspace-name mykeyspace --table-name mytable \
--schema-definition 'allColumns=[{name=pk,type=int},
{name=ck,type=int}],partitionKeys=[{name=pk},{name=ck}]' \
--capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1 \
--auto-scaling-specification file://auto-scaling.json \
--replica-specifications file://replication.json
```

Amazon Keyspaces でマルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定を更新する

このセクションでは、 コンソール、CQL、および を使用して、プロビジョニングされたマルチリージョンテーブルの Amazon Keyspaces 自動スケーリング設定 AWS CLI を管理する方法の例を示します。一般的な自動スケーリング設定のオプションの詳細と仕組みについては、「the section called “自動スケーリングによるスループットキャパシティの管理”」を参照してください。

マルチリージョンテーブルでプロビジョンドキャパシティモードを使用している場合は、常に Amazon Keyspaces API コールを使用して自動スケーリングを設定する必要があります。その理由は、基盤となる Application Auto Scaling API オペレーションがリージョンに対応していないからです。

プロビジョンドモードのマルチリージョンテーブルの書き込みキャパシティスループットを推定する方法については、「the section called “マルチリージョンテーブルのキャパシティの推定”」を参照してください。

Amazon Keyspaces API の詳細については、「Amazon Keyspaces API リファレンス」を参照してください。

マルチリージョンテーブルのププロビジョンドモードまたは自動スケーリングの設定を更新する場合、テーブルの各レプリカの読み取りキャパシティ設定と読み取りの自動スケーリング設定を更新できます。

一方、書き込みキャパシティについては、すべてのリージョン間で書き込みをレプリケートするためのキャパシティを十分に確保するため、すべてのレプリカ間で同期されます。

Cassandra Query Language (CQL)

CQL を使用してマルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定を更新する

- ALTER TABLE を使用して、既存のテーブルのキャパシティモードと自動スケーリングの設定を更新できます。現時点でオンデマンドキャパシティモードのテーブルを更新する場合は、capacity_mode が必須です。テーブルが既にプロビジョンドキャパシティモードになっている場合、このフィールドは省略できます。

  自動スケーリング設定、ターゲット追跡ポリシー、ターゲット値、オプション設定の詳細については、「the section called “自動スケーリングを有効にして新規テーブルを作成する”」を参照してください。

  同じステートメントでテーブルの replica_updates プロパティを更新して、特定のリージョンのテーブルレプリカの読み取りキャパシティと自動スケーリングの設定を更新することもできます。以下のステートメントは、この例です。

```
ALTER TABLE mykeyspace.mytable
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 1,
        'write_capacity_units': 1
    }
} AND AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50
            }
        }
    },
    'provisioned_read_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50,
                'scale_in_cooldown': 60,
```

```
                'scale_out_cooldown': 60
            }
        }
    },
    'replica_updates': {
        'us-east-1': {
            'provisioned_read_capacity_autoscaling_update': {
                'maximum_units': 20,
                'minimum_units': 5,
                'scaling_policy': {
                    'target_tracking_scaling_policy_configuration': {
                        'target_value': 70
                    }
                }
            }
        }
    }
};
```

CLI

を使用してマルチリージョンテーブルのプロビジョニングされた容量と自動スケーリング設定を更新する AWS CLI

- 既存のテーブルのプロビジョニングモードと自動スケーリング設定を更新するには、 コマンドを使用できます AWS CLI `update-table`。

  マルチリージョンの自動スケーリング設定を作成または変更するには、Amazon Keyspaces の CLI コマンドを使用する必要があります。その理由は、Application Auto Scaling (Amazon Keyspaces がテーブルの自動スケーリングを実行するために使用するサービス) が複数の AWS リージョンに対応していないからです。

  特定のリージョンのテーブルレプリカの読み取りキャパシティを更新するには、テーブルの `replicaSpecifications` の次のオプションパラメータのいずれかを変更します。

  - プロビジョニングされた読み取りキャパシティユニット (オプション)
  - 読み取りキャパシティの自動スケーリング設定 (オプション)

自動スケーリング設定が複雑で、テーブルレプリカごとに設定が異なるマルチリージョンテーブルを更新する場合は、テーブルの自動スケーリング設定とレプリカ設定を JSON ファイルからロードすると便利です。

次のコード例を使用するには、サンプル JSON ファイルを収録した auto-scaling.zip をダウンロードし、auto-scaling.json と replication.json を抽出します。ファイルのパスを書き留めておいてください。

この例では、JSON ファイルは現在のディレクトリにあります。さまざまなファイルパスオプションについては、「ファイルからパラメーターを読み込む方法」を参照してください。

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable \
--capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1 \
--auto-scaling-specification file://auto-scaling.json \
--replica-specifications file://replication.json
```

Amazon Keyspaces でマルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定を表示する

マルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定は、Amazon Keyspaces コンソール、CQL、または AWS CLIで表示できます。このセクションでは、CQL と を使用してこれを行う方法の例を示します AWS CLI。

Cassandra Query Language (CQL)

CQL を使用してマルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定を表示する

- マルチリージョンテーブルの自動スケーリング設定を表示するには、次のコマンドを使用します。

```
SELECT * FROM system_multiregion_info.autoscaling WHERE keyspace_name =
 'mykeyspace' AND table_name = 'mytable';
```

このコマンドの出力は、次のようになります。

```
 keyspace_name  | table_name | region         |
 provisioned_read_capacity_autoscaling_update

                                                                                |
 provisioned_write_capacity_autoscaling_update
---------------+------------+---------------
+-------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------------------
  mykeyspace    |  mytable   | ap-southeast-1 | {'minimum_units': 5,
 'maximum_units': 10, 'scaling_policy':
 {'target_tracking_scaling_policy_configuration': {'scale_out_cooldown':
 60, 'disable_scale_in': false, 'target_value': 50, 'scale_in_cooldown':
 60}}} | {'minimum_units': 5, 'maximum_units': 10, 'scaling_policy':
 {'target_tracking_scaling_policy_configuration': {'scale_out_cooldown': 0,
 'disable_scale_in': false, 'target_value': 50, 'scale_in_cooldown': 0}}}
  mykeyspace    |  mytable   | us-east-1      | {'minimum_units': 5,
 'maximum_units': 20, 'scaling_policy':
 {'target_tracking_scaling_policy_configuration': {'scale_out_cooldown':
 60, 'disable_scale_in': false, 'target_value': 70, 'scale_in_cooldown':
 60}}} | {'minimum_units': 5, 'maximum_units': 10, 'scaling_policy':
 {'target_tracking_scaling_policy_configuration': {'scale_out_cooldown': 0,
 'disable_scale_in': false, 'target_value': 50, 'scale_in_cooldown': 0}}}
  mykeyspace    |  mytable   | eu-west-1      | {'minimum_units': 5,
 'maximum_units': 10, 'scaling_policy':
 {'target_tracking_scaling_policy_configuration': {'scale_out_cooldown':
 60, 'disable_scale_in': false, 'target_value': 50, 'scale_in_cooldown':
 60}}} | {'minimum_units': 5, 'maximum_units': 10, 'scaling_policy':
 {'target_tracking_scaling_policy_configuration': {'scale_out_cooldown': 0,
 'disable_scale_in': false, 'target_value': 50, 'scale_in_cooldown': 0}}}
```

CLI

を使用してマルチリージョンテーブルのプロビジョニングされた容量と自動スケーリング設定を表示する AWS CLI

- マルチリージョンテーブルの自動スケーリング設定を表示するには、`get-table-auto-scaling-settings` オペレーションを使用します。次の CLI コマンドは一例です。

```
aws keyspaces get-table-auto-scaling-settings --keyspace-name mykeyspace --
table-name mytable
```

次のような出力が表示されます。

```
{
    "keyspaceName": "mykeyspace",
    "tableName": "mytable",
    "resourceArn": "arn:aws:cassandra:us-east-1:777788889999:/keyspace/
mykeyspace/table/mytable",
    "autoScalingSpecification": {
        "writeCapacityAutoScaling": {
            "autoScalingDisabled": false,
            "minimumUnits": 5,
            "maximumUnits": 10,
            "scalingPolicy": {
                "targetTrackingScalingPolicyConfiguration": {
                    "disableScaleIn": false,
                    "scaleInCooldown": 0,
                    "scaleOutCooldown": 0,
                    "targetValue": 50.0
                }
            }
        },
        "readCapacityAutoScaling": {
            "autoScalingDisabled": false,
            "minimumUnits": 5,
            "maximumUnits": 20,
            "scalingPolicy": {
                "targetTrackingScalingPolicyConfiguration": {
                    "disableScaleIn": false,
                    "scaleInCooldown": 60,
                    "scaleOutCooldown": 60,
                    "targetValue": 70.0
                }
            }
        }
    },
    "replicaSpecifications": [
        {
            "region": "us-east-1",
            "autoScalingSpecification": {
                "writeCapacityAutoScaling": {
                    "autoScalingDisabled": false,
                    "minimumUnits": 5,
                    "maximumUnits": 10,
```

```
                    "scalingPolicy": {
                        "targetTrackingScalingPolicyConfiguration": {
                            "disableScaleIn": false,
                            "scaleInCooldown": 0,
                            "scaleOutCooldown": 0,
                            "targetValue": 50.0
                        }
                    }
                },
                "readCapacityAutoScaling": {
                    "autoScalingDisabled": false,
                    "minimumUnits": 5,
                    "maximumUnits": 20,
                    "scalingPolicy": {
                        "targetTrackingScalingPolicyConfiguration": {
                            "disableScaleIn": false,
                            "scaleInCooldown": 60,
                            "scaleOutCooldown": 60,
                            "targetValue": 70.0
                        }
                    }
                }
            }
        },
        {
            "region": "eu-north-1",
            "autoScalingSpecification": {
                "writeCapacityAutoScaling": {
                    "autoScalingDisabled": false,
                    "minimumUnits": 5,
                    "maximumUnits": 10,
                    "scalingPolicy": {
                        "targetTrackingScalingPolicyConfiguration": {
                            "disableScaleIn": false,
                            "scaleInCooldown": 0,
                            "scaleOutCooldown": 0,
                            "targetValue": 50.0
                        }
                    }
                },
                "readCapacityAutoScaling": {
                    "autoScalingDisabled": false,
                    "minimumUnits": 5,
                    "maximumUnits": 10,
```

```
                    "scalingPolicy": {
                        "targetTrackingScalingPolicyConfiguration": {
                            "disableScaleIn": false,
                            "scaleInCooldown": 60,
                            "scaleOutCooldown": 60,
                            "targetValue": 50.0
                        }
                    }
                }
            }
        }
    ]
}
```

Amazon Keyspaces でテーブルの自動スケーリングを無効にする

このセクションでは、プロビジョンドキャパシティモードのマルチリージョンテーブルの自動スケーリングを無効にする方法の例を紹介します。この作業は、Amazon Keyspaces コンソール、CQL、または AWS CLIを使用して行うことができます。

> **Important**
>
> プロビジョンドキャパシティモードを使用するマルチリージョンテーブルでは、自動スケーリングを使用することをお勧めします。詳細については、「the section called “マルチリージョンテーブルのキャパシティの推定”」を参照してください。

> **Note**
>
> Application Auto Scaling で使用されているサービスリンクロールを削除するには、すべての AWS リージョンにおいてアカウント内のすべてのテーブルで自動スケーリングを無効にする必要があります。

Console

コンソールで既存のマルチリージョンテーブルの Amazon Keyspaces 自動スケーリングを無効にする

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. 操作対象のテーブルを選択してから、[キャパシティ] を選択します。
3. [キャパシティ設定] セクションで、[編集] を選択します。
4. Amazon Keyspaces の自動スケーリングを無効にするには、[自動スケーリング] チェックボックスの選択を解除します。自動スケーリングを無効にすると、Application Auto Scaling のスケーラブルターゲットとしてのテーブルの登録が解除されます。Amazon Keyspaces テーブルにアクセスするために Application Auto Scaling が使用するサービスリンクロールを削除するには、「the section called “Amazon Keyspaces のサービスリンクロールの削除”」の手順を実行してください。
5. 自動スケーリング設定が定義されたら、[Save (保存)] を選択します。

Cassandra Query Language (CQL)

CQL を使用してマルチリージョンテーブルの自動スケーリングを無効にする

- ALTER TABLE を使用して、既存のテーブルの自動スケーリングを無効にすることができます。テーブルレプリカの自動スケーリングを個別に無効化することはできません。

  次の例では、テーブルの読み取りキャパシティに対する自動スケーリングを無効にしています。

```
ALTER TABLE mykeyspace.mytable
WITH AUTOSCALING_SETTINGS = {
    'provisioned_read_capacity_autoscaling_update': {
        'autoscaling_disabled': true
    }
};
```

CLI

を使用してマルチリージョンテーブルの自動スケーリングをオフにする AWS CLI

- コマンドを使用して AWS CLI update-table、既存のテーブルの自動スケーリングをオフにできます。テーブルレプリカの自動スケーリングを個別に無効化することはできません。

  次の例では、テーブルの読み取りキャパシティに対する自動スケーリングを無効にしています。

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable
          \ --auto-scaling-specification
 readCapacityAutoScaling={autoScalingDisabled=true}
```

Amazon Keyspaces でマルチリージョンテーブルにプロビジョニングされるキャパシティを手動で設定する

マルチリージョンテーブルの自動スケーリングを無効にする必要がある場合は、CQL または AWS CLIを使用して、レプリカテーブル用の読み取りキャパシティを手動でプロビジョニングすることができます。

> **Note**
>
> プロビジョンドキャパシティモードを使用するマルチリージョンテーブルでは、自動スケーリングを使用することをお勧めします。詳細については、「the section called “マルチリージョンテーブルのキャパシティの推定”」を参照してください。

Cassandra Query Language (CQL)

CQL を使用してマルチリージョンテーブルにプロビジョニングされるキャパシティを手動で設定する

- ALTER TABLE を使用して、レプリカテーブルの読み取りキャパシティを手動でプロビジョニングすることができます。

```
ALTER TABLE mykeyspace.mytable
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
```

```
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 1,
        'write_capacity_units': 1
    },
    'replica_updates': {
        'us-east-1': {
            'read_capacity_units': 2
         }
    }
};
```

CLI

を使用してマルチリージョンテーブルのプロビジョニングされた容量を手動で設定する AWS CLI

- マルチリージョンテーブルの自動スケーリングを無効にする必要がある場合は、update-table を使用して、レプリカテーブルの読み取りキャパシティを手動でプロビジョニングすることができます。

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable \
--capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1 \
--replica-specifications region="us-east-1",readCapacityUnits=5
```

Amazon Keyspaces のポイントインタイムリカバリでデータをバックアップおよび復元する

ポイントインタイムリカバリ (PITR) により、テーブルデータの継続的なバックアップが可能になるため、Amazon Keyspaces テーブルの誤った書き込みオペレーションや削除オペレーションを防止できます。

例えば、テストスクリプトで、誤って本稼働環境の Amazon Keyspaces テーブルに書き込みを行ったとします。ポイントインタイムリカバリにより、過去 35 日以内に PITR を有効にした後の任意の時点の状態にテーブルデータを復元することができます。ポイントインタイムリカバリが有効になっているテーブルを削除した場合、35 日以内に削除されたテーブルのデータをクエリすれば (追加料金なし)、削除直前の状態に復元することができます。

コンソール、 AWS SDK と AWS Command Line Interface （ AWS CLI ）、または Cassandra クエリ言語 (CQL) を使用して、Amazon Keyspaces テーブルを特定の時点に復元できます。詳細については、「Amazon Keyspaces でポイントインタイムリカバリを使用する」を参照してください。

ポイントインタイムオペレーションのパフォーマンスや可用性がベーステーブルに影響を及ぼすことはなく、テーブルを復元しても追加のスループットは消費されません。

PITR クォータの詳細については「クォータ」を参照してください。

料金については、「Amazon Keyspaces (for Apache Cassandra) pricing (Amazon Keyspaces (Apache Cassandra 向け) の料金)」を参照してください。

トピック

- Amazon Keyspaces のポイントインタイムリカバリの仕組み
- Amazon Keyspaces でポイントインタイムリカバリを使用する

Amazon Keyspaces のポイントインタイムリカバリの仕組み

このセクションでは、Amazon Keyspaces ポイントインタイムリカバリ (PITR) の仕組みの概要について説明します。料金の詳細については、「Amazon Keyspaces (for Apache Cassandra) pricing (Amazon Keyspaces (Apache Cassandra 向け) の料金)」 を参照してください。

トピック

- PITR 継続的バックアップの時間ウィンドウ
- PITR 復元設定
- 暗号化されたテーブルの PITR 復元
- PITR によるマルチリージョンテーブルの復元
- ユーザー定義タイプ (UDTs) のテーブルの PITR 復元
- PITR によるテーブル復元時間
- Amazon Keyspaces の PITR および AWS サービスとの統合

PITR 継続的バックアップの時間ウィンドウ

Amazon Keyspaces PITR では、復元可能なバックアップをテーブルに使用できる時間枠を維持するために、2 つのタイムスタンプが使用されます。

- 最古の復元可能時間 – 最も古い復元可能バックアップの時刻をマークします。最古の復元可能バックアップは、35 日前、または PITR が有効化された時点のいずれか新しい方までさかのぼります。35 日の最大バックアップウィンドウは変更できません。
- 現在の時刻 – 最新の復元可能バックアップのタイムスタンプが現在の時刻です。復元中にタイムスタンプが指定されない場合は、現在の時刻が使用されます。

PITR を有効にすると、`EarliestRestorableDateTime` から `CurrentTime` までの期間の任意の時点の状態まで復元できます。テーブルデータは、PITR が有効化された時点までしか復元できません。

PITR を無効にして後で再度有効にした場合は、最初の使用可能バックアップの開始時刻を、PITR が再有効化された時刻にリセットします。つまり、PITR を無効にすると、バックアップ履歴が消去されるということです。

> **Note**
>
> テーブルでのデータ定義言語 (DDL) オペレーション (スキーマの変更など) は、非同期で実行されます。復元されたテーブルデータには完了したオペレーションのみが表示されますが、復元時にオペレーションが進行中である場合は、ソーステーブルで追加のアクションが表示される可能性があります。DDL ステートメントのリストについては、「the section called "DDL ステートメント"」を参照してください。

復元するテーブルはアクティブでなくても構いません。削除されたテーブルで PITR が有効になっていて、バックアップウィンドウ内 (または過去 35 日以内) に削除が行われた場合でも、削除されたテーブルを復元できます。

> **Note**
>
> 以前に削除されたテーブルと同じ修飾名 (mykeyspace.mytable など) を用いて新しいテーブルが作成された場合、削除されたテーブルは復元できなくなります。コンソールでこれを実行すると警告が表示されます。

PITR 復元設定

PITR を使用してテーブルを復元すると、Amazon Keyspaces により、ソーステーブルのスキーマとデータを、新しいテーブルに対して選択したタイムスタンプ (day:hour:minute:second) に基づいた状態に復元します。既存のテーブルは PITR によりオーバーライドされません。

PITR では、テーブルのスキーマとデータに加えて、custom_properties ソーステーブルからの復元も行われます。カスタムプロパティについては、最古の復元時刻から現在の時刻までの範囲で選択したタイムスタンプに基づいて復元されるテーブルのデータとは異なり、常に現在の時刻のテーブル設定に基づいて復元されます。

復元されたテーブルの設定は、タイムスタンプが復元開始時であるソーステーブルの設定と一致します。これらの設定は復元中にオーバーライドすることができるので、その場合は WITH custom_properties を使用します。カスタムプロパティには以下の設定が含まれます。

- 読み取り/書き込みキャパシティモード
- プロビジョンドスループット性能設定
- PITR 設定

テーブルがプロビジョンドキャパシティモードで、自動スケーリングが有効になっている場合、復元オペレーションはテーブルの自動スケーリング設定も復元します。これらの設定は、CQL の autoscaling_settings パラメータや CLI の autoScalingSpecification を使用して上書きできます。自動スケーリング設定の詳細については、「the section called “自動スケーリングによるスループットキャパシティの管理”」を参照してください。

テーブル全体を復元する場合、復元済みテーブルのすべてのテーブル設定は、復元時の送信元のテーブルの現在の設定から取得されます。

たとえば、テーブルのプロビジョニングされたスループットが 50 読み込みキャパシティーユニットおよび 50 書き込みキャパシティーユニットに最近下げられたとします。その後、このテーブルを 3 週間前の状態に復元します。このとき、プロビジョンドスループットの読み取りキャパシティユニットは 100 に、書き込みキャパシティユニットも 100 に設定されました。この場合、Amazon Keyspaces では、テーブルデータは指定の時点の状態に復元されますが、プロビジョンドスループット設定は最新の設定 (読み取りキャパシティユニット 50、書き込みキャパシティユニット 50) が使用されます。

次の設定は復元されないため、新しいテーブルに対して手動で設定する必要があります。

- Amazon Keyspaces 変更データキャプチャ (CDC) ストリーム

- AWS Identity and Access Management (IAM) ポリシー
- Amazon CloudWatch メトリクスおよびアラーム
- タグ (`WITH TAGS` を使用して CQL `RESTORE` ステートメントに追加できる)

暗号化されたテーブルの PITR 復元

PITR を使用してテーブルを復元すると、Amazon Keyspaces によりソーステーブルの暗号化設定が復元されます。テーブルが AWS 所有のキー (デフォルト) で暗号化されている場合、テーブルは同じ設定で自動的に復元されます。復元するテーブルがカスタマーマネージドキーを使用して暗号化されている場合は、テーブルデータを復元するために同じカスタマーマネージドキーを使用して Amazon Keyspaces にアクセスできる必要があります。

テーブルの暗号化設定は復元時に変更できます。から AWS 所有のキー カスタマーマネージドキーに変更するには、復元時に有効でアクセス可能なカスタマーマネージドキーを指定する必要があります。

カスタマーマネージドキーから に変更する場合は AWS 所有のキー、Amazon Keyspaces がソーステーブルのカスタマーマネージドキーにアクセスして でテーブルを復元していることを確認します AWS 所有のキー。テーブルの保管データ暗号化設定の詳細については、「the section called “動作”」を参照してください。

> **Note**
>
> Amazon Keyspaces がカスタマーマネージドキーにアクセスできなくなったためにテーブルが削除された場合は、そのテーブルを復元する前に、カスタマーマネージドキーが Amazon Keyspaces にアクセスできるか確認する必要があります。カスタマーマネージドキーで暗号化されたテーブルは、Amazon Keyspaces がそのキーにアクセスできない場合に復元できません。詳細については、 AWS Key Management Service デベロッパーガイドの「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。

PITR によるマルチリージョンテーブルの復元

PITR を使用してマルチリージョンテーブルを復元できます。復元オペレーションを成功させるには、ソーステーブルのすべてのレプリカで PITR を有効にし、ソーステーブルと宛先テーブルの両方を同じ にレプリケートする必要があります AWS リージョン。

Amazon Keyspaces は、キースペースに含まれるレプリケート先の各リージョンでソーステーブルの設定を復元します。復元操作中に設定を上書きすることもできます。復元中に変更できる設定の詳細については、「the section called “復元設定”」を参照してください。

マルチリージョンレプリケーションの詳細については、「」を参照してくださいthe section called “仕組み”。

ユーザー定義タイプ (UDTs) のテーブルの PITR 復元

UDTs を使用するテーブルを復元できます。復元オペレーションを成功させるには、参照されるUDTsが存在し、キースペースで有効である必要があります。

テーブルの復元時に必要な UDT がない場合、Amazon Keyspaces は自動的に UDT スキーマの復元を試み、テーブルの復元を続行します。

UDT を削除して再作成した場合、Amazon Keyspaces は UDT の新しいスキーマを使用して UDT を復元し、元の UDT スキーマを使用してテーブルを復元するリクエストを拒否します。この場合、古い UDT スキーマを使用してテーブルを復元する場合は、テーブルを新しいキースペースに復元できます。UDT を削除して再作成する場合、再作成された UDT のスキーマが削除された UDT のスキーマと同じであっても、再作成された UDT は新しい UDT と見なされます。この場合、Amazon Keyspaces は古い UDT スキーマを使用してテーブルを復元するリクエストを拒否します。

UDT が欠落していて、Amazon Keyspaces が UDT の復元を試みると、リージョン内のアカウントの UDTs、試行は失敗します。

UDT クォータとデフォルト値の詳細については、「」を参照してくださいthe section called “Amazon Keyspaces のユーザー定義タイプ (UDTsクォータとデフォルト値”。UDTs「」を参照してくださいthe section called “ユーザー定義タイプ (UDTs)”。

PITR によるテーブル復元時間

テーブルの復元にかかる時間は複数の要因に基づいており、テーブルのサイズに直接関連しているとは限りません。

復元時間に関する考慮事項を次に示します。

- 新しいテーブルにバックアップを復元します。新しいテーブルを作成して復元プロセスを開始するためのすべてのアクションを実行するのに、テーブルが空でも最大で 20 分かかることがあります。
- データモデルが適切に分散されている大きなテーブルの復元時間は数時間以上になる可能性があります。

- ソーステーブルに大きく歪んだデータが含まれている場合は復元時間は長くなることがあります。例えば、テーブルのプライマリキーにより 1 年のうちの 1 か月がパーティショニングに使われており、すべてのデータが 12 月のものだった場合は、データを歪めています。

災害対策を計画する際のベストプラクティスは、平均復元完了時間を定期的に記録し、これらの時間が目標復旧時間全体にどのように影響するかを確認することです。

Amazon Keyspaces の PITR および AWS サービスとの統合

次の PITR オペレーションは、継続的なモニタリングと監査を有効にする AWS CloudTrail ために を使用してログに記録されます。

- PITR が有効または無効になっている新規のテーブルを作成します。
- 既存のテーブルで PITR を有効または無効にします。
- アクティブなテーブルまたは削除されたテーブルを復元します。

詳細については、「を使用した Amazon Keyspaces API コールのログ記録 AWS CloudTrail」を参照してください。

AWS CloudFormationを使用して以下の PITR アクションを実行できます。

- PITR が有効または無効になっている新規のテーブルを作成します。
- 既存のテーブルで PITR を有効または無効にします。

詳細については、『AWS CloudFormation ユーザーガイド』の「Cassandra Resource Type Reference(Cassandra リソースタイプのリファレンス)」を参照してください。

Amazon Keyspaces でポイントインタイムリカバリを使用する

Amazon Keyspaces (Apache Cassandra 向け) では、ポイントインタイム復元 (PITR) を使用して、特定の時点の状態にテーブルを復元できます。PITR では、過去 35 日以内の以前の状態にテーブルを復元することができ、データの保護と復旧の機能として利用できます。この機能は、間違えてデータを削除してしまった場合やアプリケーションエラーが起きた場合、またはテストの目的などで活躍します。データをすばやく効率的に復旧し、ダウンタイムとデータ損失を最小限に抑えることができます。以降のセクションでは、Amazon Keyspaces で PITR を使用してテーブルを復元し、データの整合性とビジネスの継続性を確保するプロセスについて説明します。

トピック

- Amazon Keyspaces PITR のテーブル復元用の IAM アクセス許可を設定する
- Amazon Keyspaces でテーブルの PITR を設定する
- Amazon Keyspaces テーブルの PITR を無効にする
- Amazon Keyspaces でテーブルをバックアップから特定の時点に復元する
- 削除済みのテーブルを Amazon Keyspaces PITR を使用して復元する

Amazon Keyspaces PITR のテーブル復元用の IAM アクセス許可を設定する

このセクションでは、Amazon Keyspaces テーブルを復元するための AWS Identity and Access Management (IAM) プリンシパルのアクセス許可を設定する方法を要約します。IAM では、 AWS マネージドポリシーである AmazonKeyspacesFullAccess に、Amazon Keyspaces テーブルの復元に必要なアクセス許可が含まれています。最低限必要なアクセス許可を含めたカスタムポリシーを実装するには、次のセクションで概説している要件を検討してください。

テーブルを正常に復元するには、IAM プリンシパルに次の最小限のアクセス許可が必要です。

- cassandra:Restore — この復元アクションはターゲットテーブルの復元に必須です。
- cassandra:Select — この選択アクションはソーステーブルからの読み取りに必須です。
- cassandra:TagResource — タグアクションはオプションで、復元オペレーションによりタグが追加される場合にのみ必須です。

次の例は、キースペース mykeyspace 内のテーブルを復元するために必要な最小限のアクセス許可をユーザーに与えるポリシーを示しています。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Restore",
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/*",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
```

```
    ]
}
```

選択した他の機能に基づいて、テーブルを復元するための追加の許可が必要になる場合があります。例えば、ソーステーブルが保管時にカスタマーマネージドキーで暗号化されている場合、テーブルを正常に復元するために、Amazon Keyspaces にはソーステーブルのカスタマーマネージドキーへのアクセス許可が必要になります。詳細については、「the section called “PITR と暗号化されたテーブル”」を参照してください。

IAM ポリシーを条件キーとともに使用して特定のソースへの受信トラフィックを制限するには、プリンシパルの代わりに復元オペレーションを実行する許可が Amazon Keyspaces に付与されていることを確認する必要があります。ポリシーにより受信トラフィックが次のいずれかに制限されている場合は、IAM ポリシーに `aws:ViaAWSService` 条件キーを追加する必要があります。

- `aws:SourceVpce` の場合は VPC エンドポイント
- `aws:SourceIp` の場合は IP レンジ
- `aws:SourceVpc` の場合は VPC

AWS サービスでプリンシパルの認証情報を使用してリクエストを実行すると、`aws:ViaAWSService` 条件キーによりアクセスが許可されます。詳細については、『IAM ユーザーガイド』の「IAM JSON policy elements: Condition key(IAM JSON ポリシー要素: 条件キー)」を参照してください。

以下は、ソーストラフィックを特定の IP アドレスに制限し、プリンシパルの代わりに Amazon Keyspaces によってテーブルが復元されるようにするポリシーの例です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"CassandraAccessForCustomIp",
         "Effect":"Allow",
         "Action":"cassandra:*",
         "Resource":"*",
         "Condition":{
            "Bool":{
               "aws:ViaAWSService":"false"
            },
            "ForAnyValue:IpAddress":{
```

```
                "aws:SourceIp":[
                   "123.45.167.89"
                ]
             }
          }
       },
       {
          "Sid":"CassandraAccessForAwsService",
          "Effect":"Allow",
          "Action":"cassandra:*",
          "Resource":"*",
          "Condition":{
             "Bool":{
                "aws:ViaAWSService":"true"
             }
          }
       }
    ]
}
```

aws:ViaAWSService グローバル条件キーを使用したポリシーの例については、「the section called “VPC エンドポイントポリシーと Amazon Keyspaces ポイントインタイムリカバリ (PITR)”」を参照してください。

Amazon Keyspaces でテーブルの PITR を設定する

コンソール、CQL、および AWS CLIで、PITR を使用したバックアップおよび復元のオペレーションを Amazon Keyspaces のテーブルに設定できます。

CQL または を使用して新しいテーブルを作成する場合は AWS CLI、テーブル作成ステートメントで PITR を明示的に有効にする必要があります。コンソールで新しいテーブルを作成した場合は、PITR がデフォルトで有効になります。

テーブルの復元方法については、「the section called “テーブルをある時点に復元する”」を参照してください。

Console

コンソールを使用してテーブルの PITR を設定する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。

2. ナビゲーションペインで [Tables (テーブル)] を選択し、編集するテーブルを選択します。
3. [Backups (バックアップ)] タブで、[Edit (編集)] を選択します。
4. [Edit point-in-time recovery settings (ポイントインタイムリカバリ設定の編集)] セクションで、[Enable Point-in-time recovery (ポイントタイムリカバリを有効にする)] チェックボックスを選択します。
5. [Save changes] (変更の保存) をクリックします。

Cassandra Query Language (CQL)

CQL を使用してテーブルの PITR を設定する

1. テーブルの PITR 設定は、`point_in_time_recovery` カスタムプロパティを使用して管理することができます。

   新しいテーブルの作成時に PITR を有効にするには、`enabled` のステータスを `point_in_time_recovery` に設定する必要があります。例えば、次の CQL コマンドを使用できます。

```
CREATE TABLE "my_keyspace1"."my_table1"(
 "id" int,
 "name" ascii,
 "date" timestamp,
 PRIMARY KEY("id"))
WITH CUSTOM_PROPERTIES = {
 'capacity_mode':{'throughput_mode':'PAY_PER_REQUEST'},
 'point_in_time_recovery':{'status':'enabled'}
}
```

   > **Note**
   >
   > ポイントインタイムリカバリのカスタムプロパティが指定されていない場合、デフォルトにより、ポイントインタイムリカバリは無効になります。

2. CQL を使用して既存のテーブルの PITR を有効にするには、次の CQL コマンドを実行します。

```
ALTER TABLE mykeyspace.mytable
WITH custom_properties = {'point_in_time_recovery': {'status': 'enabled'}}
```

CLI

を使用してテーブルの PITR を設定する AWS CLI

1. テーブルの PITR 設定は、UpdateTable API を使用して管理することができます。

   新しいテーブルの作成時に PITR を有効にするには、create-table コマンドに point-in-time-recovery 'status=ENABLED' を指定する必要があります。例えば、次の AWS CLI コマンドを使用できます。このコマンドは読みやすくするために別々の行に分割されています。

```
aws keyspaces create-table --keyspace-name 'myKeyspace' --table-name 'myTable'
            --schema-definition 'allColumns=[{name=id,type=int},
{name=name,type=text},{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --point-in-time-recovery 'status=ENABLED'
```

   > **Note**
   >
   > ポイントインタイムリカバリ値が指定されていない場合、デフォルトにより、ポイントインタイムリカバリは無効になります。

2. テーブルのpoint-in-timeリカバリ設定を確認するには、次の AWS CLI コマンドを使用します。

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

3. を使用して既存のテーブルの PITR を有効にするには AWS CLI、次のコマンドを実行します。

```
aws keyspaces update-table --keyspace-name 'myKeyspace' --table-name 'myTable'
 --point-in-time-recovery 'status=ENABLED'
```

Amazon Keyspaces テーブルの PITR を無効にする

Amazon Keyspaces テーブルの PITR は、コンソール、CQL、または AWS CLIを使用して、いつでも無効にすることができます。

> **Important**
>
> PITR を無効にすると、35 日以内にテーブルの PITR を再度有効にしても、バックアップ履歴が直ちに削除されます。

テーブルの復元方法については、「the section called “テーブルをある時点に復元する”」を参照してください。

Console

コンソールを使用してテーブルの PITR を無効にする

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで [Tables (テーブル)] を選択し、編集するテーブルを選択します。
3. [Backups (バックアップ)] タブで、[Edit (編集)] を選択します。
4. [Edit point-in-time recovery settings (ポイントインタイムリカバリ設定の編集)] セクションで、[Enable Point-in-time recovery (ポイントタイムリカバリを有効にする)] チェックボックスの選択を解除します。
5. [Save changes] (変更の保存) をクリックします。

Cassandra Query Language (CQL)

CQL を使用してテーブルの PITR を無効にする

- 既存のテーブルの PITR を無効にするには、次の CQL コマンドを実行します。

```
ALTER TABLE mykeyspace.mytable
WITH custom_properties = {'point_in_time_recovery': {'status': 'disabled'}}
```

CLI

を使用してテーブルの PITR を無効にする AWS CLI

- 既存のテーブルの PITR を無効にするには、次の AWS CLI コマンドを実行します。

```
aws keyspaces update-table --keyspace-name 'myKeyspace' --table-name 'myTable'
 --point-in-time-recovery 'status=DISABLED'
```

Amazon Keyspaces でテーブルをバックアップから特定の時点に復元する

既存の Amazon Keyspaces テーブルを特定の時点の状態に復元する実際の方法を紹介します。

> **Note**
>
> この手順では、操作対象のテーブルにポイントインタイムリカバリが設定済みであることが前提です。テーブルの PITR を有効にするには、「the section called “PITR の設定”」を参照してください。

> **Important**
>
> 復元中は、復元を実行するアクセス許可を IAM プリンシパル (ユーザー、グループ、ロールなど) に付与する AWS Identity and Access Management (IAM) ポリシーを変更または削除しないでください。行った場合、予期しない動作が発生する場合があります。例えば、テーブルの復元中にそのテーブルへの書き込みアクセス許可を削除してしまうと、基盤となる `RestoreTableToPointInTime` オペレーションが復元済みデータをテーブルに書き込むことができなくなります。
> 復元オペレーションが完了したら、アクセス許可を変更または削除してもかまいません。

Console

コンソールを使用してテーブルを特定の時点の状態に復元する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces 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、[テーブル] を選択します。
3. テーブルのリストで、復元対象のテーブルを選択します。
4. テーブルの [バックアップ] タブの [ポイントインタイムリカバリ] セクションで、[復元] を選択します。

5. 新しいテーブル名には、復元後のテーブルの新しい名前を入力します (例: **`mytable_restored`**)。
6. 復元オペレーションのポイントインタイムを定義するために、次の 2 つのオプションのいずれかを選択できます。
   - あらかじめ設定されている [Earliest] (最も早い時間) を選択します。
   - [Specify date and time] (日時の指定) を選択し、新しいテーブルの復元時点の日付と時刻を入力します。

   > **Note**
   >
   > [Earliest] (最古) の時点から現在の時点までの期間内の任意の特定時点の状態に復元できます。Amazon Keyspaces により、選択した日時 (day:hour:minute:second) に基づいた状態にテーブルデータが復元されます。

7. [Restore (復元)] を選択して、復元プロセスを開始します。

   復元中のテーブルのステータスは、[Restoring (復元中)] と表示されます。復元が終わると、復元後のテーブルのステータスが [アクティブ] に変わります。

Cassandra Query Language (CQL)

CQL を使用してテーブルを特定の時点の状態に復元する

1. アクティブなテーブルを、`earliest_restorable_timestamp` から現在の時点までの範囲内の任意の時点の状態に復元できます。デフォルトは現在の時刻です。

   テーブルでポイントインタイムリカバリが有効になっていることを確認するには、次の例に示すように、`system_schema_mcs.tables` をクエリします。

   ```
   SELECT custom_properties
   FROM system_schema_mcs.tables
   WHERE keyspace_name = 'mykeyspace' AND table_name = 'mytable';
   ```

   次のサンプル出力に示すように、ポイントインタイムリカバリが有効になります。

   ```
   custom_properties
   -----------------
   {
   ```

```
  ...,
    "point_in_time_recovery": {
    "earliest_restorable_timestamp":"2020-06-30T19:19:21.175Z"
    "status":"enabled"
  }
}
```

2. • テーブルを現在の時点に復元します。WITH restore_timestamp = ... 句を省略すると、現在のタイムスタンプが使用されます。

```
RESTORE TABLE mykeyspace.mytable_restored
FROM TABLE mykeyspace.mytable;
```

• restore_timestamp に ISO 8601 形式で特定の時点を定義し、その時点の状態に復元することもできます。過去 35 日間の任意の時点を指定できます。たとえば、次のコマンドでは EarliestRestorableDateTime にテーブルを復元します。

```
RESTORE TABLE mykeyspace.mytable_restored
FROM TABLE mykeyspace.mytable
WITH restore_timestamp = '2020-06-30T19:19:21.175Z';
```

構文の完全な説明については、言語リファレンスの「the section called “RESTORE TABLE”」を参照してください。

3. テーブルの復元が成功したことを確認するには、system_schema_mcs.tables をクエリしてテーブルのステータスを確認します。

```
SELECT status
FROM system_schema_mcs.tables
WHERE keyspace_name = 'mykeyspace' AND table_name = 'mytable_restored'
```

クエリには次の出力が表示されます。

```
status
------
RESTORING
```

復元中のテーブルのステータスは、[Restoring (復元中)] と表示されます。復元が終わると、テーブルのステータスが [アクティブ] に変わります。

CLI

を使用してテーブルを特定の時点に復元する AWS CLI

1. PITR を有効にした myTable という名前の簡単なテーブルを作成します。コマンドは読みやすくするため、別々の行に分割されています。

```
aws keyspaces create-table --keyspace-name 'myKeyspace' --table-name 'myTable'
            --schema-definition 'allColumns=[{name=id,type=int},
{name=name,type=text},{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --point-in-time-recovery 'status=ENABLED'
```

2. 新しいテーブルのプロパティを確認し、PITR 用の earliestRestorableTimestamp を確認してください。

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

このコマンドの出力では、以下の値が返ります。

```
{
    "keyspaceName": "myKeyspace",
    "tableName": "myTable",
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/
myKeyspace/table/myTable",
    "creationTimestamp": "2022-06-20T14:34:57.049000-07:00",
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "id",
                "type": "int"
            },
            {
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
            }
        ],
        "partitionKeys": [
```

```
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2022-06-20T14:34:57.049000-07:00"
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "ENABLED",
        "earliestRestorableTimestamp": "2022-06-20T14:35:13.693000-07:00"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    }
}
```

3. • 特定の時点の状態にテーブルを復元するには、ISO 8601 形式で `restore_timestamp` を指定します。過去 35 日間の任意の時点を 1 秒間隔で選択できます。たとえば、次のコマンドでは `EarliestRestorableDateTime` にテーブルを復元します。

```
aws keyspaces restore-table --source-keyspace-name 'myKeyspace' --source-
table-name 'myTable' --target-keyspace-name 'myKeyspace' --target-table-name
 'myTable_restored' --restore-timestamp "2022-06-20 21:35:14.693"
```

このコマンドの出力は、復元されたテーブルの ARN を返します。

```
{
    "restoredTableARN": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/
myKeyspace/table/myTable_restored"
}
```

• テーブルを現在の時点に復元するには、`restore-timestamp` パラメータを省略できます。

```
aws keyspaces restore-table --source-keyspace-name 'myKeyspace' --source-
table-name 'myTable' --target-keyspace-name 'myKeyspace' --target-table-name
 'myTable_restored1'"
```

削除済みのテーブルを Amazon Keyspaces PITR を使用して復元する

次の手順は、削除されたテーブルをバックアップから削除時点の状態に復元する方法を示しています。これを行うには、CQL または を使用します AWS CLI。

> **Note**
>
> この手順では、削除されたテーブルに対して PITR が有効になっていることを前提としています。

Cassandra Query Language (CQL)

CQL を使用して削除済みテーブルを復元する

1. 削除されたテーブルに対してポイントインタイムリカバリが有効になっていることを確認するには、システムテーブルのクエリを実行します。ポイントインタイムリカバリが有効になっているテーブルのみが表示されます。

   ```
   SELECT custom_properties
   FROM system_schema_mcs.tables_history
   WHERE keyspace_name = 'mykeyspace' AND table_name = 'my_table';
   ```

   クエリには次の出力が表示されます。

   ```
   custom_properties
   ------------------
   {
       ...,
      "point_in_time_recovery":{
         "restorable_until_time":"2020-08-04T00:48:58.381Z",
         "status":"enabled"
      }
   ```

```
}
```

2. 次のサンプルステートメントを使用して、テーブルを削除時点の状態に復元します。

```
RESTORE TABLE mykeyspace.mytable_restored
FROM TABLE mykeyspace.mytable;
```

CLI

を使用して削除されたテーブルを復元する AWS CLI

1. 事前に作成し、PITR を有効にしておいたテーブルを削除します。コマンドの例を次に示します。

```
aws keyspaces delete-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

2. 次のコマンドを使用して、削除されたテーブルを削除時点の状態に復元します。

```
aws keyspaces restore-table --source-keyspace-name 'myKeyspace' --source-
table-name 'myTable' --target-keyspace-name 'myKeyspace' --target-table-name
 'myTable_restored2'
```

このコマンドの出力は、復元されたテーブルの ARN を返します。

```
{
    "restoredTableARN": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/
myKeyspace/table/myTable_restored2"
}
```

Amazon Keyspaces (Apache Cassandra 向け) で有効期限 (TTL) を使用してデータを期限切れにする

Amazon Keyspaces (Apache Cassandra 向け) の有効期限 (TTL) を使用すると、テーブルのデータを自動的に期限切れにして、アプリケーションロジックを簡素化し、ストレージの料金を最適化できます。不要になったデータは、設定した有効期限の値に基づいて、テーブルから自動的に削除されます。

これにより、必要なデータ保持期間の定義や必要なデータ削除時期の指定を行う、ビジネス/産業/規制要件に基づいたデータ保持ポリシーを容易に順守できるようになります。

例えば、AdTech アプリケーションで TTL を使用して、特定の広告のデータが期限切れになり、クライアントに表示されなくなるタイミングを予定することができます。また、TTL を使用すれば古くなったデータの使用が自動的に停止されるので、ストレージコストを節約できます。

テーブル全体にデフォルトの TTL 値を設定し、個々の行と列に対してその値をオーバーライドできます。TTL オペレーションはアプリケーションのパフォーマンスに影響しません。また、期限切れにするために TTL でマークされた行と列の数は、テーブルの可用性に影響しません。

Amazon Keyspaces では、期限切れのデータがクエリ結果で返されたり、データ操作言語 (DML) ステートメントで使用されたりしないように、期限切れのデータが自動的に除外されます。Amazon Keyspaces では、通常、有効期限後 10 日以内にストレージから期限切れデータが削除されます。

まれなケースですが、可用性を確保するために基盤となるストレージパーティションに持続的なアクティビティが存在する場合、Amazon Keyspaces では期限切れから 10 日以内にデータを削除できないことがあります。このような場合、Amazon Keyspaces では、パーティションのトラフィックが減少しても、期限切れデータの削除が連続で試行されます。

データがストレージから完全に削除されると、ストレージ料金が発生しなくなります。

コンソール、Cassandra クエリ言語 (CQL)、または AWS CLIを使用して、新規および既存のテーブルのデフォルトの TTL を設定、変更、無効化することができます。

デフォルトの TTL が設定されているテーブルでは、CQL ステートメントを使用してデフォルトの TTL 設定をオーバーライドし、カスタムの TTL 値を行や列に適用することができます。詳細については、「the section called “INSERT を使用した新しい行のカスタム TTL の設定”」および「the section called “UPDATE を使用した行と列のカスタム TTL 値の設定”」を参照してください。

TTL の料金は、有効期限を使用して削除または更新する行のサイズに基づきます。TTL オペレーションは、TTL `deletes` の単位で計測されます。削除または更新される行ごとにデータ 1 KB あたり 1 回の TTL 削除が消費されます。

例えば、2.5 KB のデータが保存されている行を更新して、行内の 1 つ以上の列を同時に削除する場合は、3 回の TTL 削除が必要です。また、3.5 KB のデータが含まれている 1 行全体を削除する場合は、4 回の TTL 削除が必要です。

行ごとに削除データ 1 KB あたり 1 回の TTL 削除が消費されます。料金の詳細については、「Amazon Keyspaces (for Apache Cassandra) pricing」(Amazon Keyspaces (Apache Cassandra 向け) の料金) を参照してください。

トピック

- Amazon Keyspaces の 有効期限 (TTL) および AWS サービスとの統合
- デフォルトの有効期限 (TTL) を設定した新規テーブルを作成する
- テーブルのデフォルトの有効期間 (TTL) 値を更新する
- カスタムの有効期限 (TTL) 設定を有効にしてテーブルを作成する
- カスタムの有効期限 (TTL) でテーブルを更新する
- INSERT ステートメントを使用して新しい行のカスタムの有効期限 (TTL) 値を設定する
- UPDATE ステートメントを使用して行と列のカスタム有効期限 (TTL) 設定を編集する

Amazon Keyspaces の 有効期限 (TTL) および AWS サービスとの統合

次の TTL メトリクスは Amazon CloudWatch で継続的モニタリングを有効にするために使用できます。

- `TTLDeletes` - 有効期限 (TTL) を使用して行のデータを削除または更新するために消費された単位。

CloudWatch のメトリクスのモニタリング方法については、「the section called “CloudWatch によるモニタリング”」を参照してください。

を使用すると AWS CloudFormation、Amazon Keyspaces テーブルの作成時に TTL を有効にできます。詳細については、「AWS CloudFormation ユーザーガイド」を参照してください。

デフォルトの有効期限 (TTL) を設定した新規テーブルを作成する

Amazon Keyspaces では、テーブルの作成時に、テーブルのすべての行にデフォルトの TTL 値を設定することができます。

テーブルのデフォルト TTL 値はゼロです。つまり、データは自動的に期限切れになりません。テーブルのデフォルト TTL 値がゼロより大きい場合、各行に有効期限タイムスタンプが追加されます。

TTL 値は秒単位で設定され、設定可能な最大値は 630,720,000 秒で、これは 20 年に相当します。

テーブルを作成した後で、CQL DML ステートメントを使用して、特定の行または列に対してテーブルのデフォルトの TTL 設定を上書きできます。詳細については、「the section called “INSERT を使用した新しい行のカスタム TTL の設定”」および「the section called “UPDATE を使用した行と列のカスタム TTL 値の設定”」を参照してください。

テーブルで TTL を有効にすると、Amazon Keyspaces により各行の追加の TTL 関連メタデータの保存が開始されます。さらに、TTL により有効期限タイムスタンプが使用されて、行または列の有効期限が切れる時期が追跡されます。タイムスタンプは、行メタデータとして保存されるため、行のストレージコストに関係します。

TTL 機能は、いったん有効化されると、テーブルに対して無効化することはできません。テーブルの `default_time_to_live` を 0 に設定すると、新しいデータのデフォルトの有効期限は無効になりますが、TTL 機能は非アクティブにならず、テーブルが元の Amazon Keyspaces ストレージメタデータや書き込み動作に戻ることもありません。

次の例では、デフォルトの TTL 値を設定した新規テーブルの作成方法を示しています。

Console

コンソールを使用してデフォルトの TTL 値を設定した新規テーブルを作成する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで [Tables] (テーブル) を選択して、[Create table (テーブルの作成)] を選択します。
3. [Table details (テーブルの詳細)] セクションの [Create table (テーブルの作成)] ページで、キースペースを選択し、新しいテーブルに名前を付けます。
4. [Schema (スキーマ)] セクションで、テーブルのスキーマを作成します。
5. [Table settings (テーブルの設定)] セクションで、[Customize settings (設定のカスタマイズ)] を選択します。
6. 有効期限 (TTL) に進みます。

   このステップでは、テーブルのデフォルトの TTL 設定を選択します。

   [Default TTL period] (デフォルト TTL 期間) で、有効期限を入力し、入力したの時間の単位 (秒、日、年など) を選択します。Amazon Keyspaces では、値が秒単位で保存されます。
7. [テーブルの作成] を選択します。テーブルは、指定したデフォルトの TTL 値で作成されます。

Cassandra Query Language (CQL)

CQL を使用してデフォルトの TTL 値を設定した新規テーブルを作成する

1. 次のステートメントでは、デフォルトの TTL 値が 3,024,000 秒 (35 日間) に設定された新規テーブルを作成します。

```
CREATE TABLE my_table (
                userid uuid,
                time timeuuid,
                subject text,
                body text,
                user inet,
                PRIMARY KEY (userid, time)
                ) WITH default_time_to_live = 3024000;
```

2. 新しいテーブルの TTL 設定を確認するには、次の例に示すように、cqlsh DESCRIBE ステートメントを使用します。出力では、テーブルのデフォルト TTL 設定が default_time_to_live として表示されます。

```
DESC TABLE my_table;
```

```
CREATE TABLE my_keyspace.my_table (
    userid uuid,
    time timeuuid,
    body text,
    subject text,
    user inet,
    PRIMARY KEY (userid, time)
) WITH CLUSTERING ORDER BY (time ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'class': 'com.amazonaws.cassandra.DefaultCaching'}
    AND comment = ''
    AND compaction = {'class': 'com.amazonaws.cassandra.DefaultCompaction'}
    AND compression = {'class': 'com.amazonaws.cassandra.DefaultCompression'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.0
    AND default_time_to_live = 3024000
    AND gc_grace_seconds = 7776000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 3600000
    AND min_index_interval = 128
```

```
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

CLI

を使用してデフォルトの TTL 値で新しいテーブルを作成する AWS CLI

1. 次のコマンドを使用して、デフォルトの TTL 値を 1 年に設定した新しいテーブルを作成できます。

```
aws keyspaces create-table --keyspace-name 'myKeyspace' --table-name 'myTable' \
            --schema-definition 'allColumns=[{name=id,type=int},
{name=name,type=text},{name=date,type=timestamp}],partitionKeys=[{name=id}]' \
            --default-time-to-live '31536000'
```

2. テーブルの TTL のステータスを確認するには、次のコマンドを使用します。

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

コマンドの出力は、次の例のようになります。

```
{
    "keyspaceName": "myKeyspace",
    "tableName": "myTable",
    "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/
myKeyspace/table/myTable",
    "creationTimestamp": "2024-09-02T10:52:22.190000+00:00",
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "id",
                "type": "int"
            },
            {
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
```

```
            }
        ],
        "partitionKeys": [
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2024-09-02T10:52:22.190000+00:00"
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "ttl": {
        "status": "ENABLED"
    },
    "defaultTimeToLive": 31536000,
    "comment": {
        "message": ""
    },
    "replicaSpecifications": []
}
```

テーブルのデフォルトの有効期間 (TTL) 値を更新する

既存のテーブルを新しいデフォルトの TTL 値で更新できます。TTL 値は秒単位で設定され、設定可能な最大値は 630,720,000 秒で、これは 20 年に相当します。

テーブルで TTL を有効にすると、Amazon Keyspaces により各行の追加の TTL 関連メタデータの保存が開始されます。さらに、TTL により有効期限タイムスタンプが使用されて、行または列の有効期限が切れる時期が追跡されます。タイムスタンプは、行メタデータとして保存されるため、行のストレージコストに関係します。

テーブルで TTL を有効にした後、CQL DML ステートメントを使用して、特定の行または列に対してテーブルのデフォルトの TTL 設定を上書きできます。詳細については、「the section called

“INSERT を使用した新しい行のカスタム TTL の設定”」および「the section called “UPDATE を使用した行と列のカスタム TTL 値の設定”」を参照してください。

TTL 機能は、いったん有効化されると、テーブルに対して無効化することはできません。テーブルの `default_time_to_live` を 0 に設定すると、新しいデータのデフォルトの有効期限は無効になりますが、TTL 機能は非アクティブにならず、テーブルが元の Amazon Keyspaces ストレージメタデータや書き込み動作に戻ることもありません。

以下の手順でコンソール、CQL、または AWS CLIを使用して、既存のテーブルのデフォルトの有効期限 (TTL) 設定を更新してください。

Console

コンソールを使用してテーブルのデフォルトの TTL 値を更新する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. 更新するテーブルを選択し、次に [Additional settings (追加設定)] タブを選択します。
3. [有効期限 (TTL)] に進み、[編集] を選択します。
4. [デフォルトの TTL 期間] で有効期限を入力し、時間の単位 (秒、日、年など) を選択します。Amazon Keyspaces では、値が秒単位で保存されます。これによって既存の行の TTL 値が変更されることはありません。
5. TTL 設定が定義されたら、[Save changes] (変更を保存) を選択します。

Cassandra Query Language (CQL)

CQL を使用してテーブルのデフォルトの TTL 値を更新する

1. `ALTER TABLE` を使用して、テーブルのデフォルトの有効期限 (TTL) 設定を編集できます。テーブルのデフォルトの TTL 設定を 2,592,000 秒 (30 日間) に更新するには、次のステートメントを使用します。

```
ALTER TABLE my_table WITH default_time_to_live = 2592000;
```

2. 更新されたテーブルの TTL 設定を確認するには、次の例に示すように、`cqlsh DESCRIBE` ステートメントを使用します。出力では、テーブルのデフォルト TTL 設定が `default_time_to_live` として表示されます。

```
DESC TABLE my_table;
```

このステートメントの出力は、次の例のようになります。

```
CREATE TABLE my_keyspace.my_table (
    id int PRIMARY KEY,
    date timestamp,
    name text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = {'class': 'com.amazonaws.cassandra.DefaultCaching'}
    AND comment = ''
    AND compaction = {'class': 'com.amazonaws.cassandra.DefaultCompaction'}
    AND compression = {'class': 'com.amazonaws.cassandra.DefaultCompression'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.0
    AND default_time_to_live = 2592000
    AND gc_grace_seconds = 7776000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 3600000
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

CLI

を使用してテーブルのデフォルトの TTL 値を更新する AWS CLI

1. update-table を使用して、テーブルのデフォルトの TTL 値を編集できます。テーブルのデフォルトの TTL 設定を 2,592,000 秒 (30 日間) に更新するには、次のステートメントを使用します。

```
aws keyspaces update-table --keyspace-name 'myKeyspace' --table-name 'myTable'
 --default-time-to-live '2592000'
```

2. 更新後のデフォルトの TTL 値を確認するには、次のステートメントを使用します。

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

ステートメントの出力は、次の例のようになります。

```
{
    "keyspaceName": "myKeyspace",
    "tableName": "myTable",
    "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/
myKeyspace/table/myTable",
    "creationTimestamp": "2024-09-02T10:52:22.190000+00:00",
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "id",
                "type": "int"
            },
            {
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2024-09-02T10:52:22.190000+00:00"
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "ttl": {
        "status": "ENABLED"
    },
```

```
    "defaultTimeToLive": 2592000,
    "comment": {
        "message": ""
    },
    "replicaSpecifications": []
}
```

カスタムの有効期限 (TTL) 設定を有効にしてテーブルを作成する

テーブル全体の有効期限 (TTL) のデフォルト設定を有効にせずに、行や列に適用可能なカスタムの TTL 設定を有効にして新しいテーブルを作成するには、次のステートメントを使用します。

> **Note**
>
> カスタムの `ttl` 設定を有効にしてテーブルを作成した場合、その設定を後から無効にすることはできません。

Cassandra Query Language (CQL)

CQL を使用してカスタム TTL 設定で新しいテーブルを作成する

- ```
  CREATE TABLE my_keyspace.my_table (id int primary key) WITH
   CUSTOM_PROPERTIES={'ttl':{'status': 'enabled'}};
  ```

CLI

を使用してカスタム TTL 設定で新しいテーブルを作成する AWS CLI

1. 次のコマンドを使用して、TTL を有効にした新しいテーブルを作成できます。

   ```
   aws keyspaces create-table --keyspace-name 'myKeyspace' --table-name 'myTable' \
                                   --schema-definition
    'allColumns=[{name=id,type=int},{name=name,type=text},
    {name=date,type=timestamp}],partitionKeys=[{name=id}]' \
                                   --ttl 'status=ENABLED'
   ```

2. テーブルで TTL が有効になっていることを確認するには、次のステートメントを使用します。
   ```

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

ステートメントの出力は、次の例のようになります。

```
{
    "keyspaceName": "myKeyspace",
    "tableName": "myTable",
    "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/
myKeyspace/table/myTable",
    "creationTimestamp": "2024-09-02T10:52:22.190000+00:00",
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
         {
                "name": "id",
                "type": "int"
            },
            {
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2024-09-02T11:18:55.796000+00:00"
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
```

```
        "status": "DISABLED"
    },
    "ttl": {
        "status": "ENABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    },
    "replicaSpecifications": []
}
```

カスタムの有効期限 (TTL) でテーブルを更新する

テーブル全体の有効期限 (TTL) のデフォルト値を設定せずに、テーブルでカスタムの TTL 設定を有効にして、TTL 値を個々の行や列に適用するには、次のコマンドを使用します。

> **Note**
>
> `ttl` を有効にした後、そのテーブルに対してそれを無効にすることはできません。

Cassandra Query Language (CQL)

CQL を使用してテーブルのカスタム TTL 設定を有効にする

- ```
  ALTER TABLE my_table WITH CUSTOM_PROPERTIES={'ttl':{'status': 'enabled'}};
  ```

CLI

を使用してテーブルのカスタム TTL 設定を有効にする AWS CLI

1. 次のコマンドを使用して、テーブルのカスタム TTL 設定を更新できます。

   ```
   aws keyspaces update-table --keyspace-name 'myKeyspace' --table-name 'myTable'
    --ttl 'status=ENABLED'
   ```

2. テーブルで TTL が有効になっているか確認するには、次のステートメントを使用します。
   ```

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

ステートメントの出力は、次の例のようになります。

```
{
    "keyspaceName": "myKeyspace",
    "tableName": "myTable",
    "resourceArn": "arn:aws:cassandra:us-east-1:123SAMPLE012:/keyspace/
myKeyspace/table/myTable",
    "creationTimestamp": "2024-09-02T11:32:27.349000+00:00",
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "id",
                "type": "int"
            },
            {
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2024-09-02T11:32:27.349000+00:00"
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
```

```
        "status": "DISABLED"
    },
    "ttl": {
        "status": "ENABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    },
    "replicaSpecifications": []
}
```

INSERT ステートメントを使用して新しい行のカスタムの有効期限 (TTL) 値を設定する

> **Note**
>
> INSERT ステートメントを使用して行のカスタム TTL 値を設定する前に、まず、テーブルでカスタム TTL を有効にする必要があります。詳細については、「the section called “テーブルのカスタム TTL の更新”」を参照してください。

個々の行の有効期限を設定してテーブルのデフォルトの TTL 値を上書きするには、INSERT ステートメントを使用できます。

- INSERT — TTL 値セットを含む新しいデータ行を挿入します。

INSERT ステートメントを使用して新しい行に TTL 値を設定すると、テーブルのデフォルトの TTL 設定よりも優先されます。

次の CQL ステートメントでは、テーブルにデータの行が挿入され、デフォルトの TTL 設定が 259,200 秒 (3 日間に相当) に変更されます。

```
INSERT INTO my_table (userid, time, subject, body, user)
        VALUES (B79CB3BA-745E-5D9A-8903-4A02327A7E09, 96a29100-5e25-11ec-90d7-
b5d91eceda0a, 'Message', 'Hello','205.212.123.123')
        USING TTL 259200;
```

挿入された行の TTL 設定を確認するには、次のステートメントを使用します。

```
SELECT TTL (subject) from my_table;
```

UPDATE ステートメントを使用して行と列のカスタム有効期限 (TTL) 設定を編集する

> **Note**
>
> 行と列のカスタム TTL 値を設定する前に、まず、テーブルで TTL を有効にしておく必要があります。詳細については、「the section called “テーブルのカスタム TTL の更新”」を参照してください。

UPDATE ステートメントを使用して、個々の行と列の有効期限を設定することで、テーブルのデフォルトの TTL 値を上書きできます。

- 行 – 既存のデータ行をカスタム TTL 値で更新できます。
- 列 – 既存の行内の列のサブセットをカスタム TTL 値で更新できます。

行の TTL 値を設定すると、テーブルのデフォルト TTL 設定よりも優先されます。

以前に挿入された「件名」列の TTL 設定を 259,200 秒 (3 日間) から 86,400 秒 (1 日間) に変更するには、次のステートメントを使用します。

```
UPDATE my_table USING TTL 86400 set subject = 'Updated Message' WHERE userid =
 B79CB3BA-745E-5D9A-8903-4A02327A7E09 and time = 96a29100-5e25-11ec-90d7-b5d91eceda0a;
```

簡単な選択クエリを実行すると、有効期限が切れる前に更新されたレコードを確認できます。

```
SELECT * from my_table;
```

クエリには次の出力が表示されます。

```
userid                                | time                                 | body   |
 subject         | user
--------------------------------------+--------------------------------------+-------
+-----------------+-----------------
```

```
b79cb3ba-745e-5d9a-8903-4a02327a7e09  | 96a29100-5e25-11ec-90d7-b5d91eceda0a | Hello |
 Updated Message | 205.212.123.123
50554d6e-29bb-11e5-b345-feff819cdc9f  | cf03fb21-59b5-11ec-b371-dff626ab9620 | Hello |
        Message | 205.212.123.123
```

有効期限が正常に終了したことを確認するには、設定した有効期限が切れた後に同じクエリをもう一度実行します。

```
SELECT * from my_table;
```

このクエリでは、「件名」列の有効期限が切れた後に次の出力が表示されます。

```
userid                                | time                                 | body  |
 subject | user
--------------------------------------+--------------------------------------+-------
+---------+-----------------
b79cb3ba-745e-5d9a-8903-4a02327a7e09  | 96a29100-5e25-11ec-90d7-b5d91eceda0a | Hello |
   null | 205.212.123.123
50554d6e-29bb-11e5-b345-feff819cdc9f  | cf03fb21-59b5-11ec-b371-dff626ab9620 | Hello |
 Message | 205.212.123.123
```

AWS SDK でのこのサービスの使用

AWS Software Development Kit (SDKsは、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようにする API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コード例
AWS SDK for C++	AWS SDK for C++ コード例
AWS CLI	AWS CLI コード例
AWS SDK for Go	AWS SDK for Go コード例
AWS SDK for Java	AWS SDK for Java コード例
AWS SDK for JavaScript	AWS SDK for JavaScript コード例
AWS SDK for Kotlin	AWS SDK for Kotlin コード例

SDK ドキュメント	コード例
AWS SDK for .NET	AWS SDK for .NET コード例
AWS SDK for PHP	AWS SDK for PHP コード例
AWS Tools for PowerShell	AWS Tools for PowerShell コード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例
AWS SDK for Ruby	AWS SDK for Ruby コード例
AWS SDK for Rust	AWS SDK for Rust コード例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例
AWS SDK for Swift	AWS SDK for Swift コード例

> **可用性の例**
>
> 必要なものが見つからなかった場合。このページの下側にある [Provide feedback (フィードバックを送信)] リンクから、コードの例をリクエストしてください。

Amazon Keyspaces リソースのタグとラベルを操作する

タグを使用して Amazon Keyspaces (Apache Cassandra 向け) リソースにラベルを付けることができます。タグを使用すると、目的別、所有者別、環境別など、さまざまな方法でリソースを分類できます。タグは、以下のことに役立ちます。

- リソースに割り当てたタグに基づいてリソースをすばやく特定します。
- タグ別に分類された AWS 請求書を参照してください。
- タグに基づいて Amazon Keyspaces リソースへのアクセスを制御します。タグを使用する IAM ポリシーの例については、「the section called “Amazon Keyspaces タグに基づいた認可”」を参照してください。

タグ付けは、Amazon Elastic Compute Cloud (Amazon EC2)、Amazon Simple Storage Service (Amazon S3)、Amazon Keyspaces などの AWS サービスでサポートされています。効率的なタグ付

けを行うと、特定のタグを持つサービス間でレポートを作成でき、コストインサイトを得ることができます。

タグ付けの使用を開始するには、次のことを行います。

1. タグを使用して Amazon Keyspaces リソースにラベルを付ける場合の制限事項 について理解します。
2. Amazon Keyspaces のキースペース、テーブル、ストリームにタグを付ける を使用してタグを作成します。
3. を使用してタグを使用して Amazon Keyspaces のコスト配分レポートを作成する、アクティブなタグあたりの AWS コストを追跡します。

最後に、最適のタグ付け手法に従うことをお勧めします。詳細については、「AWS tagging strategies (タグ付け戦略)」を参照してください。

タグを使用して Amazon Keyspaces リソースにラベルを付ける場合の制限事項

タグはそれぞれ、1 つのキーと 1 つの値で構成されており、どちらもお客様側が定義します。以下の制限が適用されます。

- 各 Amazon Keyspaces キースペース、テーブル、またはストリームには、同じキーを持つタグを 1 つだけ含めることができます。既存のタグ (同じキー) を追加しようとすると、既存のタグの値は新しい値に更新されます。
- キースペースに適用されたタグは、そのキースペース内のテーブルに自動的に適用されません。キースペースとそのすべてのテーブルに同じタグを適用するには、各リソースに個別にタグを付ける必要があります。
- テーブルに適用されたタグは、そのテーブルのストリームに自動的に適用されません。テーブルの作成時にテーブルとストリームに同じタグを適用するには、テーブルの作成時に `PropagateTagsOnEnable`フラグを使用できます。このフラグを使用すると、Amazon Keyspaces はストリームの作成中にテーブルのタグをストリームに適用します。ストリームがアクティブな場合、テーブルタグへの変更はストリームには適用されません。
- マルチリージョンのキースペースまたはテーブルを作成すると、作成プロセス中に定義したタグは、すべてのリージョンのすべてのキースペースとテーブルに自動的に適用されます。`ALTER KEYSPACE` または `ALTER TABLE` を使用して既存のタグを変更すると、その更新は変更を行うリージョンのキースペースまたはテーブルにのみ適用されます。

- 値はタグカテゴリ (キー) の記述子として機能します。Amazon Keyspaces では、値を空または null にすることはできません
- タグのキーと値では、大文字と小文字が区別されます。
- キーの最大長は Unicode 文字で 128 文字です。
- 値の最大長は Unicode 文字で 256 文字です。
- 使用できる文字は、文字、ホワイトスペース、数字、および特殊文字 (+ - = . _ : /) です。
- リソースあたりのタグの最大数は 50 です。
- AWSによって割り当てられたタグ名と値には、割り当てることができないaws:プレフィックスが自動的に割り当てられます。 AWSによって割り当てられたタグ名は、タグ制限の 50 にはカウントされません。ユーザー側で割り当てたタグ名は、user: というプレフィックスを付けてコスト配分レポートに表示されます。
- 過去にさかのぼってタグを適用することはできません。

Amazon Keyspaces のキースペース、テーブル、ストリームにタグを付ける

Amazon Keyspaces コンソール、、または Cassandra クエリ言語 (CQL) を使用して、キースペース、テーブル、ストリームのタグを追加、一覧表示、編集 AWS CLI、または削除できます。その後、これらのユーザー定義タグをアクティブ化して、コスト配分の追跡のために AWS Billing and Cost Management コンソールに表示することができます。詳細については、「タグを使用して Amazon Keyspaces のコスト配分レポートを作成する」を参照してください。

一括編集の場合は、コンソールのタグエディタを使用することもできます。詳細については、「AWS Resource Groups ユーザーガイド」の「Working with Tag Editor」(タグエディタの使用) を参照してください。

タグ構造の詳細については、「タグを使用して Amazon Keyspaces リソースにラベルを付ける場合の制限事項」を参照してください。

トピック

- 新しいキースペースの作成時にタグを追加する
- キースペースにタグを追加する
- キースペースからタグを削除する
- キースペースのタグを表示する

- 新しいテーブルの作成時にタグを追加する
- テーブルにタグを追加する
- テーブルからタグを削除する
- テーブルのタグを表示する
- テーブルの作成時に新しいストリームにタグを追加する
- 既存のテーブルの新しいストリームにタグを追加する
- ストリームに新しいタグを追加する
- ストリームからタグを削除する
- ストリームのタグを表示する

新しいキースペースの作成時にタグを追加する

Amazon Keyspaces コンソール、CQL、または を使用して AWS CLI 、新しいキースペースを作成するときにタグを追加できます。

Console

コンソールを使用して新しいキースペースの作成時にタグを設定する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで [Keyspaces (キー空間)] を選択し、次に [Create keyspace (キー空間の作成)] を選択します。
3. [Create keyspace (キー空間の作成)] ページで、キー空間に名前を付けます。
4. [タグ] で [新しいタグの追加] を選択し、キーと値を入力します。
5. [Create keyspace (キースペースの作成)] を選択します。

Cassandra Query Language (CQL)

CQL を使用して新しいキースペースの作成時にタグを設定する

- 次の例では、タグ付きの新規キースペースを作成しています。

```
CREATE KEYSPACE mykeyspace WITH TAGS = {'key1':'val1', 'key2':'val2'};
```

CLI

を使用して新しいキースペースを作成するときにタグを設定する AWS CLI

- 次のステートメントでは、タグ付きの新しいキースペースを作成します。

```
aws keyspaces create-keyspace --keyspace-name 'myKeyspace' --tags
 'key=key1,value=val1' 'key=key2,value=val2'
```

キースペースにタグを追加する

次の例では、Amazon Keyspaces でキースペースにタグを追加する方法を示しています。

Console

コンソールを使用して既存のキースペースにタグを追加する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Keyspaces (キー空間)] を選択します。
3. リストからキースペースを選択します。次に、[タグ] を選択します。キースペースのタグが表示されます。
4. [タグの管理] を選択して、タグを追加、編集、または削除します。
5. [Save changes] (変更の保存) をクリックします。

Cassandra Query Language (CQL)

CQL を使用して既存のキースペースにタグを追加する

-

```
ALTER KEYSPACE mykeyspace ADD TAGS {'key1':'val1', 'key2':'val2'};
```

CLI

を使用して既存のキースペースにタグを追加する AWS CLI

- 次の例では、既存のキースペースに新しいタグを追加する方法を示しています。

```
aws keyspaces tag-resource --resource-arn 'arn:aws:cassandra:us-
east-1:111222333444:/keyspace/myKeyspace/' --tags 'key=key3,value=val3'
 'key=key4,value=val4'
```

キースペースからタグを削除する

Console

コンソールを使用して既存のキースペースからタグを削除する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Keyspaces (キー空間)] を選択します。
3. リストからキースペースを選択します。次に、[タグ] を選択します。キースペースのタグが表示されます。
4. [タグの管理] を選択し、不要になったタグを削除します。
5. [Save changes] (変更の保存) をクリックします。

Cassandra Query Language (CQL)

CQL を使用して既存のキースペースからタグを削除する

- ```
  ALTER KEYSPACE mykeyspace DROP TAGS {'key1':'val1', 'key2':'val2'};
  ```

CLI

を使用して既存のキースペースからタグを削除する AWS CLI

- 次のステートメントは、指定されたタグをキースペースから削除します。

  ```
  aws keyspaces untag-resource --resource-arn 'arn:aws:cassandra:us-
  east-1:111222333444:/keyspace/myKeyspace/' --tags 'key=key3,value=val3'
   'key=key4,value=val4'
  ```
  ```

キースペースのタグを表示する

次の例では、コンソール、CQL、または AWS CLIを使用して、タグを読み取る方法を示しています。

Console

Amazon Keyspaces コンソールを使用してキースペースのタグを表示する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Keyspaces (キー空間)] を選択します。
3. リストからキースペースを選択します。次に、[タグ] を選択します。キースペースのタグが表示されます。

Cassandra Query Language (CQL)

CQL を使用してキースペースのタグを表示する

キースペースに追加されているタグを読み取るには、次の CQL ステートメントを使用します。

```
SELECT * FROM system_schema_mcs.tags WHERE valid_where_clause;
```

WHERE 句は必須です。次のいずれかの形式で指定する必要があります。

- keyspace_name = **'mykeyspace'** AND resource_type = 'keyspace'
- resource_id = *arn*

- 次のステートメントは、キースペースのタグの有無を示しています。

```
SELECT * FROM system_schema_mcs.tags WHERE keyspace_name = 'mykeyspace' AND
 resource_type = 'keyspace';
```

クエリの出力は以下のようになります。

```

resource_id                                                    | keyspace_name
 | resource_name | resource_type | tags
```

```
------------------------------------------------------------------
+---------------+---------------+---------------+------
arn:aws:cassandra:us-east-1:123456789:/keyspace/mykeyspace/      | mykeyspace
 | mykeyspace    | keyspace      | {'key1': 'val1', 'key2': 'val2'}
```

CLI

を使用してキースペースのタグを表示する AWS CLI

- この例では、指定されたリソースのタグを一覧表示する方法を示しています。

```
aws keyspaces list-tags-for-resource --resource-arn 'arn:aws:cassandra:us-
east-1:111222333444:/keyspace/myKeyspace/'
```

最後のコマンドの出力は次のようになります。

```
{
    "tags": [
        {
            "key": "key1",
            "value": "val1"
        },
        {
            "key": "key2",
            "value": "val2"
        },
        {
            "key": "key3",
            "value": "val3"
        },
        {
            "key": "key4",
            "value": "val4"
        }
    ]
}
```

新しいテーブルの作成時にタグを追加する

Amazon Keyspaces コンソール、CQL、または を使用して AWS CLI 、新しいテーブルの作成時にタグを追加できます。

Console

コンソールを使用して新しいテーブルの作成時にタグを追加する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで [Tables (テーブル)] を選択して、[Create table (テーブルの作成)] を選択します。
3. [Table details (テーブルの詳細)] セクションの [Create table (テーブルの作成)] ページで、キー空間を選択し、テーブルに名前を付けます。
4. [Schema (スキーマ)] セクションで、テーブルのスキーマを作成します。
5. [Table settings (テーブルの設定)] セクションで、[Customize settings (設定のカスタマイズ)] を選択します。
6. [Table tags – optional (テーブルタグ - オプション)] セクションに進み、[Add new tag (新しいタグの追加)] を選択して、新しいタグを作成します。
7. [Create table (テーブルの作成)] を選択します。

Cassandra Query Language (CQL)

CQL を使用して新しいテーブルの作成時にタグを追加する

- 次の例では、タグ付きの新規テーブルを作成しています。

```
CREATE TABLE mytable(...) WITH TAGS = {'key1':'val1', 'key2':'val2'};
```

CLI

を使用して新しいテーブルを作成するときにタグを追加する AWS CLI

- 次の例では、タグ付きの新規テーブルを作成する方法を示しています。このコマンドは、既存のキースペース myKeyspace に myTable というテーブルを作成します。コマンドは読みやすくするために複数の行に分割されていることに注意してください。

```
aws keyspaces create-table --keyspace-name 'myKeyspace' --table-name 'myTable'
            --schema-definition 'allColumns=[{name=id,type=int},
{name=name,type=text},{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --tags 'key=key1,value=val1' 'key=key2,value=val2'
```

テーブルにタグを追加する

コンソール、CQL、または AWS CLIを使用して、Amazon Keyspaces で既存のテーブルにタグを追加できます。

Console

Amazon Keyspaces コンソールを使用してテーブルにタグを追加する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Tables (テーブル)] を選択します。
3. リストからテーブルを選択し、[タグ] タブを選択します。
4. [タグの管理] を選択して、テーブルにタグを追加します。
5. [Save changes] (変更の保存) をクリックします。

Cassandra Query Language (CQL)

CQL を使用してテーブルにタグを追加する

- 次のステートメントは、既存のテーブルに列タグを追加する方法を示しています。

```
ALTER TABLE mykeyspace.mytable ADD TAGS {'key1':'val1', 'key2':'val2'};
```

CLI

を使用してテーブルにタグを追加する AWS CLI

- 次の例では、既存のテーブル に 5 つの新しいタグを追加する方法を示しています。

```
aws keyspaces tag-resource --resource-arn 'arn:aws:cassandra:us-
east-1:111222333444:/keyspace/myKeyspace/table/myTable' --tags
 'key=key3,value=val3' 'key=key4,value=val4'
```

テーブルからタグを削除する

Console

Amazon Keyspaces コンソールを使用してテーブルからタグを削除する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Tables (テーブル)] を選択します。
3. リストからテーブルを選択し、[タグ] タブを選択します。
4. [タグの管理] を選択して、テーブルからタグを削除します。
5. [Save changes] (変更の保存) をクリックします。

Cassandra Query Language (CQL)

CQL を使用してテーブルからタグを削除する

- 次のステートメントは、既存のテーブルからタグを削除する方法を示しています。

```
ALTER TABLE mytable DROP TAGS {'key3':'val3', 'key4':'val4'};
```

CLI

を使用してテーブルにタグを追加する AWS CLI

- 次のステートメントは、指定されたタグをキースペースから削除します。

```
aws keyspaces untag-resource --resource-arn 'arn:aws:cassandra:us-
east-1:111222333444:/keyspace/myKeyspace/table/myTable' --tags
 'key=key3,value=val3' 'key=key4,value=val4'
```

テーブルのタグを表示する

次の例は、コンソール、CQL、または を使用して Amazon Keyspaces でテーブルのタグを表示する方法を示しています AWS CLI。

Console

コンソールを使用してテーブルのタグを表示する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Tables (テーブル)] を選択します。
3. リストからテーブルを選択し、[タグ] タブを選択します。

Cassandra Query Language (CQL)

CQL を使用してテーブルのタグを表示する

テーブルに追加されているタグを読み取るには、次の CQL ステートメントを使用します。

```
SELECT * FROM system_schema_mcs.tags WHERE valid_where_clause;
```

WHERE 句は必須です。次のいずれかの形式で指定する必要があります。

- keyspace_name = **'mykeyspace'** AND resource_name = **'mytable'**
- resource_id = *arn*

- 次のクエリは、指定されたテーブルのタグを返します。

```
SELECT * FROM system_schema_mcs.tags WHERE keyspace_name = 'mykeyspace' AND
 resource_name = 'mytable';
```

このクエリの出力は次のようになります。

```
resource_id                                                                 |
 keyspace_name | resource_name | resource_type | tags
```

```
--------------------------------------------------------------------------------
+---------------+---------------+---------------+------
arn:aws:cassandra:us-east-1:123456789:/keyspace/mykeyspace/table/mytable|
 mykeyspace    | mytable       | table         | {'key1': 'val1', 'key2':
 'val2'}
```

CLI

を使用してテーブルのタグを表示する AWS CLI

- この例では、指定されたリソースのタグを一覧表示する方法を示しています。

```
aws keyspaces list-tags-for-resource --resource-arn
 'arn:aws:arn:aws:cassandra:aws-region:111122223333:/keyspace/my_keyspace/
table/my_table/stream/2025-05-11T21:21:33.291cassandra:us-east-1:111222333444:/
keyspace/myKeyspace/table/myTable'
```

最後のコマンドの出力は次のようになります。

```
{
    "tags": [
        {
            "key": "key1",
            "value": "val1"
        },
        {
            "key": "key2",
            "value": "val2"
        },
        {
            "key": "key3",
            "value": "val3"
        },
        {
            "key": "key4",
            "value": "val4"
        }
    ]
}
```

テーブルの作成時に新しいストリームにタグを追加する

CQL または を使用してストリームで新しいテーブルを作成するときに、ストリーム AWS CLI にタグを追加できます。

> **Note**
>
> Amazon Keyspaces CDC には、ユーザーに代わって Amazon Keyspaces CDC ストリームから CloudWatch アカウントの `"cloudwatch:namespace": "AWS/Cassandra"` にメトリクスデータを発行するサービスにリンクされたロール (AWSServiceRoleForAmazonKeyspacesCDC) が必要です。このロールは自動的に作成されます。詳細については、「the section called “CDC ストリーム”」を参照してください。

Cassandra Query Language (CQL)

CQL を使用して新しいテーブルを作成するときにストリームにタグを追加する

1. ストリームを使用して新しいテーブルを作成し、そのテーブルタグをストリームに自動的に適用するには、`'propagate_tags': 'TABLE'`フラグを使用できます。以下のステートメントは、この例です。

```
CREATE TABLE mytable (pk int, ck text, PRIMARY KEY(pk))
WITH TAGS={'key1':'val1', 'key2':'val2'}
AND cdc = TRUE
AND CUSTOM_PROPERTIES={
    'cdc_specification': {
        'view_type': 'NEW_IMAGE',
        'propagate_tags': 'TABLE'
    }
};
```

2. ストリームに新しいタグを適用するには、次の例を使用します。

```
CREATE TABLE mytable (pk int, ck text, PRIMARY KEY(pk))
WITH TAGS={'key1':'val1', 'key2':'val2'}
AND cdc = TRUE
AND CUSTOM_PROPERTIES={
    'cdc_specification': {
        'view_type': 'NEW_IMAGE',
        'tags': { 'key': 'string', 'value': 'string' },
```

```
    }
};
```

CLI

を使用して新しいテーブルを作成するときにストリームにタグを追加する AWS CLI

1. ストリームを使用してテーブルを作成し、テーブルタグをストリームに自動的に適用するには、 propagateTags=Tableフラグを使用できます。次のコードは、この例です。

```
aws keyspaces create-table \
--keyspace-name 'my_keyspace' \
--table-name 'my_table' \
--schema-definition 'allColumns=[{name=pk,type=int},
{name=ck,type=text}],clusteringKeys=[{name=ck,orderBy=ASC}],partitionKeys=[{name=pk}]'
 \
--tags key=tag_key,value=tag_value
--cdc-specification propagateTags=TABLE,status=ENABLED,viewType=NEW_IMAGE
```

2. ストリームに異なるタグを適用するには、次の例を使用します。

```
aws keyspaces create-table \
--keyspace-name 'my_keyspace' \
--table-name 'my_table' \
--schema-definition 'allColumns=[{name=pk,type=int},
{name=ck,type=text}],clusteringKeys=[{name=ck,orderBy=ASC}],partitionKeys=[{name=pk}]'
 \
--tags key=tag_key,value=tag_value
--cdc-specification 'status=ENABLED,viewType=NEW_IMAGE,tags=[{key=tag_key,
 value=tag_value}]'
```

既存のテーブルの新しいストリームにタグを追加する

既存のテーブルの新しいストリームを作成するときにタグを追加できます。PropagateTags フラグを使用して、テーブルタグをストリームに適用するか、ストリームの新しいタグを指定できます。CQL または を使用して AWS CLI 、新しいストリームにタグを付けることができます。

> **Note**
>
> Amazon Keyspaces CDC には、ユーザーに代わって Amazon Keyspaces CDC ストリームから CloudWatch アカウントの `"cloudwatch:namespace": "AWS/Cassandra"` にメトリクスデータを発行するサービスにリンクされたロール (`AWSServiceRoleForAmazonKeyspacesCDC`) が必要です。このロールは自動的に作成されます。詳細については、「the section called “CDC ストリーム”」を参照してください。

Console

を使用して新しいストリームを作成するときにタグを追加する (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、テーブルを選択し、ストリームを追加するテーブルを選択します。
3. ストリームタブを選択します。
4. ストリームの詳細セクションで、編集 を選択します。
5. ストリームをオンにする を選択します。
6. 表示タイプを選択し、タグに進み、ストリームのタグを作成します。
7. 次のいずれかのオプションを選択できます。
   - タグなし – ストリームのタグを作成しない場合は、このオプションを使用します。
   - テーブルからタグをコピーする – テーブルからストリームにタグをコピーする場合は、このオプションを使用します。タグをコピーしたら、ストリームのタグを編集できます。このオプションは、テーブルにタグがある場合にのみ使用できることに注意してください。
   - 新しいタグの追加 – 新しいタグの追加を選択すると、ストリームに最大 50 個のタグを追加できます。
8. [Save changes] (変更の保存) をクリックします。

Cassandra Query Language (CQL)

新しいストリームを作成するときにタグを追加する

1. 既存のテーブルの新しいストリームを作成し、そのストリームにテーブルのタグを適用するには、 'propagate_tags': 'TABLE'フラグを使用できます。次のステートメントはその一例です。

```
ALTER TABLE mytable WITH cdc = TRUE AND CUSTOM_PROPERTIES={ 'cdc_specification':
 { 'view_type': 'NEW_IMAGE', 'propagate_tags': 'TABLE' } };
```

2. 既存のテーブルの新しいストリームを作成し、新しいタグを指定するには、次の例を使用できます。

```
ALTER TABLE mytable WITH cdc = TRUE AND CUSTOM_PROPERTIES={ 'cdc_specification':
 { 'view_type': 'NEW_IMAGE', 'tags': { 'key': 'string', 'value': 'string' }} };
```

CLI

を使用して新しいストリームを作成するときにタグを追加する AWS CLI

1. タグを使用して新しいストリームを作成するには、 propagateTags=TABLEフラグを使用してテーブルのタグをストリームに自動的に適用できます。次のコードは、この例です。

```
aws keyspaces update-table \
--keyspace-name 'my_keyspace' \
--table-name 'my_table' \
--cdc-specification propagateTags=TABLE,status=ENABLED,viewType=NEW_IMAGE
```

2. 既存のテーブルの新しいストリームを作成し、新しいタグを指定するには、次の例を使用できます。

```
aws keyspaces update-table \
--keyspace-name 'my_keyspace' \
--table-name 'my_table' \
--cdc-specification 'status=ENABLED,viewType=NEW_IMAGE,tags=[{key=tag_key,
 value=tag_value}]'
```

ストリームに新しいタグを追加する

CQL または を使用して、Amazon Keyspaces の既存のストリームに新しいタグを追加できます AWS CLI。タグは最新のストリームにのみ追加できます。

Console

既存のストリームにタグを追加する (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、テーブルを選択し、タグ付けするストリームを含むテーブルを選択します。
3. ストリームタブを選択します。
4. [タグ] セクションで、[タグを管理] を選択します。
5. 新しいタグを追加 を選択して、新しいタグを追加します。このステップを繰り返すことで、最大 50 個のタグを作成できます。
6. [Save changes] (変更の保存) をクリックします。

Cassandra Query Language (CQL)

CQL を使用してストリームにタグを追加する

- 次のステートメントは、既存のストリームにタグを追加する方法を示しています。

```
ALTER TABLE mykeyspace.mytable ADD TAGS_FOR_CDC {'key1':'val1', 'key2':'val2'};
```

CLI

を使用してストリームにタグを追加する AWS CLI

- 次の例は、既存のストリームに新しいタグを追加する方法を示しています。

```
aws keyspaces tag-resource --resource-arn 'arn:aws:cassandra:aws-
region:111122223333:/keyspace/my_keyspace/table/my_table/
stream/2025-05-11T21:21:33.291' --tags 'key=key3,value=val3'
 'key=key4,value=val4'
```

ストリームからタグを削除する

ストリームからタグを削除するには、CQL または を使用できます AWS CLI。最新のストリームのタグのみを削除できます。

Console

Amazon Keyspaces コンソールを使用してテーブルからタグを削除する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Tables (テーブル)] を選択します。
3. リストからテーブルを選択し、ストリームタブを選択します。
4. タグセクションでタグの管理を選択して、テーブルからタグを削除します。
5. 削除するタグの後に、削除を選択します。
6. [Save changes] (変更の保存) をクリックします。

Cassandra Query Language (CQL)

CQL を使用してストリームからタグを削除する

- 次のステートメントは、既存のストリームからタグを削除する方法を示しています。

```
ALTER TABLE mytable DROP TAGS_FOR_CDC {'key3':'val3', 'key4':'val4'};
```

CLI

を使用してストリームからタグを削除する AWS CLI

- 次のステートメントは、ストリームから指定されたタグを削除します。

```
aws keyspaces untag-resource --resource-arn 'arn:aws:cassandra:aws-
region:111122223333:/keyspace/my_keyspace/table/my_table/
stream/2025-05-11T21:21:33.291' --tags 'key=key3,value=val3'
 'key=key4,value=val4'
```

ストリームのタグを表示する

次の例は、CQL または を使用して Amazon Keyspaces でストリームのタグを表示する方法を示しています AWS CLI。

Console

コンソールを使用してストリームのタグを表示する

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Tables (テーブル)] を選択します。
3. リストからテーブルを選択し、ストリームタブを選択します。
4. ストリームのタグは、「タグ」セクションで表示できます。

Cassandra Query Language (CQL)

CQL を使用してストリームのタグを表示する

ストリームにアタッチされたタグを読み取るには、 WHERE句でストリームのリソース ARN を指定する必要があります。次の CQL 構文がその例です。

```
SELECT * FROM system_schema_mcs.tags WHERE resource_id = stream_arn;
```

- 次のクエリは、指定されたストリームのタグを返します。

```
SELECT tags FROM system_schema_mcs.tags WHERE resource_id =
 'arn:aws:cassandra:aws-region:111122223333:/keyspace/my_keyspace/table/
my_table/stream/2025-05-06T17:17:39.800';
```

このクエリの出力は次のようになります。

```
 resource_id
                                    | keyspace_name | resource_name           |
 resource_type | tags
 ------------------------------------------------------------------------------------------
+---------------+-------------------------+--------------
+----------------------
```

```
 arn:aws:cassandra:aws-region:111122223333:/keyspace/my_keyspace/table/my_table/
stream/2025-04-02T23:00:07.052     |      singleks | 2025-04-02T23:00:07.052 |
      stream | {'tagkey': 'tagval'}
```

CLI

を使用してストリームのタグを表示する AWS CLI

- この例では、指定されたキースペース内のすべてのストリームのタグを一覧表示する方法を示します。

```
aws keyspaces list-tags-for-resource --resource-arn 'arn:aws:cassandra:aws-
region:111122223333:/keyspace/my_keyspace/table/my_table/
stream/2025-05-11T21:21:33.291'
```

最後のコマンドの出力は次のようになります。

```
{
    "tags": [
        {
            "key": "key1",
            "value": "val1"
        },
        {
            "key": "key2",
            "value": "val2"
        },
        {
            "key": "key3",
            "value": "val3"
        },
        {
            "key": "key4",
            "value": "val4"
        }
    ]
}
```

タグを使用して Amazon Keyspaces のコスト配分レポートを作成する

AWS はタグを使用してコスト配分レポートのリソースコストを整理します。 は 2 種類のコスト配分タグ AWS を提供します。

- AWSが生成した tag. は、このタグ AWS を定義、作成、適用します。
- ユーザー定義のタグ。これらのタグを定義、作成、適用します。

Cost Explorer またはコスト配分レポートで使用するには、事前に両方のタイプのタグを別々にアクティブ化しておく必要があります。

AWS生成されたタグをアクティブ化するには：

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/billing/home#/ で Billing and Cost Management コンソールを開きます。
2. ナビゲーションペインで、[Cost Allocation Tags] (コスト配分タグ) を選択します。
3. [AWS-Generated Cost Allocation Tags] で、[Activate] を選択します。

ユーザー定義タグをアクティブにするには:

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/billing/home#/ で Billing and Cost Management コンソールを開きます。
2. ナビゲーションペインで、[Cost Allocation Tags] (コスト配分タグ) を選択します。
3. [User-Defined Cost Allocation Tags] (ユーザー定義のコスト配分タグ) で、[Activate] (アクティブ化) を選択します。

タグを作成してアクティブ化すると、 は、使用量とコストをアクティブなタグ別にグループ化したコスト配分レポート AWS を生成します。コスト配分レポートには、各請求期間のすべての AWS コストが含まれます。タグ付きとタグなしのどちらのリソースもこのレポートに出力されるので、リソース別の請求額を明確に分類できます。

> **Note**
>
> 現時点では、Amazon Keyspaces から転送されたデータは、コスト配分レポートのタグによって分類されません。

詳細については、「Using cost allocation tags」(コスト配分タグの使用) を参照してください。

を使用して Amazon Keyspaces リソースを作成する AWS CloudFormation

Amazon Keyspaces は AWS CloudFormation、 AWS キースペースとテーブルのモデル化とセットアップに役立つサービスである と統合されているため、リソースとインフラストラクチャの作成と管理に費やす時間を短縮できます。必要なキースペースとテーブルを記述するテンプレートを作成し、 AWS CloudFormation がそれらのリソースのプロビジョニングと設定を行います。

を使用すると AWS CloudFormation、テンプレートを再利用して Amazon Keyspaces リソースを一貫して繰り返しセットアップできます。リソースを一度記述するだけで、複数の AWS アカウントおよびリージョンで同じリソースを何度もプロビジョニングできます。

Amazon Keyspaces と AWS CloudFormation テンプレート

Amazon Keyspaces のリソースをプロビジョニングして設定するには、AWS CloudFormation テンプレートについて理解しておく必要があります。テンプレートは、JSON や YAML でフォーマットされたテキストファイルです。これらのテンプレートは、 AWS CloudFormation スタックでプロビジョニングするリソースを記述します。JSON または YAML に慣れていない場合は、 AWS CloudFormation デザイナー を使用して AWS CloudFormation テンプレートの使用を開始できます。詳細については、『AWS CloudFormation ユーザーガイド』の「AWS CloudFormation Designer とは」を参照してください。

Amazon Keyspaces は、 AWS CloudFormationでのキースペースとテーブルの作成に対応しています。 AWS CloudFormation テンプレートを使用して作成するテーブルでは、スキーマ、読み取り/書き込みモード、プロビジョニングされたスループット設定、およびその他のサポートされている機能を指定できます。キースペースとテーブルの JSON テンプレートと YAML テンプレートの例を含む詳細については、「 テンプレートリファレンス」の「Amazon Keyspaces (Apache Cassandra 用) リソースタイプのAWS CloudFormation リファレンス」を参照してください。

の詳細 AWS CloudFormation

詳細については AWS CloudFormation、次のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

Amazon Keyspaces (Apache Cassandra 向け) での NoSQL Workbench の使用

NoSQL Workbench は、Amazon Keyspaces の非リレーショナルデータモデルの設計と可視化をより簡単にするクライアントサイドアプリケーションです。NoSQL Workbench クライアントは、Windows、macOS、Linux で使用できます。

データモデルの設計とリソースの自動作成

NoSQL Workbench にはポイントアンドクリックインターフェイスがあり、これを使用して Amazon Keyspaces データモデルの設計と作成を実行することができます。キースペース、テーブル、および列を定義することで、新しいデータモデルを最初から簡単に作成できます。また、既存のデータモデルをインポートし、変更 (列の追加、編集、削除など) を行い、新しいアプリケーションに合わせてデータモデルを調整することもできます。次に、NoSQL Workbench を使用すると、データモデルを Amazon Keyspaces または Apache Cassandra にコミットし、キースペースとテーブルを自動で作成することができます。データモデルの構築方法については、「the section called “データモデルの作成”」および「the section called “データモデルの編集”」を参照してください。

データモデルの可視化

NoSQL Workbench を使用すると、データモデルを可視化することで、アプリケーションのクエリやアクセスパターンのサポートを可能にします。また、コラボレーション用、ドキュメンテーション用、プレゼンテーション用などさまざまな形式でデータモデルの保存およびエクスポートを行うこともできます。詳細については、「the section called “データモデルの可視化”」を参照してください。

トピック

- NoSQL Workbench のダウンロード
- NoSQL Workbench の使用開始
- NoSQL Workbench でデータモデルを可視化する
- NoSQL Workbench で新しいデータモデルを作成する
- NoSQL Workbench で既存のデータモデルを編集する
- Amazon Keyspaces と Apache Cassandra にデータモデルをコミットする方法
- NoSQL Workbench のサンプルデータモデル
- NoSQL Workbench のリリース履歴

NoSQL Workbench のダウンロード

NoSQL Workbench をダウンロードしてインストールするには、以下の手順に従います。

NoSQL Workbench をダウンロードしてインストールするには

1. 以下のリンクのいずれかを使用して、NoSQL Workbench を無料でダウンロードします。

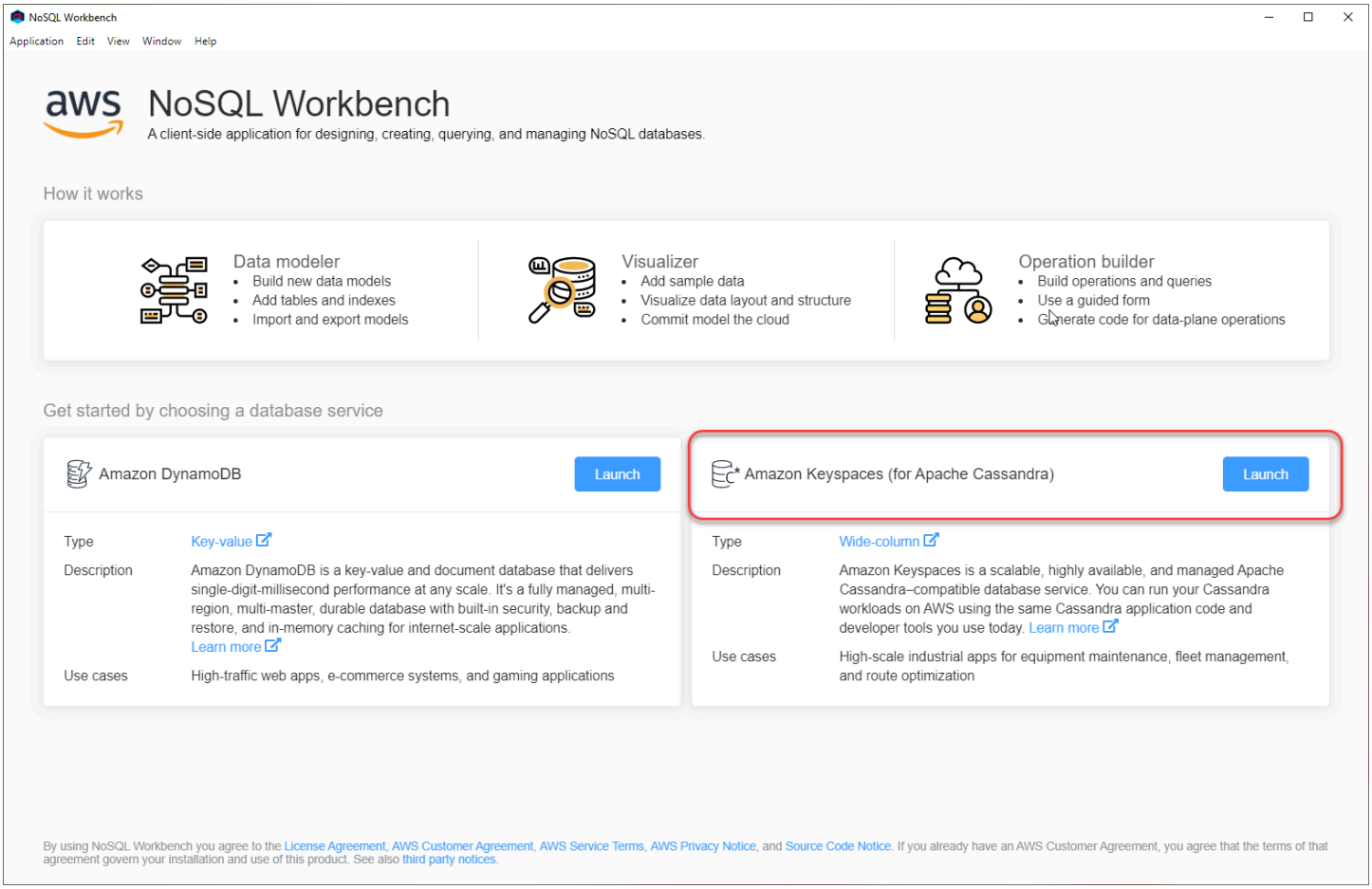
オペレーティングシステム	ダウンロードリンク
macOS	macOS 用のダウンロード
Linux*	Linux 用のダウンロード
Windows	Windows 用のダウンロード

   * NoSQL Workbench は、Ubuntu 12.04、Fedora 21、Debian 8、またはこれらの Linux ディストリビューションの新しいバージョンをサポートしています。

2. ダウンロードが完了したら、アプリケーションを起動し、画面の指示に従ってインストールを完了します。

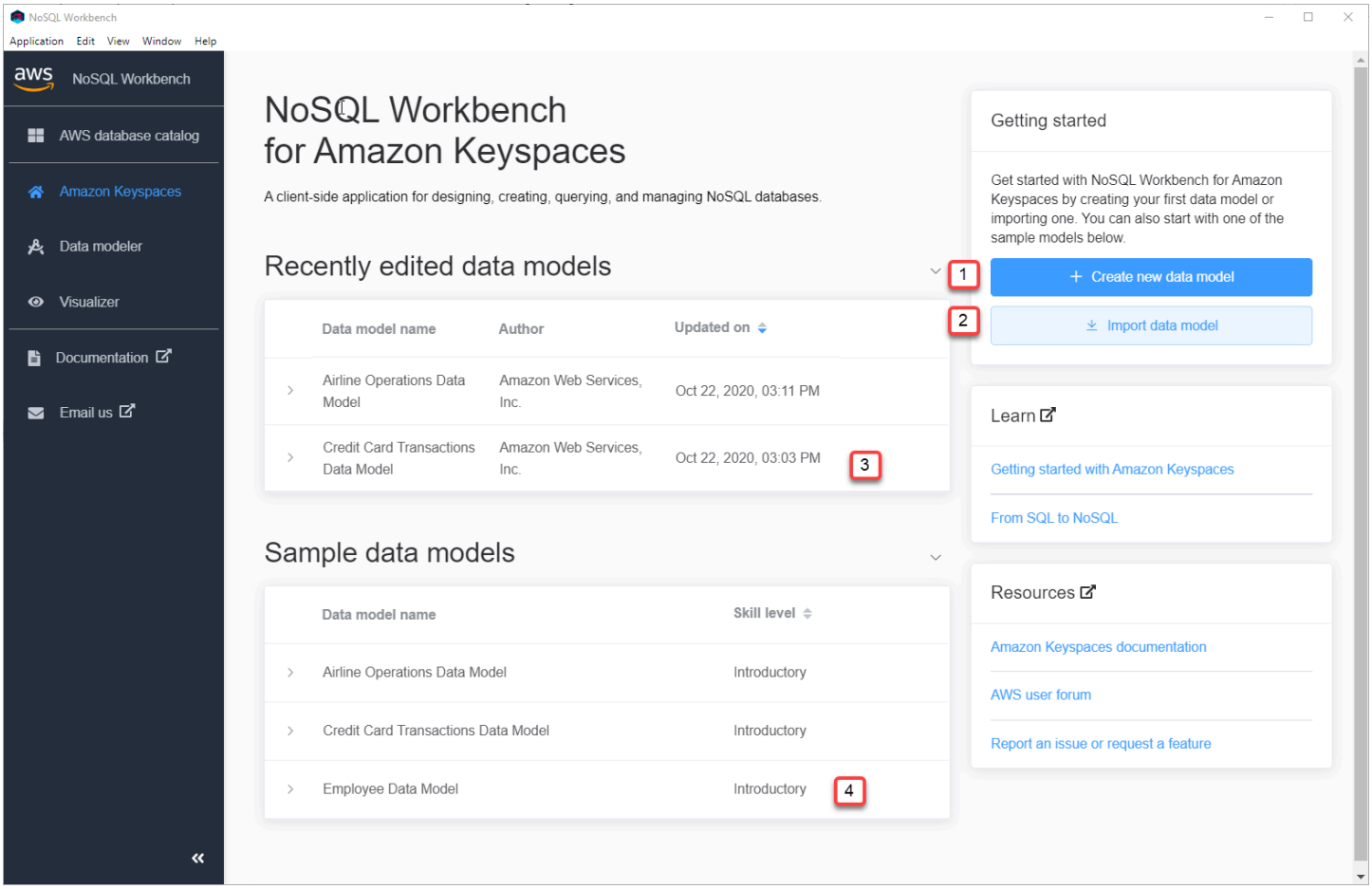
NoSQL Workbench の使用開始

NoSQL ワークベンチの使用を開始するには、NoSQL Workbench の [Database Catalog] (データベースカタログ) ページで、[Amazon Keyspaces] を選択し、[Launch] (起動) を選択します。



これにより Amazon Keyspaces の NoSQL Workbench のホームページが開くので、次のオプションを使用できます。

1. 新しいデータモデルを作成します。
2. 既存のデータモデルを JSON 形式でインポートします。
3. 最近編集したデータモデルを開きます。
4. 使用可能なサンプルモデルのいずれかを開きます。



各オプションで NoSQL Workbench データモデラーが開きます。新しいデータモデルの作成を続行するには、「the section called “データモデルの作成”」を参照してください。既存のデータモデルの編集については、「the section called “データモデルの編集”」を参照してください。

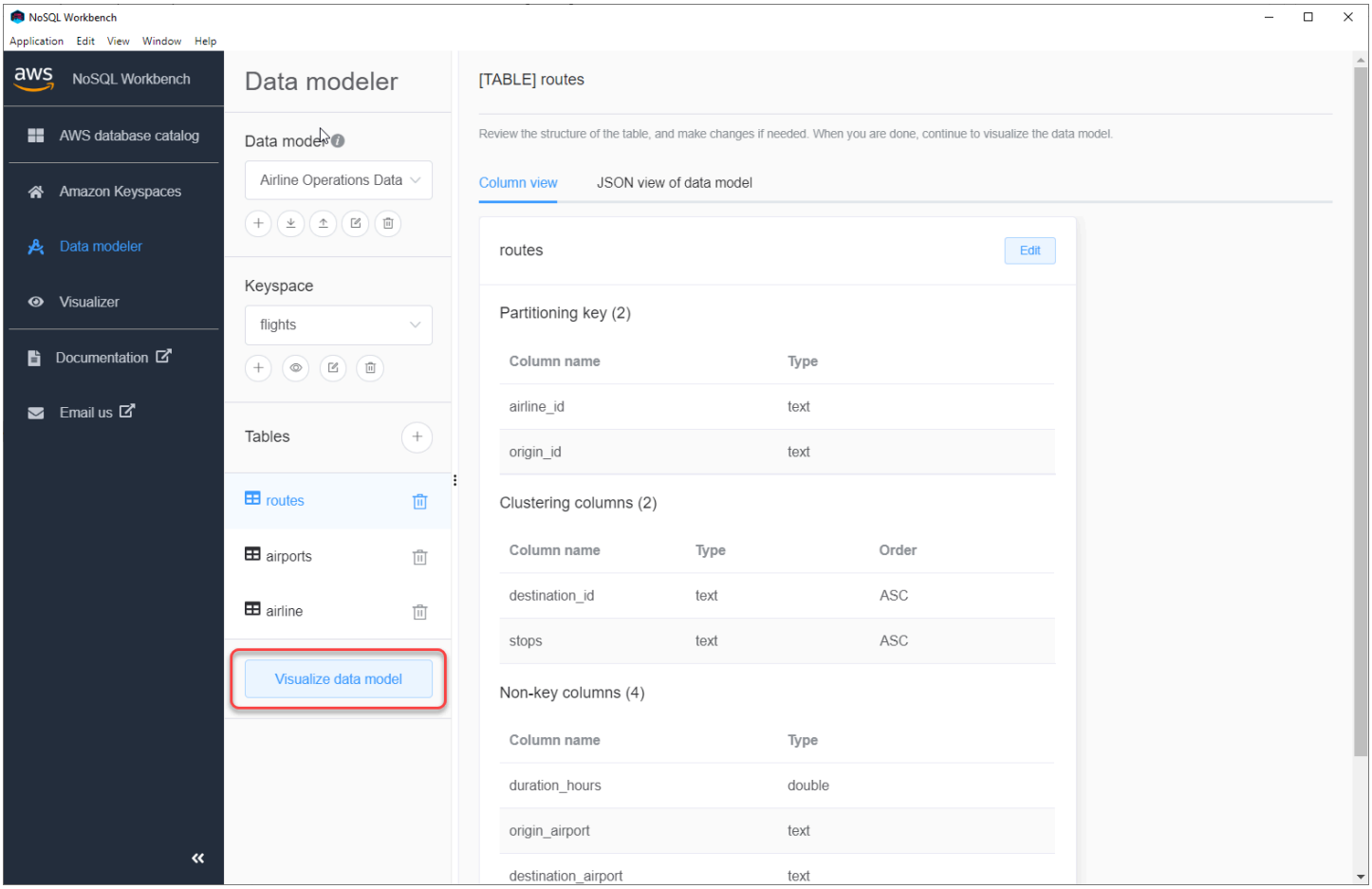
NoSQL Workbench でデータモデルを可視化する

NoSQL Workbench を使用すると、データモデルを可視化することで、アプリケーションのクエリやアクセスパターンのサポートを可能にします。また、コラボレーション用、ドキュメンテーション用、プレゼンテーション用などさまざまな形式でデータモデルの保存およびエクスポートを行うこともできます。

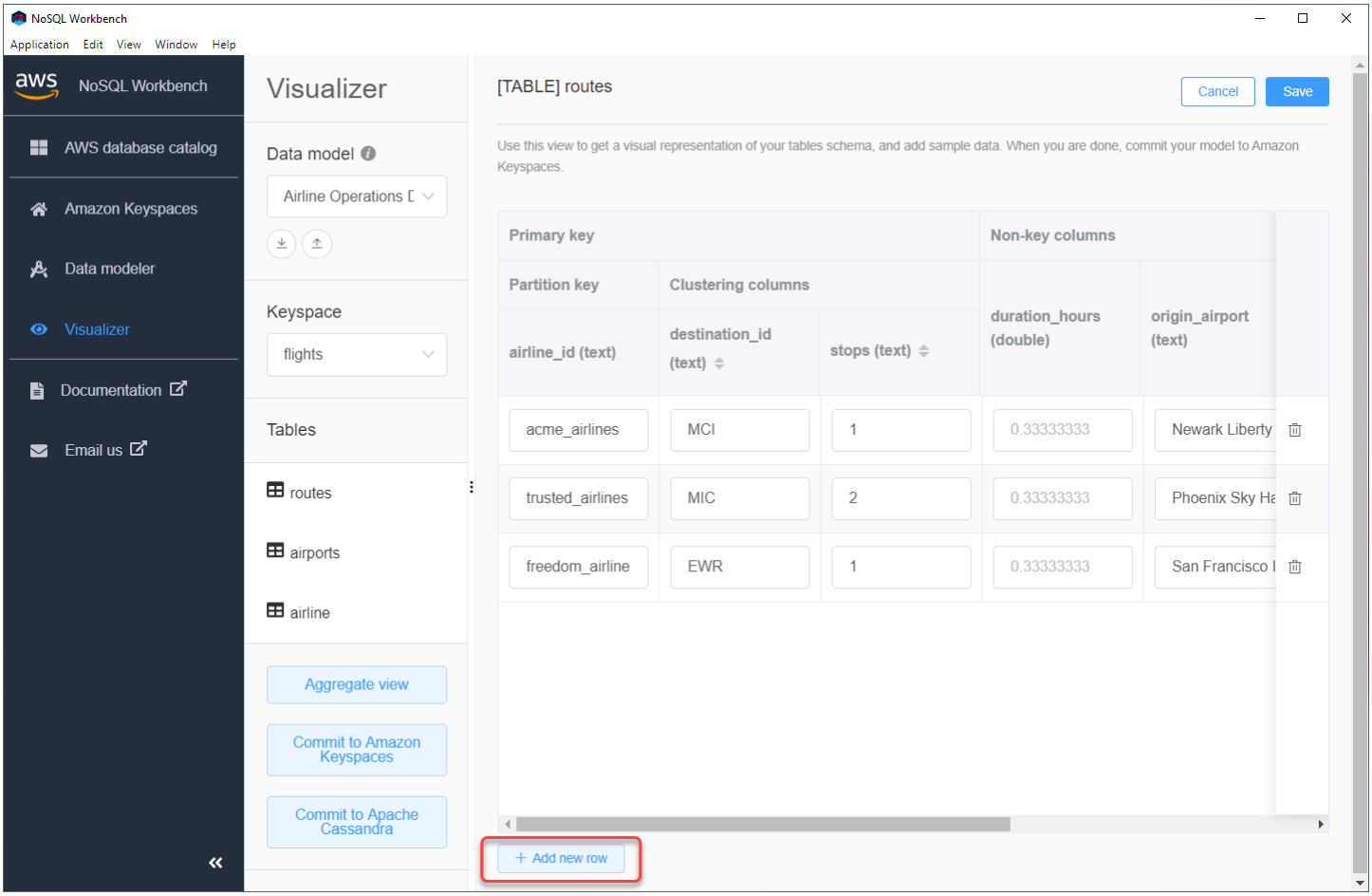
新しいデータモデルの作成後や、既存のデータモデルの編集後に、そのモデルを可視化できます。

NoSQL Workbench によるデータモデルの可視化

データモデラーでデータモデルを完成させたら、[Visualize data model] (データモデルの可視化) を選択します。

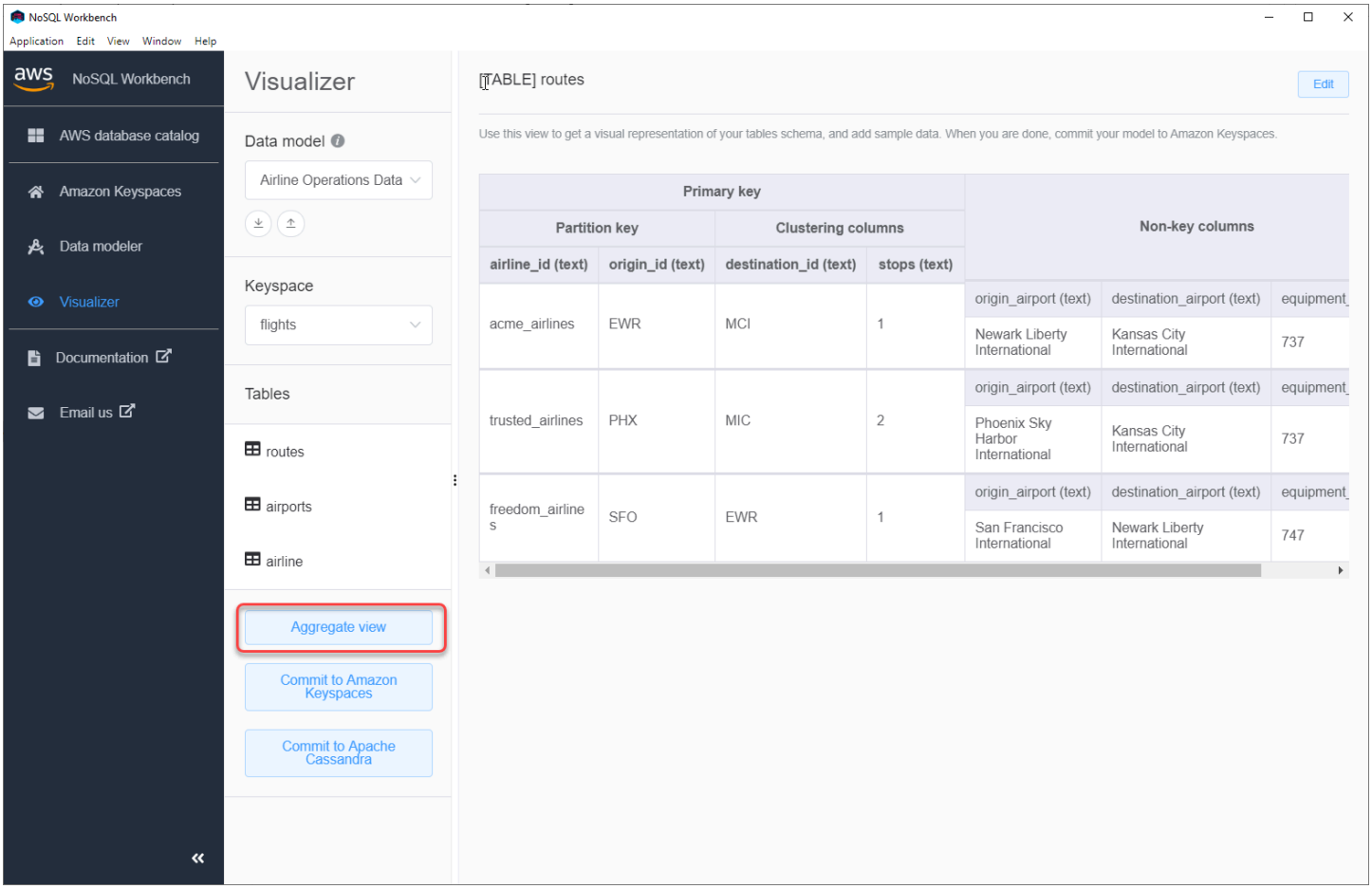


これにより、NoSQL Workbench のデータビジュアライザーに移動します。データビジュアライザーでは、テーブルのスキーマの視覚的表現と、サンプルデータの追加を行います。テーブルにサンプルデータを追加するには、モデルからテーブルを選択し、[Edit] (編集) を選択します。新しいデータ行を追加するには、画面の下部にある [Add new row] (新しい行の追加) を選択します。以上が完了したら、[Save] (保存) を選択します。



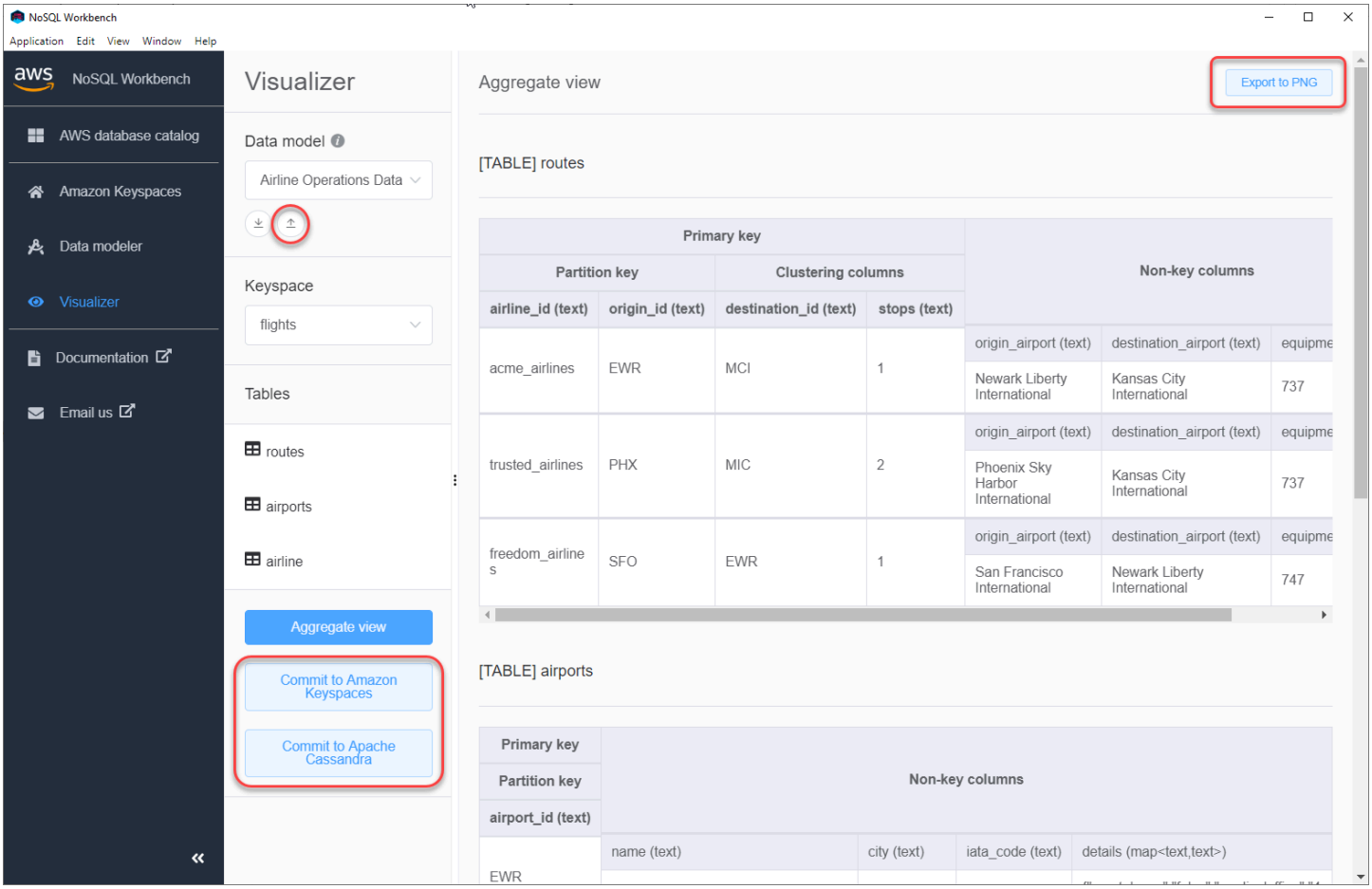
集約ビュー

テーブルのスキーマを確定したら、可視化されたデータモデルを集約できます。



データモデルのビューを集約したら、そのビューを PNG ファイルにエクスポートできます。データモデルを JSON ファイルにエクスポートするには、データモデル名の下にあるアップロードマークを選択します。

> **Note**
>
> 設計プロセスにおいて、JSON 形式でデータモデルをいつでもエクスポートできます。



以下は変更をコミットするためのオプションです。

- Amazon Keyspaces にコミットする
- Apache Cassandra クラスターにコミットする

変更をコミットする方法の詳細については、「the section called “データモデルのコミット”」を参照してください。

NoSQL Workbench で新しいデータモデルを作成する

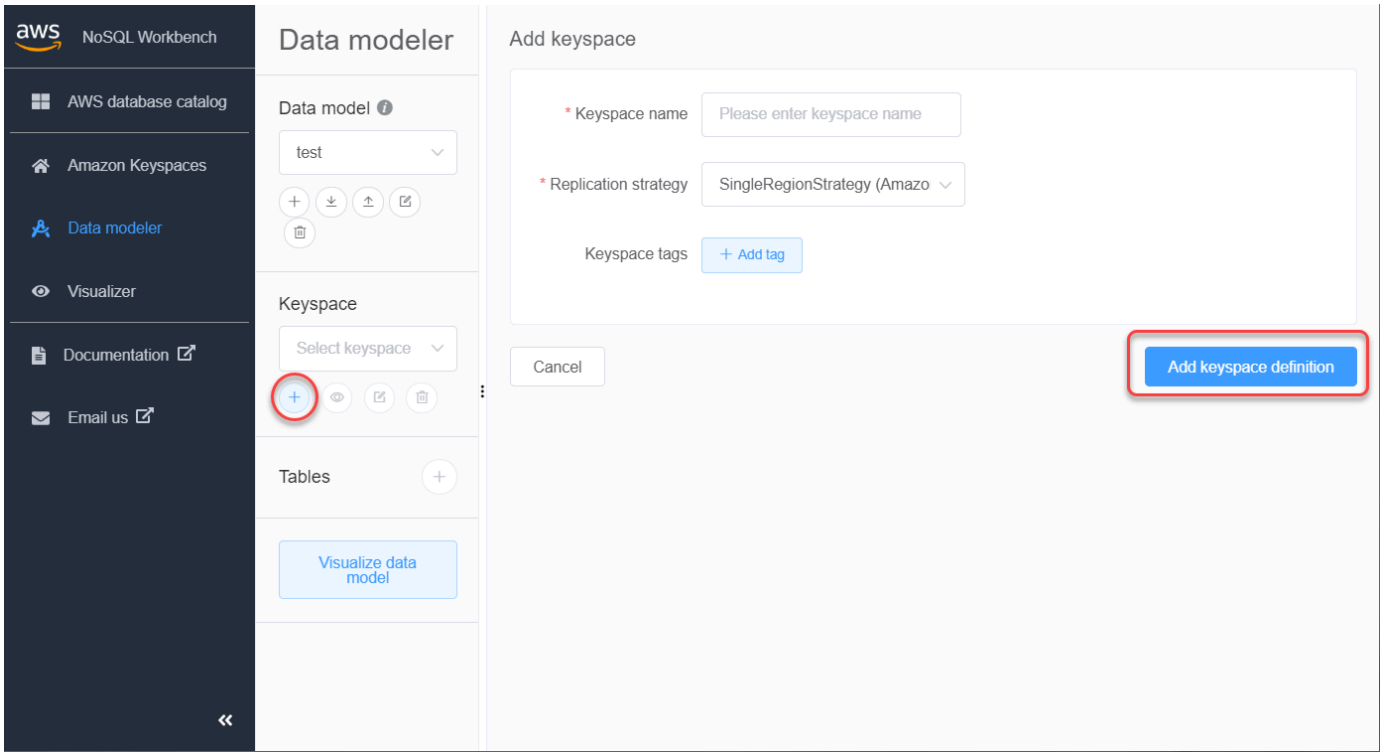
NoSQL Workbench のデータモデラーを使用すれば、お使いのアプリケーションのデータアクセスパターンに基づいて新しいモデルを設計することができます。Amazon Keyspaces の新しいデータモデルを作成するには、NoSQL Workbench データモデラーを使用してキースペース、テーブル、および列を作成します。新しいデータモデルを作成するには以下の手順を実行します。

1. 新しいキースペースを作成するには、[Keyspace] (キースペース) の下にあるプラス記号を選択します。

このステップでは、次のプロパティと設定を選択します。

- [Keyspace name] (キースペース名) — 新しいキースペースの名前を入力します。
- [Replication strategy] (レプリケーション戦略) - キースペースのレプリケーション戦略を選択します。Amazon Keyspaces は SingleRegionStrategy を使用して、複数の AWS アベイラビリティーゾーンでデータを 3 回自動的にレプリケートします。データモデルを Apache Cassandra クラスターにコミットする予定がある場合は、SimpleStrategy または NetworkTopologyStrategy を選択できます。
- [Keyspaces tags] (キースペースタグ) – リソースタグはオプションで、目的別、所有者別、環境別など、さまざまな方法でリソースを分類できます。Amazon Keyspaces リソースのタグの詳細については、「the section called “タグの操作”」を参照してください。

2. [Add keyspace definition] (キースペース定義の追加) を選択して、キースペースを作成します。

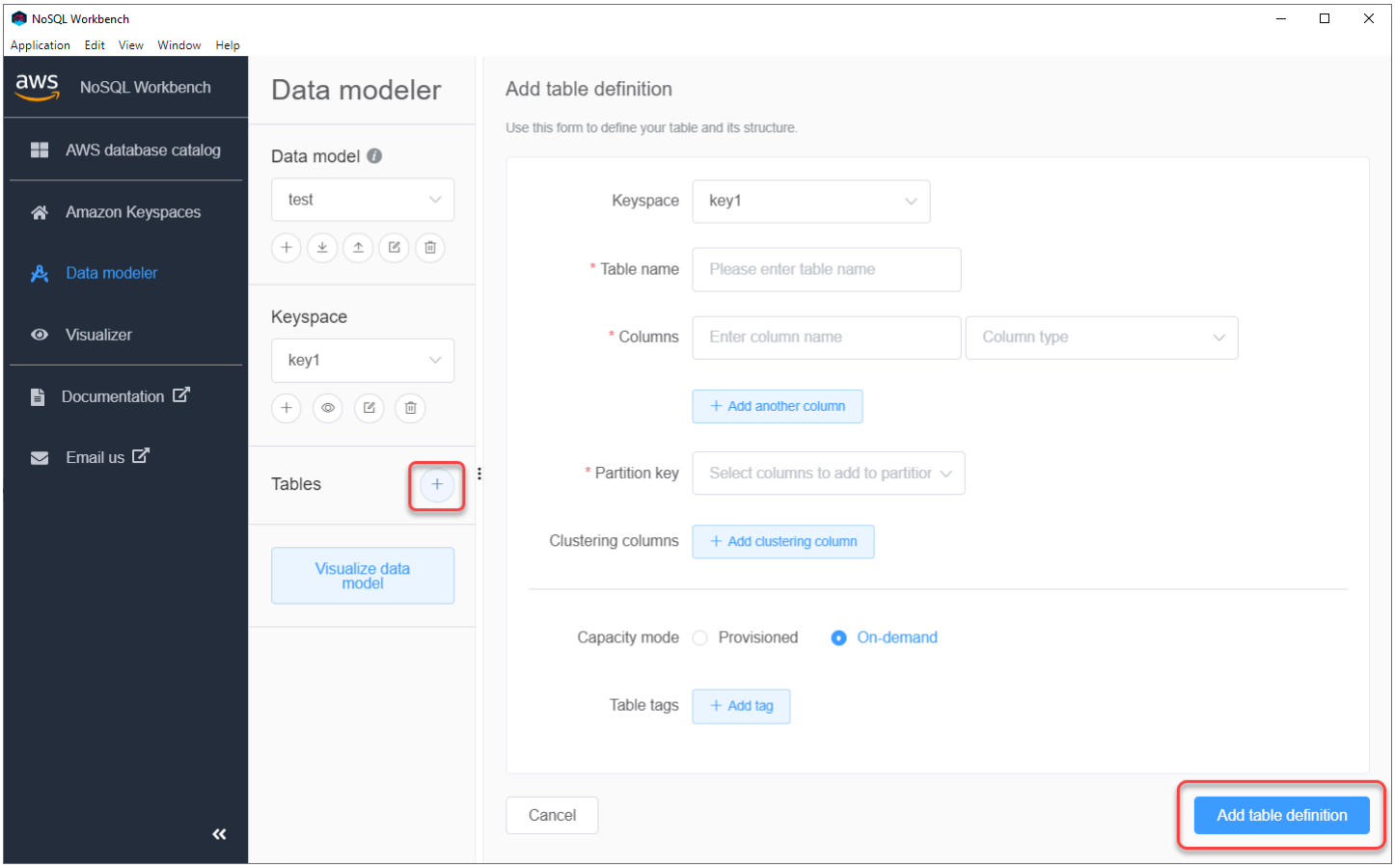


3. 新しいテーブルを作成するには、[Tables] (テーブル) の横にあるプラスマークを選択します。このステップでは、次のプロパティと設定を定義します。

- [Table name] (新しいテーブル名) – 新しいテーブルの名前。
- [Columns] (列) — 列名を追加し、データ型を選択します。スキーマのすべての列に対して、これらのステップを繰り返します。
- [Partition key] (パーティションキー) — パーティションキーの列を選択します。

- [Clustering columns] (クラスタリング列) — クラスタリング列を選択します (オプション)。
- [Capacity mode] (キャパシティモード) — テーブルの読み取り/書き込みキャパシティモードを選択します。プロビジョンドキャパシティまたはオンデマンドキャパシティを選択できます。キャパシティモードの詳細については、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。
- [Table tags] (テーブルタグ) – リソースタグはオプションで、目的別、所有者別、環境別など、さまざまな方法でリソースを分類できます。Amazon Keyspaces リソースのタグの詳細については、「the section called “タグの操作”」を参照してください。

4. [Add table definition] (テーブル定義の追加) を選択して、新しいテーブルを作成します。
5. 追加のテーブルを作成する場合はこのステップを繰り返します。
6. the section called “データモデルの可視化” に進み、作成したデータモデルを可視化します。



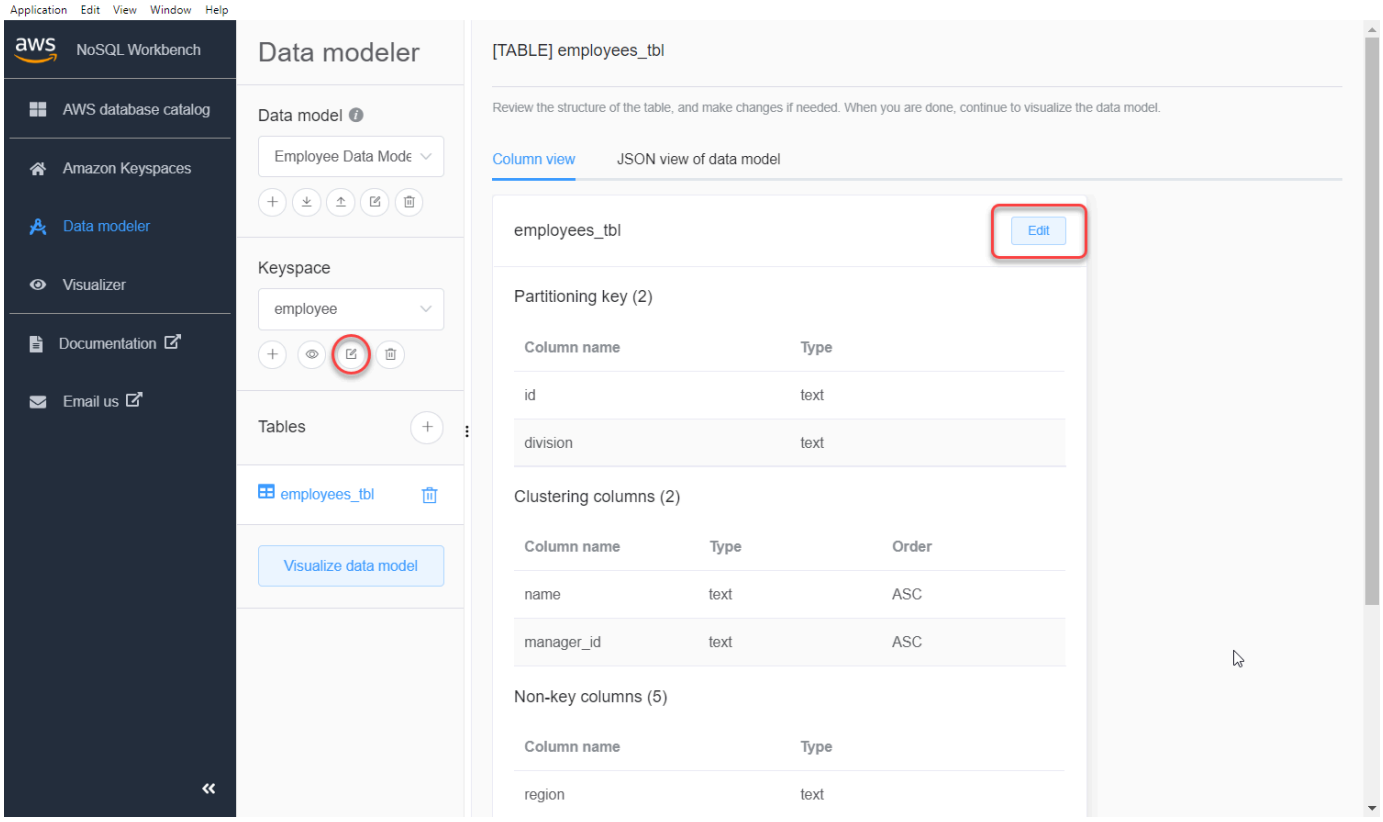
NoSQL Workbench で既存のデータモデルを編集する

データモデラーを使用すれば、NoSQL Workbench で作成された既存のデータモデルをインポートして変更できます。データモデラーには、データモデリングの開始時に役立つサンプルデータモデルがいくつか含まれています。NoSQL Workbench では、ファイルからインポートされたデータモデル、提供されたサンプルデータモデル、以前に作成したデータモデルを編集できます。

1. キースペースを編集するには、[Keyspace] (キースペース) の下にある編集マークを選択します。

   このステップでは、次のプロパティと設定を編集できます。

   - [Keyspace name] (キースペース名) — 新しいキースペースの名前を入力します。
   - [Replication strategy] (レプリケーション戦略) - キースペースのレプリケーション戦略を選択します。Amazon Keyspaces は SingleRegionStrategy を使用して、複数の AWS アベイラビリティーゾーンでデータを 3 回自動的にレプリケートします。データモデルを Apache Cassandra クラスターにコミットする予定がある場合は、SimpleStrategy または NetworkTopologyStrategy を選択できます。
   - [Keyspaces tags] (キースペースタグ) – リソースタグはオプションで、目的別、所有者別、環境別など、さまざまな方法でリソースを分類できます。Amazon Keyspaces リソースのタグの詳細については、「the section called “タグの操作”」を参照してください。
2. [Save edits] (編集内容を保存) を選択して、キースペースを更新します。



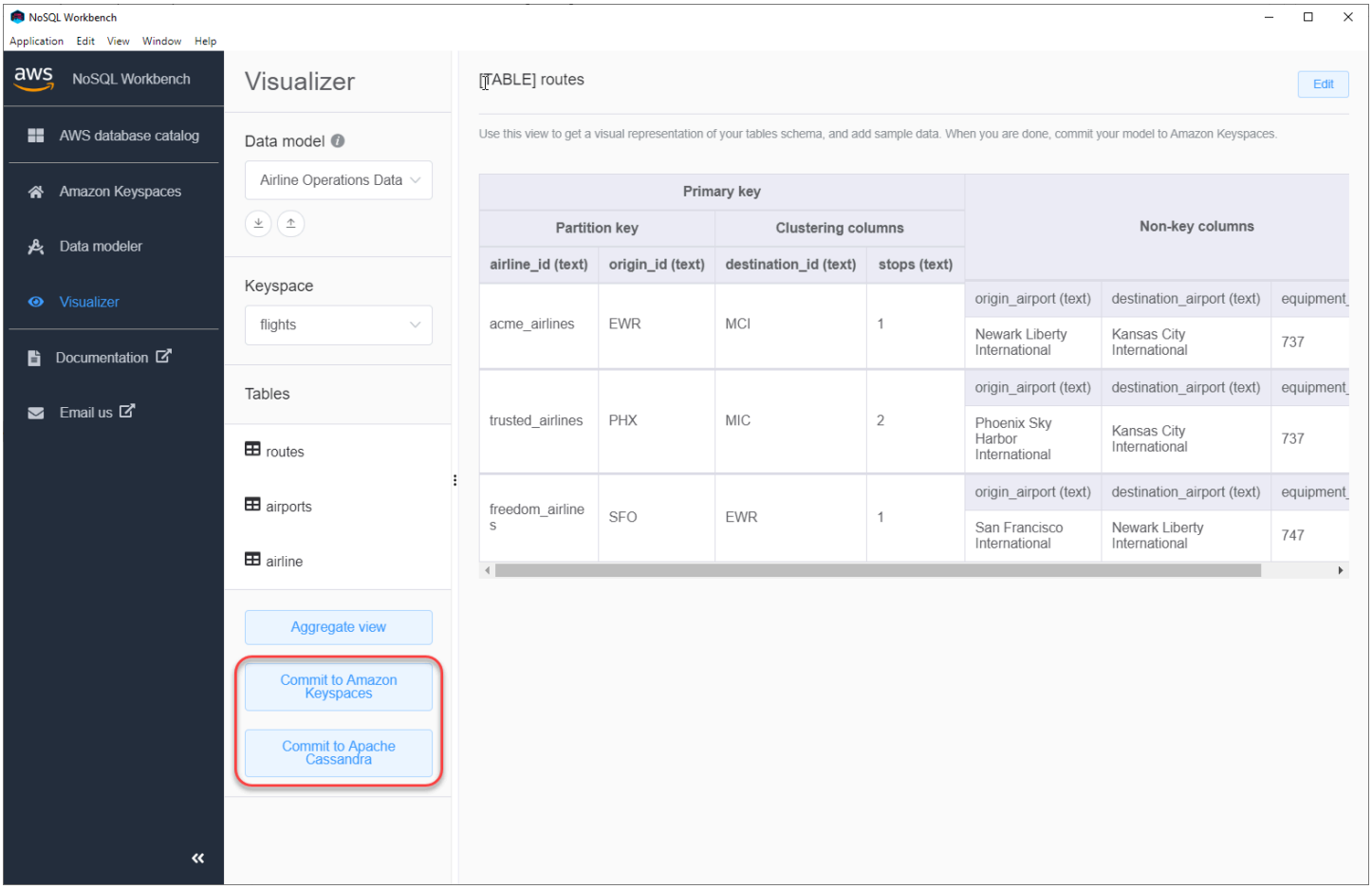
3. テーブルを編集するには、テーブル名の横にある [Edit] (編集) を選択します。このステップでは、次のプロパティと設定を更新できます。

   - [Table name] (新しいテーブル名) – 新しいテーブルの名前。
   - [Columns] (列) — 列名を追加し、データ型を選択します。スキーマのすべての列に対して、これらのステップを繰り返します。
   - [Partition key] (パーティションキー) — パーティションキーの列を選択します。
   - [Clustering columns] (クラスタリング列) — クラスタリング列を選択します (オプション)。
   - [Capacity mode] (キャパシティモード) — テーブルの読み取り/書き込みキャパシティモードを選択します。プロビジョンドキャパシティまたはオンデマンドキャパシティを選択できます。キャパシティモードの詳細については、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。
   - [Table tags] (テーブルタグ) – リソースタグはオプションで、目的別、所有者別、環境別など、さまざまな方法でリソースを分類できます。Amazon Keyspaces リソースのタグの詳細については、「the section called “タグの操作”」を参照してください。

4. [Save edits] (編集内容を保存) を選択して、テーブルを更新します。
5. the section called “データモデルの可視化” に進み、更新したデータモデルを可視化します。

Amazon Keyspaces と Apache Cassandra にデータモデルをコミットする方法

このセクションでは、完成したデータモデルを Amazon Keyspaces クラスターと Apache Cassandra クラスターにコミットする方法を示します。このプロセスでは、データモデルで定義した設定に基づいて、キースペースとテーブルのサーバー側リソースが自動的に作成されます。



トピック

- [開始する前に]
- サービス固有の認証情報で Amazon Keyspaces に接続する
- AWS Identity and Access Management (IAM) 認証情報を使用して Amazon Keyspaces に接続する
- 保存済み接続を使用する
- Apache Cassandra にコミットする

[開始する前に]

Amazon Keyspaces では、クライアントとの安全な接続を確保するために Transport Layer Security (TLS) を使用する必要があります。TLS を使用して Amazon Keyspaces への接続を開始する前に、以下のタスクを行う必要があります。

- 次のコマンドを使用して Starfield デジタル証明書をダウンロードし、`sf-class2-root.crt` をローカルまたはホームディレクトリ内に保存します。

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

> **Note**
>
> Amazon デジタル証明書を使用して Amazon Keyspaces に接続することもできます。クライアントが Amazon Keyspaces に正常に接続されている場合は、引き続き Amazon Keyspaces に接続できます。Starfield 証明書は、古い認定権限を使用しているクライアントに対して追加の下位互換性を提供するものです。


```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```


証明書ファイルを保存したら、Amazon Keyspaces に接続できます。1 つ目のオプションは、サービス固有の認証情報を使用した接続です。サービス固有の認証情報は、特定の IAM ユーザーに関連付けられているユーザー名とパスワードであり、指定されたサービスでしか使用できません。2 つ目のオプションは、AWS 署名バージョン 4 プロセス (SigV4) を使用している IAM 認証情報での接続です。これら 2 つのオプションの詳細については、「the section called “プログラムによるアクセスのための認証情報の作成”」を参照してください。

サービス固有の認証情報による接続については、「the section called “サービス固有の認証情報で接続する”」を参照してください。

IAM 認証情報による接続については、「the section called “IAM 認証情報で接続する”」を参照してください。

サービス固有の認証情報で Amazon Keyspaces に接続する

このセクションでは、サービス固有の認証情報を使用して、NoSQL Workbench で作成または編集したデータモデルをコミットする方法について説明します。

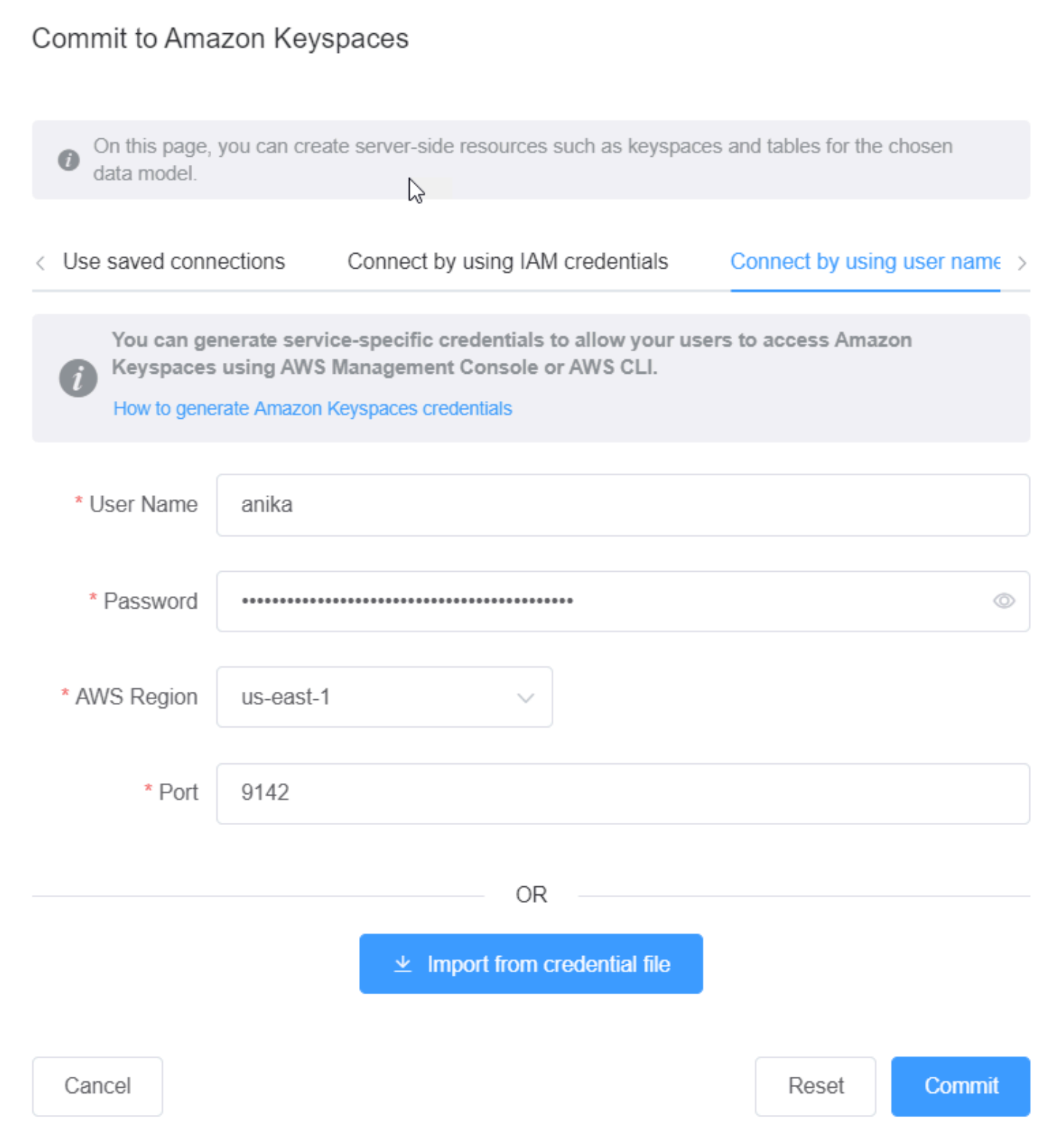
1. サービス固有の認証情報を使用して新しい接続を作成するには、[Connect by using user name and password] (ユーザー名とパスワードを使用した接続) タブを選択します。

   - 開始する前に、the section called “サービス固有の認証情報の作成” に記載されているプロセスを使用して、サービス固有の認証情報を作成しておく必要があります。

   サービス固有の認証情報を取得したら、続いて接続をセットアップできます。次のいずれかの操作を行います。

   - [User name] (ユーザー名) - ユーザー名を入力します。
   - [Password] (パスワード) – パスワードを入力します。
   - AWS リージョン — 使用可能なリージョンについては、「the section called “サービスエンドポイント”」を参照してください。
   - ポート — Amazon Keyspaces はポート 9142 を使用します。

   または、保存されている認証情報をファイルからインポートすることもできます。
2. [Commit] (コミット) を選択して、データモデルで Amazon Keyspaces を更新します。


Commit to Amazon Keyspaces
On this page, you can create server-side resources such as keyspaces and tables for the chosen data model.
Use saved connections
Connect by using IAM credentials
Connect by using user name
You can generate service-specific credentials to allow your users to access Amazon Keyspaces using AWS Management Console or AWS CLI.
How to generate Amazon Keyspaces credentials
* User Name
anika
* Password
* AWS Region
us-east-1
* Port
9142
OR
Import from credential file
Cancel
Reset
Commit

AWS Identity and Access Management (IAM) 認証情報を使用して Amazon Keyspaces に接続する

このセクションでは、IAM 認証情報を使用して、NoSQL Workbench で作成または編集したデータモデルをコミットする方法について説明します。

1. IAM 認証情報を使用して新しい接続を作成するには、[Connect by using IAM credentials] (IAM 認証情報を使用した接続) タブを選択します。

   - 開始する前に、次のいずれかの方法で IAM 認証情報を作成しておく必要があります。

     - コンソールアクセスの場合は、IAM ユーザー名とパスワードを使用して IAM サインインページから AWS Management Console にサインインします。プログラムによるアクセスや長期的な認証情報の代替など、 AWS セキュリティ認証情報の詳細については、「IAM ユーザーガイド」のAWS 「 セキュリティ認証情報」を参照してください。へのサインインの詳細については AWS アカウント、「 AWS サインイン ユーザーガイド」の「 へのサインイン方法 AWS」を参照してください。
     - CLI アクセスには、アクセスキー ID とシークレットアクセスキーが必要です。長期のアクセスキーの代わりに一時的な認証情報をできるだけ使用します。一時的な認証情報には、アクセスキー ID、シークレットアクセスキー、および認証情報の失効を示すセキュリティトークンが含まれています。詳細については、「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」を参照してください。
     - API アクセスには、アクセスキー ID とシークレットアクセスキーが必要です。 AWS アカウントのルートユーザー のアクセスキーの代わりに IAM ユーザーアクセスキーを使用します。アクセスキーの作成の詳細については、「IAM ユーザーガイド」の「IAM ユーザーのアクセスキーを管理する」を参照してください。

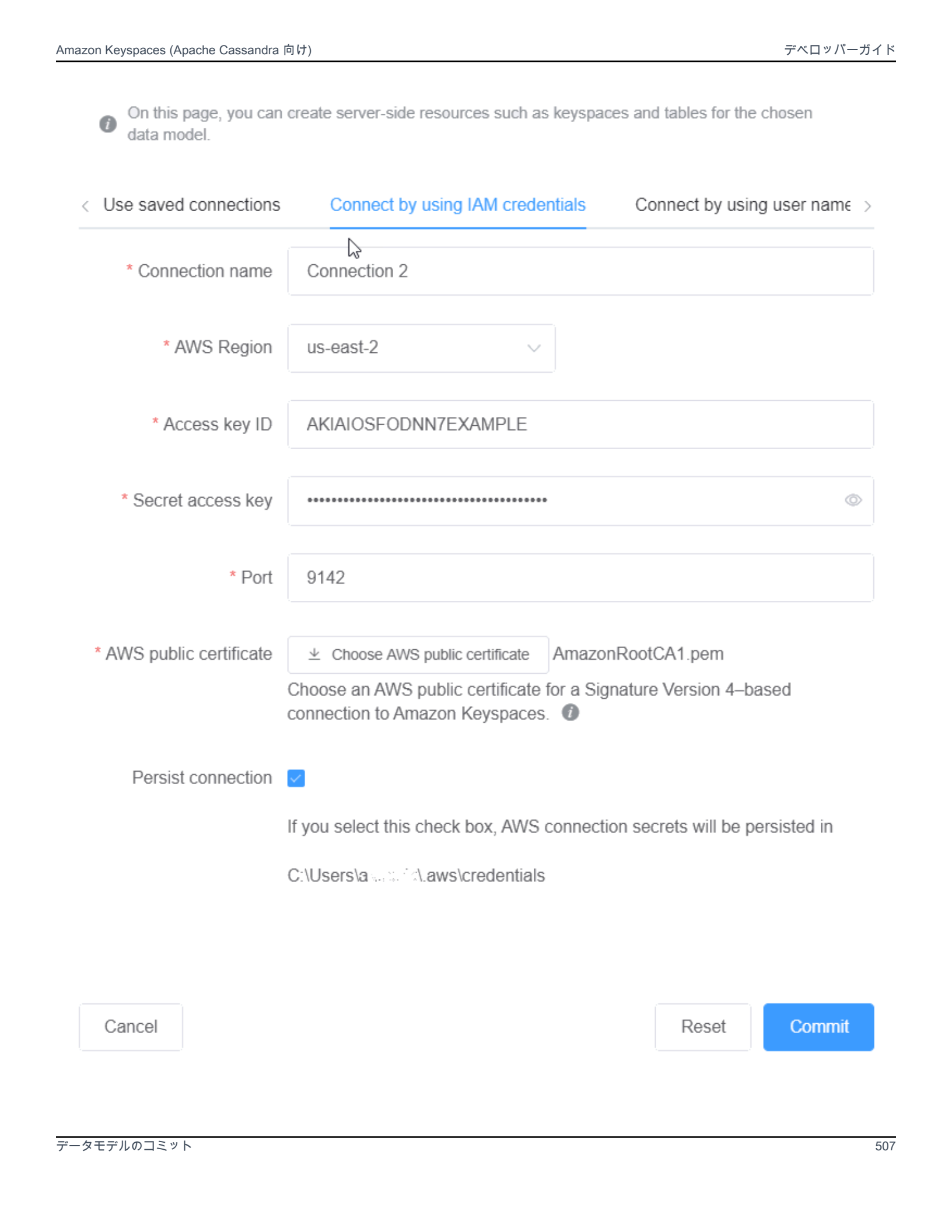
     詳細については、「Managing access keys for IAM users」(IAM ユーザーのアクセスキーの管理) を参照してください。

   IAM 認証情報を取得したら、続いて接続をセットアップできます。

   - [Connection name] (接続名) — 接続の名称。
   - AWS リージョン — 使用可能なリージョンについては、「the section called “サービスエンドポイント”」を参照してください。
   - [Access key ID] (アクセスキー ID) – アクセスキー ID を入力します。

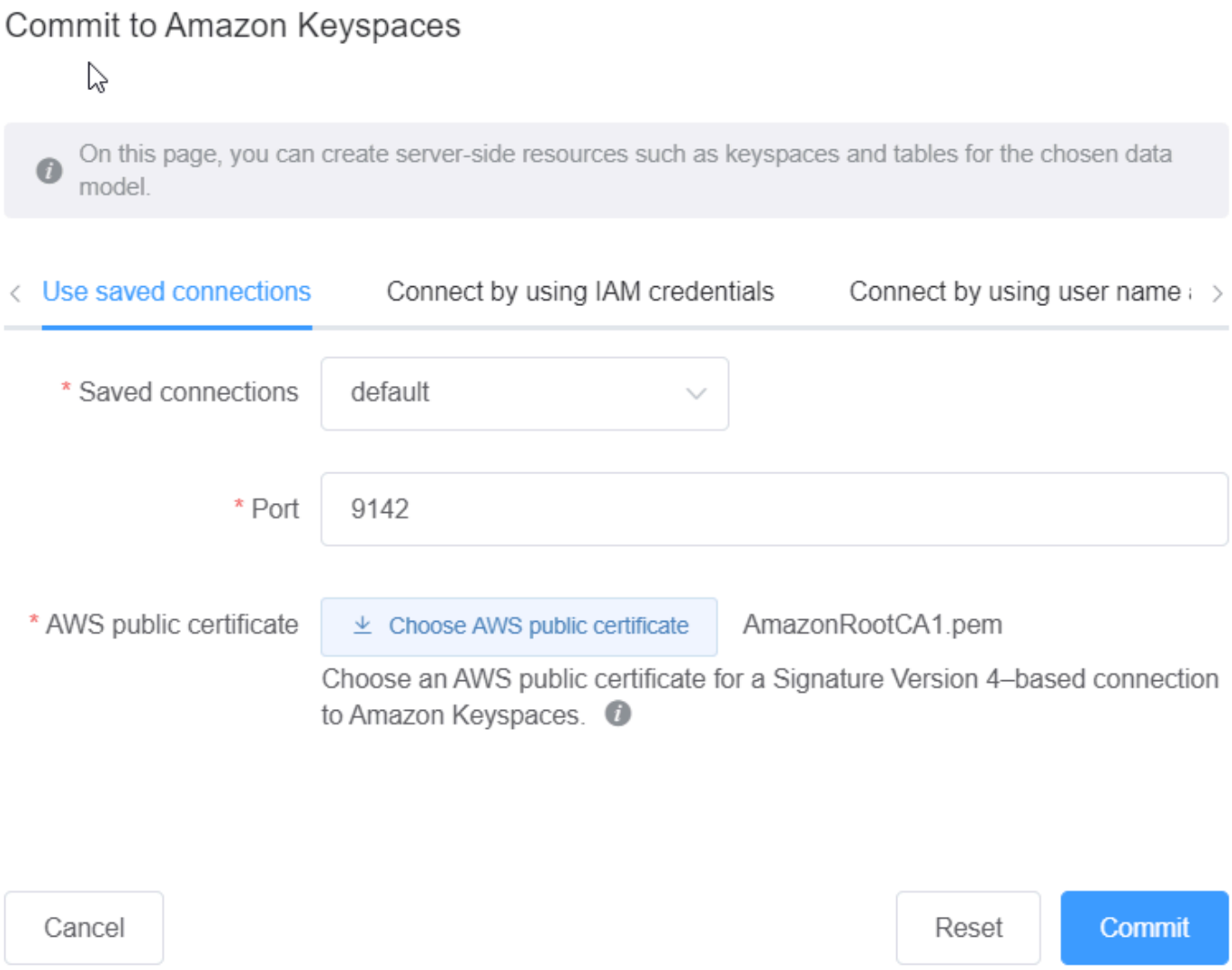
- [Secret access key] (シークレットアクセスキー) – シークレットアクセスキーを入力します。
- ポート — Amazon Keyspaces はポート 9142 を使用します。
- AWS パブリック証明書 – 最初のステップでダウンロードした証明書を指します AWS 。
- 接続の永続化 – AWS 接続シークレットをローカルに保存する場合は、このチェックボックスをオンにします。

2. [Commit] (コミット) を選択して、データモデルで Amazon Keyspaces を更新します。


On this page, you can create server-side resources such as keyspaces and tables for the chosen data model.
Use saved connections
Connect by using IAM credentials
Connect by using user name
* Connection name
Connection 2
* AWS Region
us-east-2
* Access key ID
AKIAIOSFODNN7EXAMPLE
* Secret access key
* Port
9142
* AWS public certificate
Choose AWS public certificate
AmazonRootCA1.pem
Choose an AWS public certificate for a Signature Version 4–based connection to Amazon Keyspaces.
Persist connection
If you select this check box, AWS connection secrets will be persisted in
C:\Users\a
\.aws\credentials
Cancel
Reset
Commit

保存済み接続を使用する

Amazon Keyspaces への接続を事前に設定している場合は、それをデフォルトの接続として使用して、データモデルの変更をコミットすることができます。[Use saved connections] (保存済み接続の使用) タブを選択して、更新情報に進みます。

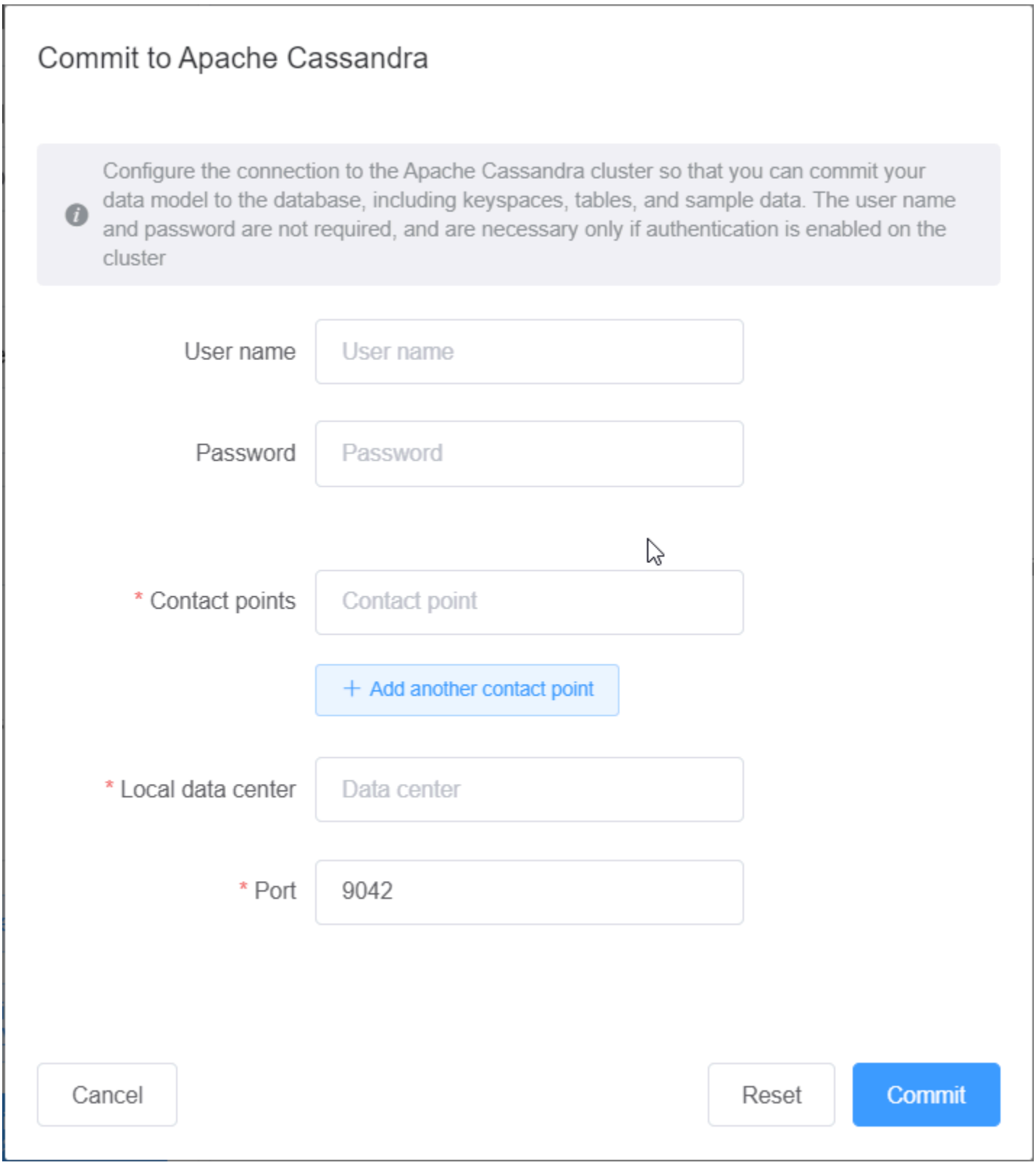


Apache Cassandra にコミットする

このセクションでは、Apache Cassandra クラスターに接続して NoSQL Workbench で作成または編集したデータモデルをコミットする手順について説明します。

> **Note**
>
> `SimpleStrategy` または `NetworkTopologyStrategy` で作成されたデータモデルしか Apache Cassandra クラスターにコミットできません。レプリケーション戦略を変更するには、データモデラーでキースペースを編集します。

1.
   - [User name] (ユーザー名) — クラスターで認証が有効になっている場合は、ユーザー名を入力します。
   - [Password] (パスワード) — クラスターで認証が有効になっている場合は、パスワードを入力します。
   - [Contact points] (コンタクトポイント) — コンタクトポイントを入力します。
   - [Local data center] (ローカルデータセンター) — ローカルデータセンターの名前を入力します。
   - [Port] (ポート) — 接続にはポート 9042 を使用します。
2. [Commit] (コミット) を選択して、データモデルで Apache Cassandra クラスターを更新します。


Commit to Apache Cassandra
Configure the connection to the Apache Cassandra cluster so that you can commit your data model to the database, including keyspaces, tables, and sample data. The user name and password are not required, and are necessary only if authentication is enabled on the cluster
User name
User name
Password
Password
* Contact points
Contact point
+ Add another contact point
* Local data center
Data center
* Port
9042
Cancel
Reset
Commit

NoSQL Workbench のサンプルデータモデル

モデラーとビジュアライザーのホームページには、NoSQL Workbench に付属のサンプルモデルが多数表示されます。このセクションでは、これらのモデルとその可能性のある用途について説明します。

トピック

- 従業員データモデル
- クレジットカード取引データモデル
- 航空関連オペレーションデータモデル

従業員データモデル

このデータモデルは、従業員データベースアプリケーションの Amazon Keyspaces スキーマを表します。

特定の会社の従業員情報にアクセスするアプリケーションで、このデータモデルを使用できます。

このデータモデルでサポートされているアクセスパターンは次の通りです。

- ID を指定した従業員レコードの検索。
- ID と部署を指定した従業員レコードの検索。
- ID と氏名を指定した従業員レコードの検索。

クレジットカード取引データモデル

このデータモデルは、小売店でのクレジットカード取引の Amazon Keyspaces スキーマを表します。

クレジットカード取引情報を保存しておくと、店舗の簿記に役立つだけでなく、店舗管理者が購入傾向を分析する際にも役立ち、結果的に店舗管理者による予測や計画に役立ちます。

このデータモデルでサポートされているアクセスパターンは次の通りです。

- クレジットカード番号別、年月別、日別での取引情報の検索。
- クレジットカード番号別、カテゴリ別、日別での取引情報の検索。
- カテゴリ別、場所別、クレジットカード番号別での取引情報の検索。

- クレジットカード番号別、係争状況別での取引情報の検索。

航空関連オペレーションデータモデル

このデータモデルは、空港、航空会社、飛行ルートを含む飛行機移動に関するデータを示します。

実証されている Amazon Keyspaces モデリングの主要コンポーネントとは、キーバリューペア、ワイド列データストア、複合キー、一般的な NoSQL データアクセスパターンを実証するための一般的なデータ型 (マップなど) です。

このデータモデルでサポートされているアクセスパターンは次の通りです。

- 出発空港と航空会社を指定したルート検索。
- 到着空港を指定したルート検索。
- 直行便がある空港の検索。
- 空港の詳細と航空会社の詳細の検索。

NoSQL Workbench のリリース履歴

NoSQL Workbench クライアント側アプリケーションの各リリースにおける重要な変更点を次の表に示します。

変更	説明	日付
Amazon Keyspaces 用 NoSQL Workbench — GA。	Amazon Keyspaces 用 NoSQL Workbench の一般提供が開始されました。	2020 年 10 月 28 日
NoSQL Workbench プレビュー版がリリースされました。	NoSQL Workbench は、Amazon Keyspaces の非リレーショナルデータモデルの設計と可視化をより簡単にするクライアント側アプリケーションです。NoSQL Workbench クライアントは、Windows、macOS、Linux で使用できます。	2020 年 10 月 5 日

変更	説明	日付
	詳細については、「NoSQL Workbench for Amazon Keyspaces」(Amazon Keyspaces 用 NoSQL Workbench) を参照してください。	

AWS SDKsコード例

次のコード例は、 AWS Software Development Kit (SDK) で Amazon Keyspaces を使用する方法を示しています。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

開始方法

Hello Amazon Keyspaces

次のコード例は、Amazon Keyspaces の使用を開始する方法を示しています。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace KeyspacesActions;

public class HelloKeyspaces
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Keyspaces (for Apache
 Cassandra).
```

```
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonKeyspaces>()
                .AddTransient<KeyspacesWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloKeyspaces>();

        var keyspacesClient =
 host.Services.GetRequiredService<IAmazonKeyspaces>();
        var keyspacesWrapper = new KeyspacesWrapper(keyspacesClient);

        Console.WriteLine("Hello, Amazon Keyspaces! Let's list your keyspaces:");
        await keyspacesWrapper.ListKeyspaces();
    }
}
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「ListKeyspaces」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.keyspaces.KeyspacesClient;
import software.amazon.awssdk.services.keyspaces.model.KeyspaceSummary;
import software.amazon.awssdk.services.keyspaces.model.KeyspacesException;
import software.amazon.awssdk.services.keyspaces.model.ListKeyspacesRequest;
import software.amazon.awssdk.services.keyspaces.model.ListKeyspacesResponse;
import java.util.List;

/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloKeyspaces {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        KeyspacesClient keyClient = KeyspacesClient.builder()
                .region(region)
                .build();

        listKeyspaces(keyClient);
    }

    public static void listKeyspaces(KeyspacesClient keyClient) {
        try {
            ListKeyspacesRequest keyspacesRequest =
 ListKeyspacesRequest.builder()
                    .maxResults(10)
                    .build();

            ListKeyspacesResponse response =
 keyClient.listKeyspaces(keyspacesRequest);
            List<KeyspaceSummary> keyspaces = response.keyspaces();
            for (KeyspaceSummary keyspace : keyspaces) {
                System.out.println("The name of the keyspace is " +
 keyspace.keyspaceName());
            }

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
```

```
        }
    }
}
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「ListKeyspaces」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
*/

suspend fun main() {
    listKeyspaces()
}

suspend fun listKeyspaces() {
    val keyspacesRequest =
        ListKeyspacesRequest {
            maxResults = 10
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.listKeyspaces(keyspacesRequest)
        response.keyspaces?.forEach { keyspace ->
```

```
            println("The name of the keyspace is ${keyspace.keyspaceName}")
        }
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「ListKeyspaces」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import boto3


def hello_keyspaces(keyspaces_client):
    """
    Use the AWS SDK for Python (Boto3) to create an Amazon Keyspaces (for Apache
 Cassandra)
    client and list the keyspaces in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param keyspaces_client: A Boto3 Amazon Keyspaces Client object. This object
 wraps
                             the low-level Amazon Keyspaces service API.
    """
    print("Hello, Amazon Keyspaces! Let's list some of your keyspaces:\n")
    for ks in keyspaces_client.list_keyspaces(maxResults=5).get("keyspaces", []):
        print(ks["keyspaceName"])
        print(f"\t{ks['resourceArn']}")


if __name__ == "__main__":
```

```
    hello_keyspaces(boto3.client("keyspaces"))
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「ListKeyspaces」を参照してください。

コードの例

- AWS SDKs基本的な例
  - Hello Amazon Keyspaces
  - AWS SDK を使用した Amazon Keyspaces の基本について説明します。
  - AWS SDKsアクション
    - AWS SDK CreateKeyspaceで を使用する
    - AWS SDK CreateTableで を使用する
    - AWS SDK DeleteKeyspaceで を使用する
    - AWS SDK DeleteTableで を使用する
    - AWS SDK GetKeyspaceで を使用する
    - AWS SDK GetTableで を使用する
    - AWS SDK ListKeyspacesで を使用する
    - AWS SDK ListTablesで を使用する
    - AWS SDK RestoreTableで を使用する
    - AWS SDK UpdateTableで を使用する

AWS SDKs基本的な例

次のコード例では、Amazon Keyspaces (Apache Cassandra 向け) の基本操作を AWS SDK で行う方法を説明します。

例

- Hello Amazon Keyspaces
- AWS SDK を使用した Amazon Keyspaces の基本について説明します。
- AWS SDKsアクション
  - AWS SDK CreateKeyspaceで を使用する
  - AWS SDK CreateTableで を使用する

- AWS SDK DeleteKeyspaceで を使用する
- AWS SDK DeleteTableで を使用する
- AWS SDK GetKeyspaceで を使用する
- AWS SDK GetTableで を使用する
- AWS SDK ListKeyspacesで を使用する
- AWS SDK ListTablesで を使用する
- AWS SDK RestoreTableで を使用する
- AWS SDK UpdateTableで を使用する

Hello Amazon Keyspaces

次のコード例は、Amazon Keyspaces の使用を開始する方法を示しています。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace KeyspacesActions;

public class HelloKeyspaces
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Keyspaces (for Apache
 Cassandra).
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
```

```
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonKeyspaces>()
                .AddTransient<KeyspacesWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloKeyspaces>();

        var keyspacesClient =
 host.Services.GetRequiredService<IAmazonKeyspaces>();
        var keyspacesWrapper = new KeyspacesWrapper(keyspacesClient);

        Console.WriteLine("Hello, Amazon Keyspaces! Let's list your keyspaces:");
        await keyspacesWrapper.ListKeyspaces();
    }
}
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「ListKeyspaces」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.keyspaces.KeyspacesClient;
import software.amazon.awssdk.services.keyspaces.model.KeyspaceSummary;
import software.amazon.awssdk.services.keyspaces.model.KeyspacesException;
import software.amazon.awssdk.services.keyspaces.model.ListKeyspacesRequest;
import software.amazon.awssdk.services.keyspaces.model.ListKeyspacesResponse;
```

```
import java.util.List;

/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloKeyspaces {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        KeyspacesClient keyClient = KeyspacesClient.builder()
                .region(region)
                .build();

        listKeyspaces(keyClient);
    }

    public static void listKeyspaces(KeyspacesClient keyClient) {
        try {
            ListKeyspacesRequest keyspacesRequest =
 ListKeyspacesRequest.builder()
                    .maxResults(10)
                    .build();

            ListKeyspacesResponse response =
 keyClient.listKeyspaces(keyspacesRequest);
            List<KeyspaceSummary> keyspaces = response.keyspaces();
            for (KeyspaceSummary keyspace : keyspaces) {
                System.out.println("The name of the keyspace is " +
 keyspace.keyspaceName());
            }

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「ListKeyspaces」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
*/

suspend fun main() {
    listKeyspaces()
}

suspend fun listKeyspaces() {
    val keyspacesRequest =
        ListKeyspacesRequest {
            maxResults = 10
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.listKeyspaces(keyspacesRequest)
        response.keyspaces?.forEach { keyspace ->
            println("The name of the keyspace is ${keyspace.keyspaceName}")
        }
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「ListKeyspaces」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import boto3


def hello_keyspaces(keyspaces_client):
    """
    Use the AWS SDK for Python (Boto3) to create an Amazon Keyspaces (for Apache
 Cassandra)
    client and list the keyspaces in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param keyspaces_client: A Boto3 Amazon Keyspaces Client object. This object
 wraps
                             the low-level Amazon Keyspaces service API.
    """
    print("Hello, Amazon Keyspaces! Let's list some of your keyspaces:\n")
    for ks in keyspaces_client.list_keyspaces(maxResults=5).get("keyspaces", []):
        print(ks["keyspaceName"])
        print(f"\t{ks['resourceArn']}")


if __name__ == "__main__":
    hello_keyspaces(boto3.client("keyspaces"))
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「ListKeyspaces」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した Amazon Keyspaces の基本について説明します。

次のコード例は、以下を実行する方法を示しています。

- キースペースとテーブルを作成します。テーブルスキーマにはムービーデータが保存され、特定時点のリカバリが有効になっています。
- SIGv4 認証による安全な TLS 接続を使用してキースペースに接続します。
- テーブルに対してクエリを実行します。ムービーデータを追加、取得、更新します。
- テーブルを更新する。視聴したムービーを追跡する列を追加します。
- テーブルを以前の状態に戻し、リソースをクリーンアップします。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
global using System.Security.Cryptography.X509Certificates;
global using Amazon.Keyspaces;
global using Amazon.Keyspaces.Model;
global using KeyspacesActions;
global using KeyspacesScenario;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;
global using Newtonsoft.Json;
```

```
namespace KeyspacesBasics;

/// <summary>
/// Amazon Keyspaces (for Apache Cassandra) scenario. Shows some of the basic
/// actions performed with Amazon Keyspaces.
/// </summary>
public class KeyspacesBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonKeyspaces>()
            .AddTransient<KeyspacesWrapper>()
            .AddTransient<CassandraWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<KeyspacesBasics>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var keyspacesWrapper =
 host.Services.GetRequiredService<KeyspacesWrapper>();
        var uiMethods = new UiMethods();

        var keyspaceName = configuration["KeyspaceName"];
        var tableName = configuration["TableName"];
```

```
        bool success; // Used to track the results of some operations.

        uiMethods.DisplayOverview();
        uiMethods.PressEnter();

        // Create the keyspace.
        var keyspaceArn = await keyspacesWrapper.CreateKeyspace(keyspaceName);

        // Wait for the keyspace to be available. GetKeyspace results in a
        // resource not found error until it is ready for use.
        try
        {
            var getKeyspaceArn = "";
            Console.Write($"Created {keyspaceName}. Waiting for it to become
available. ");
            do
            {
                getKeyspaceArn = await
keyspacesWrapper.GetKeyspace(keyspaceName);
                Console.Write(". ");
            } while (getKeyspaceArn != keyspaceArn);
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine("Waiting for keyspace to be created.");
        }

        Console.WriteLine($"\nThe keyspace {keyspaceName} is ready for use.");

        uiMethods.PressEnter();

        // Create the table.
        // First define the schema.
        var allColumns = new List<ColumnDefinition>
        {
            new ColumnDefinition { Name = "title", Type = "text" },
            new ColumnDefinition { Name = "year", Type = "int" },
            new ColumnDefinition { Name = "release_date", Type = "timestamp" },
            new ColumnDefinition { Name = "plot", Type = "text" },
        };

        var partitionKeys = new List<PartitionKey>
        {
            new PartitionKey { Name = "year", },
```

```
            new PartitionKey { Name = "title" },
        };

        var tableSchema = new SchemaDefinition
        {
            AllColumns = allColumns,
            PartitionKeys = partitionKeys,
        };

        var tableArn = await keyspacesWrapper.CreateTable(keyspaceName,
tableSchema, tableName);

        // Wait for the table to be active.
        try
        {
            var resp = new GetTableResponse();
            Console.Write("Waiting for the new table to be active. ");
            do
            {
                try
                {
                    resp = await keyspacesWrapper.GetTable(keyspaceName,
tableName);
                    Console.Write(".");
                }
                catch (ResourceNotFoundException)
                {
                    Console.Write(".");
                }
            } while (resp.Status != TableStatus.ACTIVE);

            // Display the table's schema.
            Console.WriteLine($"\nTable {tableName} has been created in
{keyspaceName}");
            Console.WriteLine("Let's take a look at the schema.");
            uiMethods.DisplayTitle("All columns");
            resp.SchemaDefinition.AllColumns.ForEach(column =>
            {
                Console.WriteLine($"{column.Name,-40}\t{column.Type,-20}");
            });

            uiMethods.DisplayTitle("Cluster keys");
            resp.SchemaDefinition.ClusteringKeys.ForEach(clusterKey =>
            {
```

```
Console.WriteLine($"{clusterKey.Name,-40}\t{clusterKey.OrderBy,-20}");
            });

            uiMethods.DisplayTitle("Partition keys");
            resp.SchemaDefinition.PartitionKeys.ForEach(partitionKey =>
            {
                Console.WriteLine($"{partitionKey.Name}");
            });

            uiMethods.PressEnter();
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }

        // Access Apache Cassandra using the Cassandra drive for C#.
        var cassandraWrapper =
host.Services.GetRequiredService<CassandraWrapper>();
        var movieFilePath = configuration["MovieFile"];

        Console.WriteLine("Let's add some movies to the table we created.");
        var inserted = await cassandraWrapper.InsertIntoMovieTable(keyspaceName,
tableName, movieFilePath);

        uiMethods.PressEnter();

        Console.WriteLine("Added the following movies to the table:");
        var rows = await cassandraWrapper.GetMovies(keyspaceName, tableName);
        uiMethods.DisplayTitle("All Movies");

        foreach (var row in rows)
        {
            var title = row.GetValue<string>("title");
            var year = row.GetValue<int>("year");
            var plot = row.GetValue<string>("plot");
            var release_date = row.GetValue<DateTime>("release_date");
            Console.WriteLine($"{release_date}\t{title}\t{year}\n{plot}");
            Console.WriteLine(uiMethods.SepBar);
        }

        // Update the table schema
        uiMethods.DisplayTitle("Update table schema");
```

```
        Console.WriteLine("Now we will update the table to add a boolean field
called watched.");

        // First save the current time as a UTC Date so the original
        // table can be restored later.
        var timeChanged = DateTime.UtcNow;

        // Now update the schema.
        var resourceArn = await keyspacesWrapper.UpdateTable(keyspaceName,
tableName);
        uiMethods.PressEnter();

        Console.WriteLine("Now let's mark some of the movies as watched.");

        // Pick some files to mark as watched.
        var movieToWatch = rows[2].GetValue<string>("title");
        var watchedMovieYear = rows[2].GetValue<int>("year");
        var changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[6].GetValue<string>("title");
        watchedMovieYear = rows[6].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[9].GetValue<string>("title");
        watchedMovieYear = rows[9].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[10].GetValue<string>("title");
        watchedMovieYear = rows[10].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[13].GetValue<string>("title");
        watchedMovieYear = rows[13].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        uiMethods.DisplayTitle("Watched movies");
        Console.WriteLine("These movies have been marked as watched:");
        rows = await cassandraWrapper.GetWatchedMovies(keyspaceName, tableName);
        foreach (var row in rows)
```

```
        {
            var title = row.GetValue<string>("title");
            var year = row.GetValue<int>("year");
            Console.WriteLine($"{title,-40}\t{year,8}");
        }
        uiMethods.PressEnter();

        Console.WriteLine("We can restore the table to its previous state but
that can take up to 20 minutes to complete.");
        string answer;
        do
        {
            Console.WriteLine("Do you want to restore the table? (y/n)");
            answer = Console.ReadLine();
        } while (answer.ToLower() != "y" && answer.ToLower() != "n");

        if (answer == "y")
        {
            var restoredTableName = $"{tableName}_restored";
            var restoredTableArn = await keyspacesWrapper.RestoreTable(
                keyspaceName,
                tableName,
                restoredTableName,
                timeChanged);
            // Loop and call GetTable until the table is gone. Once it has been
            // deleted completely, GetTable will raise a
ResourceNotFoundException.
            bool wasRestored = false;

            try
            {
                do
                {
                    var resp = await keyspacesWrapper.GetTable(keyspaceName,
restoredTableName);
                    wasRestored = (resp.Status == TableStatus.ACTIVE);
                } while (!wasRestored);
            }
            catch (ResourceNotFoundException)
            {
                // If the restored table raised an error, it isn't
                // ready yet.
                Console.Write(".");
            }
```

```
        }

        uiMethods.DisplayTitle("Clean up resources.");

        // Delete the table.
        success = await keyspacesWrapper.DeleteTable(keyspaceName, tableName);

        Console.WriteLine($"Table {tableName} successfully deleted from
 {keyspaceName}.");
        Console.WriteLine("Waiting for the table to be removed completely. ");

        // Loop and call GetTable until the table is gone. Once it has been
        // deleted completely, GetTable will raise a ResourceNotFoundException.
        bool wasDeleted = false;

        try
        {
            do
            {
                var resp = await keyspacesWrapper.GetTable(keyspaceName,
 tableName);
            } while (!wasDeleted);
        }
        catch (ResourceNotFoundException ex)
        {
            wasDeleted = true;
            Console.WriteLine($"{ex.Message} indicates that the table has been
 deleted.");
        }

        // Delete the keyspace.
        success = await keyspacesWrapper.DeleteKeyspace(keyspaceName);
        Console.WriteLine("The keyspace has been deleted and the demo is now
 complete.");
    }
}
```

```
namespace KeyspacesActions;

/// <summary>
/// Performs Amazon Keyspaces (for Apache Cassandra) actions.
```

```
/// </summary>
public class KeyspacesWrapper
{
    private readonly IAmazonKeyspaces _amazonKeyspaces;

    /// <summary>
    /// Constructor for the KeyspaceWrapper.
    /// </summary>
    /// <param name="amazonKeyspaces">An Amazon Keyspaces client object.</param>
    public KeyspacesWrapper(IAmazonKeyspaces amazonKeyspaces)
    {
        _amazonKeyspaces = amazonKeyspaces;
    }

    /// <summary>
    /// Create a new keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name for the new keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
    public async Task<string> CreateKeyspace(string keyspaceName)
    {
        var response =
            await _amazonKeyspaces.CreateKeyspaceAsync(
                new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }


    /// <summary>
    /// Create a new Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace where the table will be
 created.</param>
    /// <param name="schema">The schema for the new table.</param>
    /// <param name="tableName">The name of the new table.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
    public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
 schema, string tableName)
    {
        var request = new CreateTableRequest
        {
            KeyspaceName = keyspaceName,
            SchemaDefinition = schema,
            TableName = tableName,
```

```
            PointInTimeRecovery = new PointInTimeRecovery { Status =
PointInTimeRecoveryStatus.ENABLED }
        };

        var response = await _amazonKeyspaces.CreateTableAsync(request);
        return response.ResourceArn;
    }


    /// <summary>
    /// Delete an existing keyspace.
    /// </summary>
    /// <param name="keyspaceName"></param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
            new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTable(string keyspaceName, string tableName)
    {
        var response = await _amazonKeyspaces.DeleteTableAsync(
            new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Get data about a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
    public async Task<string> GetKeyspace(string keyspaceName)
    {
```

```
        var response = await _amazonKeyspaces.GetKeyspaceAsync(
            new GetKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }


    /// <summary>
    /// Get information about an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
            new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response;
    }


    /// <summary>
    /// Lists all keyspaces for the account.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task ListKeyspaces()
    {
        var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

        Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
        Console.WriteLine(new string('-', Console.WindowWidth));
        await foreach (var keyspace in paginator.Keyspaces)
        {

Console.WriteLine($"{keyspace.KeyspaceName,-30}\t{keyspace.ResourceArn}");
        }
    }


    /// <summary>
    /// Lists the Amazon Keyspaces tables in a keyspace.
    /// </summary>
```

```
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>A list of TableSummary objects.</returns>
    public async Task<List<TableSummary>> ListTables(string keyspaceName)
    {
        var response = await _amazonKeyspaces.ListTablesAsync(new
 ListTablesRequest { KeyspaceName = keyspaceName });
        response.Tables.ForEach(table =>
        {

 Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
        });

        return response.Tables;
    }


    /// <summary>
    /// Restores the specified table to the specified point in time.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to restore.</param>
    /// <param name="timestamp">The time to which the table will be restored.</
param>
    /// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
    public async Task<string> RestoreTable(string keyspaceName, string tableName,
 string restoredTableName, DateTime timestamp)
    {
        var request = new RestoreTableRequest
        {
            RestoreTimestamp = timestamp,
            SourceKeyspaceName = keyspaceName,
            SourceTableName = tableName,
            TargetKeyspaceName = keyspaceName,
            TargetTableName = restoredTableName
        };

        var response = await _amazonKeyspaces.RestoreTableAsync(request);
        return response.RestoredTableARN;
    }


    /// <summary>
    /// Updates the movie table to add a boolean column named watched.
    /// </summary>
```

```
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to change.</param>
    /// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
    public async Task<string> UpdateTable(string keyspaceName, string tableName)
    {
        var newColumn = new ColumnDefinition { Name = "watched", Type =
 "boolean" };
        var request = new UpdateTableRequest
        {
            KeyspaceName = keyspaceName,
            TableName = tableName,
            AddColumns = new List<ColumnDefinition> { newColumn }
        };
        var response = await _amazonKeyspaces.UpdateTableAsync(request);
        return response.ResourceArn;
    }

}
```

```
using System.Net;
using Cassandra;

namespace KeyspacesScenario;

/// <summary>
/// Class to perform CRUD methods on an Amazon Keyspaces (for Apache Cassandra)
 database.
///
/// NOTE: This sample uses a plain text authenticator for example purposes only.
/// Recommended best practice is to use a SigV4 authentication plugin, if
 available.
/// </summary>
public class CassandraWrapper
{
    private readonly IConfiguration _configuration;
    private readonly string _localPathToFile;
    private const string _certLocation = "https://certs.secureserver.net/
repository/sf-class2-root.crt";
    private const string _certFileName = "sf-class2-root.crt";
    private readonly X509Certificate2Collection _certCollection;
    private X509Certificate2 _amazoncert;
```

```
    private Cluster _cluster;

    // User name and password for the service.
    private string _userName = null!;
    private string _pwd = null!;

    public CassandraWrapper()
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        _localPathToFile = Path.GetTempPath();

        // Get the Starfield digital certificate and save it locally.
        var client = new WebClient();
        client.DownloadFile(_certLocation, $"{_localPathToFile}/
{_certFileName}");

        //var httpClient = new HttpClient();
        //var httpResult = httpClient.Get(fileUrl);
        //using var resultStream = await httpResult.Content.ReadAsStreamAsync();
        //using var fileStream = File.Create(pathToSave);
        //resultStream.CopyTo(fileStream);

        _certCollection = new X509Certificate2Collection();
        _amazoncert = new X509Certificate2($"{_localPathToFile}/
{_certFileName}");

        // Get the user name and password stored in the configuration file.
        _userName = _configuration["UserName"]!;
        _pwd = _configuration["Password"]!;

        // For a list of Service Endpoints for Amazon Keyspaces, see:
        // https://docs.aws.amazon.com/keyspaces/latest/devguide/
programmatic.endpoints.html
        var awsEndpoint = _configuration["ServiceEndpoint"];

        _cluster = Cluster.Builder()
            .AddContactPoints(awsEndpoint)
            .WithPort(9142)
```

```
            .WithAuthProvider(new PlainTextAuthProvider(_userName, _pwd))
            .WithSSL(new SSLOptions().SetCertificateCollection(_certCollection))
            .WithQueryOptions(
                new QueryOptions()
                    .SetConsistencyLevel(ConsistencyLevel.LocalQuorum)
                    .SetSerialConsistencyLevel(ConsistencyLevel.LocalSerial))
            .Build();
    }

    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the Apache Cassandra table.
    /// </summary>
    /// <param name="movieFileName">The full path to the JSON file.</param>
    /// <returns>A list of movie objects.</returns>
    public List<Movie> ImportMoviesFromJson(string movieFileName, int numToImport
= 0)
    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();

        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

        // If numToImport = 0, return all movies in the collection.
        if (numToImport == 0)
        {
            // Now return the entire list of movies.
            return allMovies;
        }
        else
        {
            // Now return the first numToImport entries.
            return allMovies.GetRange(0, numToImport);
        }
    }

    /// <summary>
    /// Insert movies into the movie table.
    /// </summary>
```

```
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="movieTableName">The Amazon Keyspaces table.</param>
    /// <param name="movieFilePath">The path to the resource file containing
    /// movie data to insert into the table.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> InsertIntoMovieTable(string keyspaceName, string
movieTableName, string movieFilePath, int numToImport = 20)
    {
        // Get some movie data from the movies.json file
        var movies = ImportMoviesFromJson(movieFilePath, numToImport);

        var session = _cluster.Connect(keyspaceName);

        string insertCql;

        RowSet rs;

        // Now we insert the numToImport movies into the table.
        foreach (var movie in movies)
        {
            // Escape single quote characters in the plot.
            insertCql = $"INSERT INTO {keyspaceName}.{movieTableName}
(title, year, release_date, plot) values($${movie.Title}$$, {movie.Year},
'{movie.Info.Release_Date.ToString("yyyy-MM-dd")}', $${movie.Info.Plot}$$)";
            rs = await session.ExecuteAsync(new SimpleStatement(insertCql));
        }

        return true;
    }

    /// <summary>
    /// Gets all of the movies in the movies table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <returns>A list of row objects containing movie data.</returns>
    public async Task<List<Row>> GetMovies(string keyspaceName, string tableName)
    {
        var session = _cluster.Connect();
        RowSet rs;
        try
        {
            rs = await session.ExecuteAsync(new SimpleStatement($"SELECT * FROM
{keyspaceName}.{tableName}"));
```

```
            // Extract the row data from the returned RowSet.
            var rows = rs.GetRows().ToList();
            return rows;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return null!;
        }
    }

    /// <summary>
    /// Mark a movie in the movie table as watched.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="title">The title of the movie to mark as watched.</param>
    /// <param name="year">The year the movie was released.</param>
    /// <returns>A set of rows containing the changed data.</returns>
    public async Task<List<Row>> MarkMovieAsWatched(string keyspaceName, string
tableName, string title, int year)
    {
        var session = _cluster.Connect();
        string updateCql = $"UPDATE {keyspaceName}.{tableName} SET watched=true
WHERE title = $${title}$$ AND year = {year};";
        var rs = await session.ExecuteAsync(new SimpleStatement(updateCql));
        var rows = rs.GetRows().ToList();
        return rows;
    }

    /// <summary>
    /// Retrieve the movies in the movies table where watched is true.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <returns>A list of row objects containing information about movies
    /// where watched is true.</returns>
    public async Task<List<Row>> GetWatchedMovies(string keyspaceName, string
tableName)
    {
        var session = _cluster.Connect();
        RowSet rs;
        try
```

```
        {
            rs = await session.ExecuteAsync(new SimpleStatement($"SELECT
 title, year, plot FROM {keyspaceName}.{tableName} WHERE watched = true ALLOW
 FILTERING"));

            // Extract the row data from the returned RowSet.
            var rows = rs.GetRows().ToList();
            return rows;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return null!;
        }
    }
}
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の以下のトピックを参照してください。
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Before running this Java code example, you must create a
 * Java keystore (JKS) file and place it in your project's resources folder.
 *
 * This file is a secure file format used to hold certificate information for
 * Java applications. This is required to make a connection to Amazon Keyspaces.
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/keyspaces/latest/devguide/using_java_driver.html
 *
 * This Java example performs the following tasks:
 *
 * 1. Create a keyspace.
 * 2. Check for keyspace existence.
 * 3. List keyspaces using a paginator.
 * 4. Create a table with a simple movie data schema and enable point-in-time
 * recovery.
 * 5. Check for the table to be in an Active state.
 * 6. List all tables in the keyspace.
 * 7. Use a Cassandra driver to insert some records into the Movie table.
 * 8. Get all records from the Movie table.
 * 9. Get a specific Movie.
 * 10. Get a UTC timestamp for the current time.
 * 11. Update the table schema to add a ‘watched’ Boolean column.
```

```
 * 12. Update an item as watched.
 * 13. Query for items with watched = True.
 * 14. Restore the table back to the previous state using the timestamp.
 * 15. Check for completion of the restore action.
 * 16. Delete the table.
 * 17. Confirm that both tables are deleted.
 * 18. Delete the keyspace.
 */

public class ScenarioKeyspaces {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    /*
     * Usage:
     * fileName - The name of the JSON file that contains movie data. (Get this
 file
     * from the GitHub repo at resources/sample_file.)
     * keyspaceName - The name of the keyspace to create.
     */
    public static void main(String[] args) throws InterruptedException,
 IOException {
        String fileName = "<Replace with the JSON file that contains movie
 data>";
        String keyspaceName = "<Replace with the name of the keyspace to
 create>";
        String titleUpdate = "The Family";
        int yearUpdate = 2013;
        String tableName = "Movie";
        String tableNameRestore = "MovieRestore";
        Region region = Region.US_EAST_1;
        KeyspacesClient keyClient = KeyspacesClient.builder()
                .region(region)
                .build();

        DriverConfigLoader loader =
 DriverConfigLoader.fromClasspath("application.conf");
        CqlSession session = CqlSession.builder()
                .withConfigLoader(loader)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Keyspaces example scenario.");
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("1. Create a keyspace.");
        createKeySpace(keyClient, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        Thread.sleep(5000);
        System.out.println("2. Check for keyspace existence.");
        checkKeyspaceExistence(keyClient, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. List keyspaces using a paginator.");
        listKeyspacesPaginator(keyClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Create a table with a simple movie data schema and
enable point-in-time recovery.");
        createTable(keyClient, keyspaceName, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Check for the table to be in an Active state.");
        Thread.sleep(6000);
        checkTable(keyClient, keyspaceName, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. List all tables in the keyspace.");
        listTables(keyClient, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Use a Cassandra driver to insert some records into
the Movie table.");
        Thread.sleep(6000);
        loadData(session, fileName, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Get all records from the Movie table.");
        getMovieData(session, keyspaceName);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Get a specific Movie.");
        getSpecificMovie(session, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Get a UTC timestamp for the current time.");
        ZonedDateTime utc = ZonedDateTime.now(ZoneOffset.UTC);
        System.out.println("DATETIME = " + Date.from(utc.toInstant()));
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Update the table schema to add a watched Boolean
column.");
        updateTable(keyClient, keyspaceName, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Update an item as watched.");
        Thread.sleep(10000); // Wait 10 secs for the update.
        updateRecord(session, keyspaceName, titleUpdate, yearUpdate);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Query for items with watched = True.");
        getWatchedData(session, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Restore the table back to the previous state
using the timestamp.");
        System.out.println("Note that the restore operation can take up to 20
minutes.");
        restoreTable(keyClient, keyspaceName, utc);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Check for completion of the restore action.");
        Thread.sleep(5000);
        checkRestoredTable(keyClient, keyspaceName, "MovieRestore");
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("16. Delete both tables.");
        deleteTable(keyClient, keyspaceName, tableName);
        deleteTable(keyClient, keyspaceName, tableNameRestore);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("17. Confirm that both tables are deleted.");
        checkTableDelete(keyClient, keyspaceName, tableName);
        checkTableDelete(keyClient, keyspaceName, tableNameRestore);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("18. Delete the keyspace.");
        deleteKeyspace(keyClient, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The scenario has completed successfully.");
        System.out.println(DASHES);
    }

    public static void deleteKeyspace(KeyspacesClient keyClient, String
keyspaceName) {
        try {
            DeleteKeyspaceRequest deleteKeyspaceRequest =
DeleteKeyspaceRequest.builder()
                    .keyspaceName(keyspaceName)
                    .build();

            keyClient.deleteKeyspace(deleteKeyspaceRequest);

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void checkTableDelete(KeyspacesClient keyClient, String
keyspaceName, String tableName)
            throws InterruptedException {
        try {
            String status;
            GetTableResponse response;
```

```
            GetTableRequest tableRequest = GetTableRequest.builder()
                    .keyspaceName(keyspaceName)
                    .tableName(tableName)
                    .build();

            // Keep looping until table cannot be found and a
ResourceNotFoundException is
            // thrown.
            while (true) {
                response = keyClient.getTable(tableRequest);
                status = response.statusAsString();
                System.out.println(". The table status is " + status);
                Thread.sleep(500);
            }

        } catch (ResourceNotFoundException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
        System.out.println("The table is deleted");
    }

    public static void deleteTable(KeyspacesClient keyClient, String
keyspaceName, String tableName) {
        try {
            DeleteTableRequest tableRequest = DeleteTableRequest.builder()
                    .keyspaceName(keyspaceName)
                    .tableName(tableName)
                    .build();

            keyClient.deleteTable(tableRequest);

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void checkRestoredTable(KeyspacesClient keyClient, String
keyspaceName, String tableName)
            throws InterruptedException {
        try {
            boolean tableStatus = false;
            String status;
            GetTableResponse response = null;
```

```
            GetTableRequest tableRequest = GetTableRequest.builder()
                    .keyspaceName(keyspaceName)
                    .tableName(tableName)
                    .build();

            while (!tableStatus) {
                response = keyClient.getTable(tableRequest);
                status = response.statusAsString();
                System.out.println("The table status is " + status);

                if (status.compareTo("ACTIVE") == 0) {
                    tableStatus = true;
                }
                Thread.sleep(500);
            }

            List<ColumnDefinition> cols =
response.schemaDefinition().allColumns();
            for (ColumnDefinition def : cols) {
                System.out.println("The column name is " + def.name());
                System.out.println("The column type is " + def.type());
            }

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void restoreTable(KeyspacesClient keyClient, String
keyspaceName, ZonedDateTime utc) {
        try {
            Instant myTime = utc.toInstant();
            RestoreTableRequest restoreTableRequest =
RestoreTableRequest.builder()
                    .restoreTimestamp(myTime)
                    .sourceTableName("Movie")
                    .targetKeyspaceName(keyspaceName)
                    .targetTableName("MovieRestore")
                    .sourceKeyspaceName(keyspaceName)
                    .build();

            RestoreTableResponse response =
keyClient.restoreTable(restoreTableRequest);
```

```
            System.out.println("The ARN of the restored table is " +
response.restoredTableARN());

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void getWatchedData(CqlSession session, String keyspaceName) {
        ResultSet resultSet = session
                .execute("SELECT * FROM \"" + keyspaceName + "\".\"Movie\" WHERE
watched = true ALLOW FILTERING;");
        resultSet.forEach(item -> {
            System.out.println("The Movie title is " + item.getString("title"));
            System.out.println("The Movie year is " + item.getInt("year"));
            System.out.println("The plot is " + item.getString("plot"));
        });
    }

    public static void updateRecord(CqlSession session, String keySpace, String
titleUpdate, int yearUpdate) {
        String sqlStatement = "UPDATE \"" + keySpace
                + "\".\"Movie\" SET watched=true WHERE title = :k0 AND year
= :k1;";
        BatchStatementBuilder builder =
BatchStatement.builder(DefaultBatchType.UNLOGGED);
        builder.setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM);
        PreparedStatement preparedStatement = session.prepare(sqlStatement);
        builder.addStatement(preparedStatement.boundStatementBuilder()
                .setString("k0", titleUpdate)
                .setInt("k1", yearUpdate)
                .build());

        BatchStatement batchStatement = builder.build();
        session.execute(batchStatement);
    }

    public static void updateTable(KeyspacesClient keyClient, String keySpace,
String tableName) {
        try {
            ColumnDefinition def = ColumnDefinition.builder()
                    .name("watched")
                    .type("boolean")
```

```
                    .build();

            UpdateTableRequest tableRequest = UpdateTableRequest.builder()
                    .keyspaceName(keySpace)
                    .tableName(tableName)
                    .addColumns(def)
                    .build();

            keyClient.updateTable(tableRequest);

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void getSpecificMovie(CqlSession session, String keyspaceName)
{
        ResultSet resultSet = session.execute(
                "SELECT * FROM \"" + keyspaceName + "\".\"Movie\" WHERE title =
'The Family' ALLOW FILTERING ;");
        resultSet.forEach(item -> {
            System.out.println("The Movie title is " + item.getString("title"));
            System.out.println("The Movie year is " + item.getInt("year"));
            System.out.println("The plot is " + item.getString("plot"));
        });
    }

    // Get records from the Movie table.
    public static void getMovieData(CqlSession session, String keyspaceName) {
        ResultSet resultSet = session.execute("SELECT * FROM \"" + keyspaceName +
"\".\"Movie\";");
        resultSet.forEach(item -> {
            System.out.println("The Movie title is " + item.getString("title"));
            System.out.println("The Movie year is " + item.getInt("year"));
            System.out.println("The plot is " + item.getString("plot"));
        });
    }

    // Load data into the table.
    public static void loadData(CqlSession session, String fileName, String
keySpace) throws IOException {
        String sqlStatement = "INSERT INTO \"" + keySpace + "\".\"Movie\" (title,
year, plot) values (:k0, :k1, :k2)";
```

```
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
        ObjectNode currentNode;
        int t = 0;
        while (iter.hasNext()) {

            // Add 20 movies to the table.
            if (t == 20)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String plot = currentNode.path("info").path("plot").toString();

            // Insert the data into the Amazon Keyspaces table.
            BatchStatementBuilder builder =
BatchStatement.builder(DefaultBatchType.UNLOGGED);
            builder.setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM);
            PreparedStatement preparedStatement = session.prepare(sqlStatement);
            builder.addStatement(preparedStatement.boundStatementBuilder()
                    .setString("k0", title)
                    .setInt("k1", year)
                    .setString("k2", plot)
                    .build());

            BatchStatement batchStatement = builder.build();
            session.execute(batchStatement);
            t++;
        }

        System.out.println("You have added " + t + " records successfully!");
    }

    public static void listTables(KeyspacesClient keyClient, String keyspaceName)
{
        try {
            ListTablesRequest tablesRequest = ListTablesRequest.builder()
                    .keyspaceName(keyspaceName)
                    .build();
```

```
            ListTablesIterable listRes =
keyClient.listTablesPaginator(tablesRequest);
            listRes.stream()
                    .flatMap(r -> r.tables().stream())
                    .forEach(content -> System.out.println(" ARN: " +
content.resourceArn() +
                            " Table name: " + content.tableName()));

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void checkTable(KeyspacesClient keyClient, String keyspaceName,
String tableName)
            throws InterruptedException {
        try {
            boolean tableStatus = false;
            String status;
            GetTableResponse response = null;
            GetTableRequest tableRequest = GetTableRequest.builder()
                    .keyspaceName(keyspaceName)
                    .tableName(tableName)
                    .build();

            while (!tableStatus) {
                response = keyClient.getTable(tableRequest);
                status = response.statusAsString();
                System.out.println(". The table status is " + status);

                if (status.compareTo("ACTIVE") == 0) {
                    tableStatus = true;
                }
                Thread.sleep(500);
            }

            List<ColumnDefinition> cols =
response.schemaDefinition().allColumns();
            for (ColumnDefinition def : cols) {
                System.out.println("The column name is " + def.name());
                System.out.println("The column type is " + def.type());
            }
```

```
        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createTable(KeyspacesClient keyClient, String keySpace,
String tableName) {
        try {
            // Set the columns.
            ColumnDefinition defTitle = ColumnDefinition.builder()
                    .name("title")
                    .type("text")
                    .build();

            ColumnDefinition defYear = ColumnDefinition.builder()
                    .name("year")
                    .type("int")
                    .build();

            ColumnDefinition defReleaseDate = ColumnDefinition.builder()
                    .name("release_date")
                    .type("timestamp")
                    .build();

            ColumnDefinition defPlot = ColumnDefinition.builder()
                    .name("plot")
                    .type("text")
                    .build();

            List<ColumnDefinition> colList = new ArrayList<>();
            colList.add(defTitle);
            colList.add(defYear);
            colList.add(defReleaseDate);
            colList.add(defPlot);

            // Set the keys.
            PartitionKey yearKey = PartitionKey.builder()
                    .name("year")
                    .build();

            PartitionKey titleKey = PartitionKey.builder()
                    .name("title")
                    .build();
```

```
            List<PartitionKey> keyList = new ArrayList<>();
            keyList.add(yearKey);
            keyList.add(titleKey);

            SchemaDefinition schemaDefinition = SchemaDefinition.builder()
                    .partitionKeys(keyList)
                    .allColumns(colList)
                    .build();

            PointInTimeRecovery timeRecovery = PointInTimeRecovery.builder()
                    .status(PointInTimeRecoveryStatus.ENABLED)
                    .build();

            CreateTableRequest tableRequest = CreateTableRequest.builder()
                    .keyspaceName(keySpace)
                    .tableName(tableName)
                    .schemaDefinition(schemaDefinition)
                    .pointInTimeRecovery(timeRecovery)
                    .build();

            CreateTableResponse response = keyClient.createTable(tableRequest);
            System.out.println("The table ARN is " + response.resourceArn());

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void listKeyspacesPaginator(KeyspacesClient keyClient) {
        try {
            ListKeyspacesRequest keyspacesRequest =
ListKeyspacesRequest.builder()
                    .maxResults(10)
                    .build();

            ListKeyspacesIterable listRes =
keyClient.listKeyspacesPaginator(keyspacesRequest);
            listRes.stream()
                    .flatMap(r -> r.keyspaces().stream())
                    .forEach(content -> System.out.println(" Name: " +
content.keyspaceName()));
```

```
        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void checkKeyspaceExistence(KeyspacesClient keyClient, String
 keyspaceName) {
        try {
            GetKeyspaceRequest keyspaceRequest = GetKeyspaceRequest.builder()
                    .keyspaceName(keyspaceName)
                    .build();

            GetKeyspaceResponse response =
 keyClient.getKeyspace(keyspaceRequest);
            String name = response.keyspaceName();
            System.out.println("The " + name + " KeySpace is ready");

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createKeySpace(KeyspacesClient keyClient, String
 keyspaceName) {
        try {
            CreateKeyspaceRequest keyspaceRequest =
 CreateKeyspaceRequest.builder()
                    .keyspaceName(keyspaceName)
                    .build();

            CreateKeyspaceResponse response =
 keyClient.createKeyspace(keyspaceRequest);
            System.out.println("The ARN of the KeySpace is " +
 response.resourceArn());

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の以下のトピックを参照してください。
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/**
 Before running this Kotlin code example, set up your development environment,
 including your credentials.

 For more information, see the following documentation topic:

 https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

 This example uses a secure file format to hold certificate information for
 Kotlin applications. This is required to make a connection to Amazon Keyspaces.
 For more information, see the following documentation topic:
```

```
 https://docs.aws.amazon.com/keyspaces/latest/devguide/using_java_driver.html

 This Kotlin example performs the following tasks:

 1. Create a keyspace.
 2. Check for keyspace existence.
 3. List keyspaces using a paginator.
 4. Create a table with a simple movie data schema and enable point-in-time
 recovery.
 5. Check for the table to be in an Active state.
 6. List all tables in the keyspace.
 7. Use a Cassandra driver to insert some records into the Movie table.
 8. Get all records from the Movie table.
 9. Get a specific Movie.
 10. Get a UTC timestamp for the current time.
 11. Update the table schema to add a ‘watched’ Boolean column.
 12. Update an item as watched.
 13. Query for items with watched = True.
 14. Restore the table back to the previous state using the timestamp.
 15. Check for completion of the restore action.
 16. Delete the table.
 17. Confirm that both tables are deleted.
 18. Delete the keyspace.
 */

/*
   Usage:
     fileName - The name of the JSON file that contains movie data. (Get this
 file from the GitHub repo at resources/sample_file.)
     keyspaceName - The name of the keyspace to create.
 */
val DASHES: String = String(CharArray(80)).replace("\u0000", "-")

suspend fun main() {
    val fileName = "<Replace with the JSON file that contains movie data>"
    val keyspaceName = "<Replace with the name of the keyspace to create>"
    val titleUpdate = "The Family"
    val yearUpdate = 2013
    val tableName = "MovieKotlin"
    val tableNameRestore = "MovieRestore"

    val loader = DriverConfigLoader.fromClasspath("application.conf")
    val session =
```

```
        CqlSession
            .builder()
            .withConfigLoader(loader)
            .build()

    println(DASHES)
    println("Welcome to the Amazon Keyspaces example scenario.")
    println(DASHES)

    println(DASHES)
    println("1. Create a keyspace.")
    createKeySpace(keyspaceName)
    println(DASHES)

    println(DASHES)
    delay(5000)
    println("2. Check for keyspace existence.")
    checkKeyspaceExistence(keyspaceName)
    println(DASHES)

    println(DASHES)
    println("3. List keyspaces using a paginator.")
    listKeyspacesPaginator()
    println(DASHES)

    println(DASHES)
    println("4. Create a table with a simple movie data schema and enable point-
in-time recovery.")
    createTable(keyspaceName, tableName)
    println(DASHES)

    println(DASHES)
    println("5. Check for the table to be in an Active state.")
    delay(6000)
    checkTable(keyspaceName, tableName)
    println(DASHES)

    println(DASHES)
    println("6. List all tables in the keyspace.")
    listTables(keyspaceName)
    println(DASHES)

    println(DASHES)
```

```
    println("7. Use a Cassandra driver to insert some records into the Movie
table.")
    delay(6000)
    loadData(session, fileName, keyspaceName)
    println(DASHES)

    println(DASHES)
    println("8. Get all records from the Movie table.")
    getMovieData(session, keyspaceName)
    println(DASHES)

    println(DASHES)
    println("9. Get a specific Movie.")
    getSpecificMovie(session, keyspaceName)
    println(DASHES)

    println(DASHES)
    println("10. Get a UTC timestamp for the current time.")
    val utc = ZonedDateTime.now(ZoneOffset.UTC)
    println("DATETIME = ${Date.from(utc.toInstant())}")
    println(DASHES)

    println(DASHES)
    println("11. Update the table schema to add a watched Boolean column.")
    updateTable(keyspaceName, tableName)
    println(DASHES)

    println(DASHES)
    println("12. Update an item as watched.")
    delay(10000) // Wait 10 seconds for the update.
    updateRecord(session, keyspaceName, titleUpdate, yearUpdate)
    println(DASHES)

    println(DASHES)
    println("13. Query for items with watched = True.")
    getWatchedData(session, keyspaceName)
    println(DASHES)

    println(DASHES)
    println("14. Restore the table back to the previous state using the
timestamp.")
    println("Note that the restore operation can take up to 20 minutes.")
    restoreTable(keyspaceName, utc)
    println(DASHES)
```

```
    println(DASHES)
    println("15. Check for completion of the restore action.")
    delay(5000)
    checkRestoredTable(keyspaceName, "MovieRestore")
    println(DASHES)

    println(DASHES)
    println("16. Delete both tables.")
    deleteTable(keyspaceName, tableName)
    deleteTable(keyspaceName, tableNameRestore)
    println(DASHES)

    println(DASHES)
    println("17. Confirm that both tables are deleted.")
    checkTableDelete(keyspaceName, tableName)
    checkTableDelete(keyspaceName, tableNameRestore)
    println(DASHES)

    println(DASHES)
    println("18. Delete the keyspace.")
    deleteKeyspace(keyspaceName)
    println(DASHES)

    println(DASHES)
    println("The scenario has completed successfully.")
    println(DASHES)
}

suspend fun deleteKeyspace(keyspaceNameVal: String?) {
    val deleteKeyspaceRequest =
        DeleteKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        keyClient.deleteKeyspace(deleteKeyspaceRequest)
    }
}

suspend fun checkTableDelete(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
```

```
    var status: String
    var response: GetTableResponse
    val tableRequest =
        GetTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }

    try {
        KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient
 ->
            // Keep looping until the table cannot be found and a
 ResourceNotFoundException is thrown.
            while (true) {
                response = keyClient.getTable(tableRequest)
                status = response.status.toString()
                println(". The table status is $status")
                delay(500)
            }
        }
    } catch (e: ResourceNotFoundException) {
        println(e.message)
    }
    println("The table is deleted")
}

suspend fun deleteTable(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
    val tableRequest =
        DeleteTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        keyClient.deleteTable(tableRequest)
    }
}

suspend fun checkRestoredTable(
    keyspaceNameVal: String?,
    tableNameVal: String?,
```

```
) {
    var tableStatus = false
    var status: String
    var response: GetTableResponse? = null

    val tableRequest =
        GetTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        while (!tableStatus) {
            response = keyClient.getTable(tableRequest)
            status = response!!.status.toString()
            println("The table status is $status")

            if (status.compareTo("ACTIVE") == 0) {
                tableStatus = true
            }
            delay(500)
        }

        val cols = response!!.schemaDefinition?.allColumns
        if (cols != null) {
            for (def in cols) {
                println("The column name is ${def.name}")
                println("The column type is ${def.type}")
            }
        }
    }
}

suspend fun restoreTable(
    keyspaceName: String?,
    utc: ZonedDateTime,
) {
    // Create an aws.smithy.kotlin.runtime.time.Instant value.
    val timeStamp =
        aws.smithy.kotlin.runtime.time
            .Instant(utc.toInstant())
    val restoreTableRequest =
        RestoreTableRequest {
            restoreTimestamp = timeStamp
```

```
            sourceTableName = "MovieKotlin"
            targetKeyspaceName = keyspaceName
            targetTableName = "MovieRestore"
            sourceKeyspaceName = keyspaceName
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.restoreTable(restoreTableRequest)
        println("The ARN of the restored table is ${response.restoredTableArn}")
    }
}

fun getWatchedData(
    session: CqlSession,
    keyspaceName: String,
) {
    val resultSet = session.execute("SELECT * FROM \"$keyspaceName\".
\"MovieKotlin\" WHERE watched = true ALLOW FILTERING;")
    resultSet.forEach { item: Row ->
        println("The Movie title is ${item.getString("title")}")
        println("The Movie year is ${item.getInt("year")}")
        println("The plot is ${item.getString("plot")}")
    }
}

fun updateRecord(
    session: CqlSession,
    keySpace: String,
    titleUpdate: String?,
    yearUpdate: Int,
) {
    val sqlStatement =
        "UPDATE \"$keySpace\".\"MovieKotlin\" SET watched=true WHERE title = :k0
 AND year = :k1;"
    val builder = BatchStatement.builder(DefaultBatchType.UNLOGGED)
    builder.setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM)
    val preparedStatement = session.prepare(sqlStatement)
    builder.addStatement(
        preparedStatement
            .boundStatementBuilder()
            .setString("k0", titleUpdate)
            .setInt("k1", yearUpdate)
            .build(),
    )
```

```
    val batchStatement = builder.build()
    session.execute(batchStatement)
}

suspend fun updateTable(
    keySpace: String?,
    tableNameVal: String?,
) {
    val def =
        ColumnDefinition {
            name = "watched"
            type = "boolean"
        }

    val tableRequest =
        UpdateTableRequest {
            keyspaceName = keySpace
            tableName = tableNameVal
            addColumns = listOf(def)
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        keyClient.updateTable(tableRequest)
    }
}

fun getSpecificMovie(
    session: CqlSession,
    keyspaceName: String,
) {
    val resultSet =
        session.execute("SELECT * FROM \"$keyspaceName\".\"MovieKotlin\" WHERE
 title = 'The Family' ALLOW FILTERING ;")

    resultSet.forEach { item: Row ->
        println("The Movie title is ${item.getString("title")}")
        println("The Movie year is ${item.getInt("year")}")
        println("The plot is ${item.getString("plot")}")
    }
}

// Get records from the Movie table.
fun getMovieData(
    session: CqlSession,
```

```
    keyspaceName: String,
) {
    val resultSet = session.execute("SELECT * FROM \"$keyspaceName\".
\"MovieKotlin\";")
    resultSet.forEach { item: Row ->
        println("The Movie title is ${item.getString("title")}")
        println("The Movie year is ${item.getInt("year")}")
        println("The plot is ${item.getString("plot")}")
    }
}

// Load data into the table.
fun loadData(
    session: CqlSession,
    fileName: String,
    keySpace: String,
) {
    val sqlStatement =
        "INSERT INTO \"$keySpace\".\"MovieKotlin\" (title, year, plot) values
 (:k0, :k1, :k2)"
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()

        // Insert the data into the Amazon Keyspaces table.
        val builder = BatchStatement.builder(DefaultBatchType.UNLOGGED)
        builder.setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM)
        val preparedStatement: PreparedStatement = session.prepare(sqlStatement)
        builder.addStatement(
            preparedStatement
                .boundStatementBuilder()
                .setString("k0", title)
```

```
                .setInt("k1", year)
                .setString("k2", info)
                .build(),
        )

        val batchStatement = builder.build()
        session.execute(batchStatement)
        t++
    }
}

suspend fun listTables(keyspaceNameVal: String?) {
    val tablesRequest =
        ListTablesRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        keyClient
            .listTablesPaginated(tablesRequest)
            .transform { it.tables?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println(" ARN: ${obj.resourceArn} Table name: ${obj.tableName}")
            }
    }
}

suspend fun checkTable(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
    var tableStatus = false
    var status: String
    var response: GetTableResponse? = null

    val tableRequest =
        GetTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }
    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        while (!tableStatus) {
            response = keyClient.getTable(tableRequest)
            status = response!!.status.toString()
```

```
            println(". The table status is $status")
            if (status.compareTo("ACTIVE") == 0) {
                tableStatus = true
            }
            delay(500)
        }
        val cols: List<ColumnDefinition>? =
 response!!.schemaDefinition?.allColumns
        if (cols != null) {
            for (def in cols) {
                println("The column name is ${def.name}")
                println("The column type is ${def.type}")
            }
        }
    }
}

suspend fun createTable(
    keySpaceVal: String?,
    tableNameVal: String?,
) {
    // Set the columns.
    val defTitle =
        ColumnDefinition {
            name = "title"
            type = "text"
        }

    val defYear =
        ColumnDefinition {
            name = "year"
            type = "int"
        }

    val defReleaseDate =
        ColumnDefinition {
            name = "release_date"
            type = "timestamp"
        }

    val defPlot =
        ColumnDefinition {
            name = "plot"
            type = "text"
```

```
        }

    val colList = ArrayList<ColumnDefinition>()
    colList.add(defTitle)
    colList.add(defYear)
    colList.add(defReleaseDate)
    colList.add(defPlot)

    // Set the keys.
    val yearKey =
        PartitionKey {
            name = "year"
        }

    val titleKey =
        PartitionKey {
            name = "title"
        }

    val keyList = ArrayList<PartitionKey>()
    keyList.add(yearKey)
    keyList.add(titleKey)

    val schemaDefinitionOb =
        SchemaDefinition {
            partitionKeys = keyList
            allColumns = colList
        }

    val timeRecovery =
        PointInTimeRecovery {
            status = PointInTimeRecoveryStatus.Enabled
        }

    val tableRequest =
        CreateTableRequest {
            keyspaceName = keySpaceVal
            tableName = tableNameVal
            schemaDefinition = schemaDefinitionOb
            pointInTimeRecovery = timeRecovery
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.createTable(tableRequest)
```

```
        println("The table ARN is ${response.resourceArn}")
    }
}

suspend fun listKeyspacesPaginator() {
    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        keyClient
            .listKeyspacesPaginated(ListKeyspacesRequest {})
            .transform { it.keyspaces?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name: ${obj.keyspaceName}")
            }
    }
}

suspend fun checkKeyspaceExistence(keyspaceNameVal: String?) {
    val keyspaceRequest =
        GetKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }
    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response: GetKeyspaceResponse =
 keyClient.getKeyspace(keyspaceRequest)
        val name = response.keyspaceName
        println("The $name KeySpace is ready")
    }
}

suspend fun createKeySpace(keyspaceNameVal: String) {
    val keyspaceRequest =
        CreateKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.createKeyspace(keyspaceRequest)
        println("The ARN of the KeySpace is ${response.resourceArn}")
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の以下のトピックを参照してください。

- CreateKeyspace
- CreateTable
- DeleteKeyspace
- DeleteTable
- GetKeyspace
- GetTable
- ListKeyspaces
- ListTables
- RestoreTable
- UpdateTable

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
class KeyspaceScenario:
    """Runs an interactive scenario that shows how to get started using Amazon
 Keyspaces."""

    def __init__(self, ks_wrapper):
        """
        :param ks_wrapper: An object that wraps Amazon Keyspace actions.
        """
        self.ks_wrapper = ks_wrapper

    @demo_func
    def create_keyspace(self):
        """
        1. Creates a keyspace.
        2. Lists up to 10 keyspaces in your account.
```

```
        """
        print("Let's create a keyspace.")
        ks_name = q.ask(
            "Enter a name for your new keyspace.\nThe name can contain only
letters, "
            "numbers and underscores: ",
            q.non_empty,
        )
        if self.ks_wrapper.exists_keyspace(ks_name):
            print(f"A keyspace named {ks_name} exists.")
        else:
            ks_arn = self.ks_wrapper.create_keyspace(ks_name)
            ks_exists = False
            while not ks_exists:
                wait(3)
                ks_exists = self.ks_wrapper.exists_keyspace(ks_name)
            print(f"Created a new keyspace.\n\t{ks_arn}.")
        print("The first 10 keyspaces in your account are:\n")
        self.ks_wrapper.list_keyspaces(10)

    @demo_func
    def create_table(self):
        """
        1. Creates a table in the keyspace. The table is configured with a schema
to hold
           movie data and has point-in-time recovery enabled.
        2. Waits for the table to be in an active state.
        3. Displays schema information for the table.
        4. Lists tables in the keyspace.
        """
        print("Let's create a table for movies in your keyspace.")
        table_name = q.ask("Enter a name for your table: ", q.non_empty)
        table = self.ks_wrapper.get_table(table_name)
        if table is not None:
            print(
                f"A table named {table_name} already exists in keyspace "
                f"{self.ks_wrapper.ks_name}."
            )
        else:
            table_arn = self.ks_wrapper.create_table(table_name)
            print(f"Created table {table_name}:\n\t{table_arn}")
            table = {"status": None}
            print("Waiting for your table to be ready...")
            while table["status"] != "ACTIVE":
```

```
                wait(5)
                table = self.ks_wrapper.get_table(table_name)
        print(f"Your table is {table['status']}. Its schema is:")
        pp(table["schemaDefinition"])
        print("\nThe tables in your keyspace are:\n")
        self.ks_wrapper.list_tables()

    @demo_func
    def ensure_tls_cert(self):
        """
        Ensures you have a TLS certificate available to use to secure the
connection
        to the keyspace. This function downloads a default certificate or lets
you
        specify your own.
        """
        print("To connect to your keyspace, you must have a TLS certificate.")
        print("Checking for TLS certificate...")
        cert_path = os.path.join(
            os.path.dirname(__file__), QueryManager.DEFAULT_CERT_FILE
        )
        if not os.path.exists(cert_path):
            cert_choice = q.ask(
                f"Press enter to download a certificate from
{QueryManager.CERT_URL} "
                f"or enter the full path to the certificate you want to use: "
            )
            if cert_choice:
                cert_path = cert_choice
            else:
                cert = requests.get(QueryManager.CERT_URL).text
                with open(cert_path, "w") as cert_file:
                    cert_file.write(cert)
        else:
            q.ask(f"Certificate {cert_path} found. Press Enter to continue.")
        print(
            f"Certificate {cert_path} will be used to secure the connection to
your keyspace."
        )
        return cert_path

    @demo_func
    def query_table(self, qm, movie_file):
        """
```

```
        1. Adds movies to the table from a sample movie data file.
        2. Gets a list of movies from the table and lets you select one.
        3. Displays more information about the selected movie.
        """
        qm.add_movies(self.ks_wrapper.table_name, movie_file)
        movies = qm.get_movies(self.ks_wrapper.table_name)
        print(f"Added {len(movies)} movies to the table:")
        sel = q.choose("Pick one to learn more about it: ", [m.title for m in
movies])
        movie_choice = qm.get_movie(
            self.ks_wrapper.table_name, movies[sel].title, movies[sel].year
        )
        print(movie_choice.title)
        print(f"\tReleased: {movie_choice.release_date}")
        print(f"\tPlot: {movie_choice.plot}")

    @demo_func
    def update_and_restore_table(self, qm):
        """
        1. Updates the table by adding a column to track watched movies.
        2. Marks some of the movies as watched.
        3. Gets the list of watched movies from the table.
        4. Restores to a movies_restored table at a previous point in time.
        5. Gets the list of movies from the restored table.
        """
        print("Let's add a column to record which movies you've watched.")
        pre_update_timestamp = datetime.utcnow()
        print(
            f"Recorded the current UTC time of {pre_update_timestamp} so we can
restore the table later."
        )
        self.ks_wrapper.update_table()
        print("Waiting for your table to update...")
        table = {"status": "UPDATING"}
        while table["status"] != "ACTIVE":
            wait(5)
            table = self.ks_wrapper.get_table(self.ks_wrapper.table_name)
        print("Column 'watched' added to table.")
        q.ask(
            "Let's mark some of the movies as watched. Press Enter when you're
ready.\n"
        )
        movies = qm.get_movies(self.ks_wrapper.table_name)
        for movie in movies[:10]:
```

```
            qm.watched_movie(self.ks_wrapper.table_name, movie.title, movie.year)
            print(f"Marked {movie.title} as watched.")
        movies = qm.get_movies(self.ks_wrapper.table_name, watched=True)
        print("-" * 88)
        print("The watched movies in our table are:\n")
        for movie in movies:
            print(movie.title)
        print("-" * 88)
        if q.ask(
            "Do you want to restore the table to the way it was before all of
these\n"
            "updates? Keep in mind, this can take up to 20 minutes. (y/n) ",
            q.is_yesno,
        ):
            starting_table_name = self.ks_wrapper.table_name
            table_name_restored =
self.ks_wrapper.restore_table(pre_update_timestamp)
            table = {"status": "RESTORING"}
            while table["status"] != "ACTIVE":
                wait(10)
                table = self.ks_wrapper.get_table(table_name_restored)
            print(
                f"Restored {starting_table_name} to {table_name_restored} "
                f"at a point in time of {pre_update_timestamp}."
            )
            movies = qm.get_movies(table_name_restored)
            print("Now the movies in our table are:")
            for movie in movies:
                print(movie.title)

    def cleanup(self, cert_path):
        """
        1. Deletes the table and waits for it to be removed.
        2. Deletes the keyspace.

        :param cert_path: The path of the TLS certificate used in the demo. If
the
                          certificate was downloaded during the demo, it is
removed.
        """
        if q.ask(
            f"Do you want to delete your {self.ks_wrapper.table_name} table and "
            f"{self.ks_wrapper.ks_name} keyspace? (y/n) ",
            q.is_yesno,
```

```
        ):
            table_name = self.ks_wrapper.table_name
            self.ks_wrapper.delete_table()
            table = self.ks_wrapper.get_table(table_name)
            print("Waiting for the table to be deleted.")
            while table is not None:
                wait(5)
                table = self.ks_wrapper.get_table(table_name)
            print("Table deleted.")
            self.ks_wrapper.delete_keyspace()
            print(
                "Keyspace deleted. If you chose to restore your table during the
"
                "demo, the original table is also deleted."
            )
            if cert_path == os.path.join(
                os.path.dirname(__file__), QueryManager.DEFAULT_CERT_FILE
            ) and os.path.exists(cert_path):
                os.remove(cert_path)
                print("Removed certificate that was downloaded for this demo.")

    def run_scenario(self):
        logging.basicConfig(level=logging.INFO, format="%(levelname)s:
%(message)s")

        print("-" * 88)
        print("Welcome to the Amazon Keyspaces (for Apache Cassandra) demo.")
        print("-" * 88)

        self.create_keyspace()
        self.create_table()
        cert_file_path = self.ensure_tls_cert()
        # Use a context manager to ensure the connection to the keyspace is
closed.
        with QueryManager(
            cert_file_path, boto3.DEFAULT_SESSION, self.ks_wrapper.ks_name
        ) as qm:
            self.query_table(qm, "../../../resources/sample_files/movies.json")
            self.update_and_restore_table(qm)
        self.cleanup(cert_file_path)

        print("\nThanks for watching!")
        print("-" * 88)
```

```
if __name__ == "__main__":
    try:
        scenario = KeyspaceScenario(KeyspaceWrapper.from_client())
        scenario.run_scenario()
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

キースペースとテーブルアクションをラップするクラスを定義します。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def create_keyspace(self, name):
        """
        Creates a keyspace.

        :param name: The name to give the keyspace.
        :return: The Amazon Resource Name (ARN) of the new keyspace.
        """
        try:
            response = self.keyspaces_client.create_keyspace(keyspaceName=name)
            self.ks_name = name
            self.ks_arn = response["resourceArn"]
        except ClientError as err:
            logger.error(
```

```
                "Couldn't create %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.ks_arn


    def exists_keyspace(self, name):
        """
        Checks whether a keyspace exists.

        :param name: The name of the keyspace to look up.
        :return: True when the keyspace exists. Otherwise, False.
        """
        try:
            response = self.keyspaces_client.get_keyspace(keyspaceName=name)
            self.ks_name = response["keyspaceName"]
            self.ks_arn = response["resourceArn"]
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Keyspace %s does not exist.", name)
                exists = False
            else:
                logger.error(
                    "Couldn't verify %s exists. Here's why: %s: %s",
                    name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return exists


    def list_keyspaces(self, limit):
        """
        Lists the keyspaces in your account.

        :param limit: The maximum number of keyspaces to list.
        """
        try:
```

```
            ks_paginator = self.keyspaces_client.get_paginator("list_keyspaces")
            for page in ks_paginator.paginate(PaginationConfig={"MaxItems":
limit}):
                for ks in page["keyspaces"]:
                    print(ks["keyspaceName"])
                    print(f"\t{ks['resourceArn']}")
        except ClientError as err:
            logger.error(
                "Couldn't list keyspaces. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def create_table(self, table_name):
        """
        Creates a table in the  keyspace.
        The table is created with a schema for storing movie data
        and has point-in-time recovery enabled.

        :param table_name: The name to give the table.
        :return: The ARN of the new table.
        """
        try:
            response = self.keyspaces_client.create_table(
                keyspaceName=self.ks_name,
                tableName=table_name,
                schemaDefinition={
                    "allColumns": [
                        {"name": "title", "type": "text"},
                        {"name": "year", "type": "int"},
                        {"name": "release_date", "type": "timestamp"},
                        {"name": "plot", "type": "text"},
                    ],
                    "partitionKeys": [{"name": "year"}, {"name": "title"}],
                },
                pointInTimeRecovery={"status": "ENABLED"},
            )
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
```

```
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["resourceArn"]


    def get_table(self, table_name):
        """
        Gets data about a table in the keyspace.

        :param table_name: The name of the table to look up.
        :return: Data about the table.
        """
        try:
            response = self.keyspaces_client.get_table(
                keyspaceName=self.ks_name, tableName=table_name
            )
            self.table_name = table_name
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Table %s does not exist.", table_name)
                self.table_name = None
                response = None
            else:
                logger.error(
                    "Couldn't verify %s exists. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return response


    def list_tables(self):
        """
        Lists the tables in the keyspace.
        """
        try:
            table_paginator = self.keyspaces_client.get_paginator("list_tables")
            for page in table_paginator.paginate(keyspaceName=self.ks_name):
                for table in page["tables"]:
                    print(table["tableName"])
```

```
                    print(f"\t{table['resourceArn']}")
        except ClientError as err:
            logger.error(
                "Couldn't list tables in keyspace %s. Here's why: %s: %s",
                self.ks_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def update_table(self):
        """
        Updates the schema of the table.

        This example updates a table of movie data by adding a new column
        that tracks whether the movie has been watched.
        """
        try:
            self.keyspaces_client.update_table(
                keyspaceName=self.ks_name,
                tableName=self.table_name,
                addColumns=[{"name": "watched", "type": "boolean"}],
            )
        except ClientError as err:
            logger.error(
                "Couldn't update table %s. Here's why: %s: %s",
                self.table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def restore_table(self, restore_timestamp):
        """
        Restores the table to a previous point in time. The table is restored
        to a new table in the same keyspace.

        :param restore_timestamp: The point in time to restore the table. This
time
                                  must be in UTC format.
        :return: The name of the restored table.
        """
```

```
        try:
            restored_table_name = f"{self.table_name}_restored"
            self.keyspaces_client.restore_table(
                sourceKeyspaceName=self.ks_name,
                sourceTableName=self.table_name,
                targetKeyspaceName=self.ks_name,
                targetTableName=restored_table_name,
                restoreTimestamp=restore_timestamp,
            )
        except ClientError as err:
            logger.error(
                "Couldn't restore table %s. Here's why: %s: %s",
                restore_timestamp,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return restored_table_name


    def delete_table(self):
        """
        Deletes the table from the keyspace.
        """
        try:
            self.keyspaces_client.delete_table(
                keyspaceName=self.ks_name, tableName=self.table_name
            )
            self.table_name = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table %s. Here's why: %s: %s",
                self.table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_keyspace(self):
        """
        Deletes the keyspace.
        """
```

```
        try:
            self.keyspaces_client.delete_keyspace(keyspaceName=self.ks_name)
            self.ks_name = None
        except ClientError as err:
            logger.error(
                "Couldn't delete keyspace %s. Here's why: %s: %s",
                self.ks_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

キースペースへの TLS 接続を作成し、SigV4 で認証し、キースペース内のテーブルに CQL クエリーを送信するクラスを定義します。

```
class QueryManager:
    """
    Manages queries to an Amazon Keyspaces (for Apache Cassandra) keyspace.
    Queries are secured by TLS and authenticated by using the Signature V4
 (SigV4)
    AWS signing protocol. This is more secure than sending username and password
    with a plain-text authentication provider.

    This example downloads a default certificate to secure TLS, or lets you
 specify
    your own.

    This example uses a table of movie data to demonstrate basic queries.
    """

    DEFAULT_CERT_FILE = "sf-class2-root.crt"
    CERT_URL = f"https://certs.secureserver.net/repository/sf-class2-root.crt"

    def __init__(self, cert_file_path, boto_session, keyspace_name):
        """
        :param cert_file_path: The path and file name of the certificate used for
 TLS.
        :param boto_session: A Boto3 session. This is used to acquire your AWS
 credentials.
```

```
        :param keyspace_name: The name of the keyspace to connect.
        """
        self.cert_file_path = cert_file_path
        self.boto_session = boto_session
        self.ks_name = keyspace_name
        self.cluster = None
        self.session = None

    def __enter__(self):
        """
        Creates a session connection to the keyspace that is secured by TLS and
        authenticated by SigV4.
        """
        ssl_context = SSLContext(PROTOCOL_TLSv1_2)
        ssl_context.load_verify_locations(self.cert_file_path)
        ssl_context.verify_mode = CERT_REQUIRED
        auth_provider = SigV4AuthProvider(self.boto_session)
        contact_point = f"cassandra.
{self.boto_session.region_name}.amazonaws.com"
        exec_profile = ExecutionProfile(
            consistency_level=ConsistencyLevel.LOCAL_QUORUM,
            load_balancing_policy=DCAwareRoundRobinPolicy(),
        )
        self.cluster = Cluster(
            [contact_point],
            ssl_context=ssl_context,
            auth_provider=auth_provider,
            port=9142,
            execution_profiles={EXEC_PROFILE_DEFAULT: exec_profile},
            protocol_version=4,
        )
        self.cluster.__enter__()
        self.session = self.cluster.connect(self.ks_name)
        return self

    def __exit__(self, *args):
        """
        Exits the cluster. This shuts down all existing session connections.
        """
        self.cluster.__exit__(*args)

    def add_movies(self, table_name, movie_file_path):
        """
        Gets movies from a JSON file and adds them to a table in the keyspace.
```

```
        :param table_name: The name of the table.
        :param movie_file_path: The path and file name of a JSON file that
 contains movie data.
        """
        with open(movie_file_path, "r") as movie_file:
            movies = json.loads(movie_file.read())
        stmt = self.session.prepare(
            f"INSERT INTO {table_name} (year, title, release_date, plot) VALUES
 (?, ?, ?, ?);"
        )
        for movie in movies[:20]:
            self.session.execute(
                stmt,
                parameters=[
                    movie["year"],
                    movie["title"],
                    date.fromisoformat(movie["info"]
["release_date"].partition("T")[0]),
                    movie["info"]["plot"],
                ],
            )

    def get_movies(self, table_name, watched=None):
        """
        Gets the title and year of the full list of movies from the table.

        :param table_name: The name of the movie table.
        :param watched: When specified, the returned list of movies is filtered
 to
                        either movies that have been watched or movies that have
 not
                        been watched. Otherwise, all movies are returned.
        :return: A list of movies in the table.
        """
        if watched is None:
            stmt = SimpleStatement(f"SELECT title, year from {table_name}")
            params = None
        else:
            stmt = SimpleStatement(
                f"SELECT title, year from {table_name} WHERE watched = %s ALLOW
 FILTERING"
            )
            params = [watched]
```

```
        return self.session.execute(stmt, parameters=params).all()

    def get_movie(self, table_name, title, year):
        """
        Gets a single movie from the table, by title and year.

        :param table_name: The name of the movie table.
        :param title: The title of the movie.
        :param year: The year of the movie's release.
        :return: The requested movie.
        """
        return self.session.execute(
            SimpleStatement(
                f"SELECT * from {table_name} WHERE title = %s AND year = %s"
            ),
            parameters=[title, year],
        ).one()

    def watched_movie(self, table_name, title, year):
        """
        Updates a movie as having been watched.

        :param table_name: The name of the movie table.
        :param title: The title of the movie.
        :param year: The year of the movie's release.
        """
        self.session.execute(
            SimpleStatement(
                f"UPDATE {table_name} SET watched=true WHERE title = %s AND year
= %s"
            ),
            parameters=[title, year],
        )
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の以下のトピックを参照してください。
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace

- DeleteTable
- GetKeyspace
- GetTable
- ListKeyspaces
- ListTables
- RestoreTable
- UpdateTable

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDKsアクション

次のコード例は、 AWS SDKs を使用して個々の Amazon Keyspaces アクションを実行する方法を示しています。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細なリストについては、「Amazon Keyspaces (Apache Cassandra 向け) API リファレンス」を参照してください。

例

- AWS SDK CreateKeyspaceで を使用する
- AWS SDK CreateTableで を使用する
- AWS SDK DeleteKeyspaceで を使用する
- AWS SDK DeleteTableで を使用する
- AWS SDK GetKeyspaceで を使用する
- AWS SDK GetTableで を使用する
- AWS SDK ListKeyspacesで を使用する
- AWS SDK ListTablesで を使用する
- AWS SDK RestoreTableで を使用する
- AWS SDK UpdateTableで を使用する

AWS SDK **CreateKeyspace**で を使用する

次のサンプルコードは、CreateKeyspace を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Create a new keyspace.
/// </summary>
/// <param name="keyspaceName">The name for the new keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
public async Task<string> CreateKeyspace(string keyspaceName)
{
    var response =
        await _amazonKeyspaces.CreateKeyspaceAsync(
            new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「CreateKeyspace」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   public static void createKeySpace(KeyspacesClient keyClient, String
keyspaceName) {
       try {
           CreateKeyspaceRequest keyspaceRequest =
CreateKeyspaceRequest.builder()
                   .keyspaceName(keyspaceName)
                   .build();

           CreateKeyspaceResponse response =
keyClient.createKeyspace(keyspaceRequest);
           System.out.println("The ARN of the KeySpace is " +
response.resourceArn());

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「CreateKeyspace」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun createKeySpace(keyspaceNameVal: String) {
    val keyspaceRequest =
        CreateKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.createKeyspace(keyspaceRequest)
        println("The ARN of the KeySpace is ${response.resourceArn}")
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「CreateKeyspace」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""
```

```
    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def create_keyspace(self, name):
        """
        Creates a keyspace.

        :param name: The name to give the keyspace.
        :return: The Amazon Resource Name (ARN) of the new keyspace.
        """
        try:
            response = self.keyspaces_client.create_keyspace(keyspaceName=name)
            self.ks_name = name
            self.ks_arn = response["resourceArn"]
        except ClientError as err:
            logger.error(
                "Couldn't create %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.ks_arn
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「CreateDBCluster」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **CreateTable**で を使用する

次のサンプルコードは、CreateTable を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Create a new Amazon Keyspaces table.
   /// </summary>
   /// <param name="keyspaceName">The keyspace where the table will be
created.</param>
   /// <param name="schema">The schema for the new table.</param>
   /// <param name="tableName">The name of the new table.</param>
   /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
   public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
schema, string tableName)
   {
       var request = new CreateTableRequest
       {
           KeyspaceName = keyspaceName,
           SchemaDefinition = schema,
           TableName = tableName,
           PointInTimeRecovery = new PointInTimeRecovery { Status =
PointInTimeRecoveryStatus.ENABLED }
```

```
        };

        var response = await _amazonKeyspaces.CreateTableAsync(request);
        return response.ResourceArn;
    }
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「CreateTable」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   public static void createTable(KeyspacesClient keyClient, String keySpace,
String tableName) {
       try {
           // Set the columns.
           ColumnDefinition defTitle = ColumnDefinition.builder()
                   .name("title")
                   .type("text")
                   .build();

           ColumnDefinition defYear = ColumnDefinition.builder()
                   .name("year")
                   .type("int")
                   .build();

           ColumnDefinition defReleaseDate = ColumnDefinition.builder()
                   .name("release_date")
                   .type("timestamp")
                   .build();

           ColumnDefinition defPlot = ColumnDefinition.builder()
```

```
                .name("plot")
                .type("text")
                .build();

        List<ColumnDefinition> colList = new ArrayList<>();
        colList.add(defTitle);
        colList.add(defYear);
        colList.add(defReleaseDate);
        colList.add(defPlot);

        // Set the keys.
        PartitionKey yearKey = PartitionKey.builder()
                .name("year")
                .build();

        PartitionKey titleKey = PartitionKey.builder()
                .name("title")
                .build();

        List<PartitionKey> keyList = new ArrayList<>();
        keyList.add(yearKey);
        keyList.add(titleKey);

        SchemaDefinition schemaDefinition = SchemaDefinition.builder()
                .partitionKeys(keyList)
                .allColumns(colList)
                .build();

        PointInTimeRecovery timeRecovery = PointInTimeRecovery.builder()
                .status(PointInTimeRecoveryStatus.ENABLED)
                .build();

        CreateTableRequest tableRequest = CreateTableRequest.builder()
                .keyspaceName(keySpace)
                .tableName(tableName)
                .schemaDefinition(schemaDefinition)
                .pointInTimeRecovery(timeRecovery)
                .build();

        CreateTableResponse response = keyClient.createTable(tableRequest);
        System.out.println("The table ARN is " + response.resourceArn());

    } catch (KeyspacesException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
```

```
            System.exit(1);
        }
    }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「CreateTable」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun createTable(
    keySpaceVal: String?,
    tableNameVal: String?,
) {
    // Set the columns.
    val defTitle =
        ColumnDefinition {
            name = "title"
            type = "text"
        }

    val defYear =
        ColumnDefinition {
            name = "year"
            type = "int"
        }

    val defReleaseDate =
        ColumnDefinition {
            name = "release_date"
            type = "timestamp"
        }
```

```
    val defPlot =
        ColumnDefinition {
            name = "plot"
            type = "text"
        }

    val colList = ArrayList<ColumnDefinition>()
    colList.add(defTitle)
    colList.add(defYear)
    colList.add(defReleaseDate)
    colList.add(defPlot)

    // Set the keys.
    val yearKey =
        PartitionKey {
            name = "year"
        }

    val titleKey =
        PartitionKey {
            name = "title"
        }

    val keyList = ArrayList<PartitionKey>()
    keyList.add(yearKey)
    keyList.add(titleKey)

    val schemaDefinitionOb =
        SchemaDefinition {
            partitionKeys = keyList
            allColumns = colList
        }

    val timeRecovery =
        PointInTimeRecovery {
            status = PointInTimeRecoveryStatus.Enabled
        }

    val tableRequest =
        CreateTableRequest {
            keyspaceName = keySpaceVal
            tableName = tableNameVal
            schemaDefinition = schemaDefinitionOb
            pointInTimeRecovery = timeRecovery
```

```
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.createTable(tableRequest)
        println("The table ARN is ${response.resourceArn}")
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「CreateTable」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)
```

```
    def create_table(self, table_name):
        """
        Creates a table in the  keyspace.
        The table is created with a schema for storing movie data
        and has point-in-time recovery enabled.

        :param table_name: The name to give the table.
        :return: The ARN of the new table.
        """
        try:
            response = self.keyspaces_client.create_table(
                keyspaceName=self.ks_name,
                tableName=table_name,
                schemaDefinition={
                    "allColumns": [
                        {"name": "title", "type": "text"},
                        {"name": "year", "type": "int"},
                        {"name": "release_date", "type": "timestamp"},
                        {"name": "plot", "type": "text"},
                    ],
                    "partitionKeys": [{"name": "year"}, {"name": "title"}],
                },
                pointInTimeRecovery={"status": "ENABLED"},
            )
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["resourceArn"]
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「CreateTable」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **DeleteKeyspace**で を使用する

次のサンプルコードは、DeleteKeyspace を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Delete an existing keyspace.
/// </summary>
/// <param name="keyspaceName"></param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
        new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「DeleteKeyspace」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   public static void deleteKeyspace(KeyspacesClient keyClient, String
keyspaceName) {
       try {
           DeleteKeyspaceRequest deleteKeyspaceRequest =
DeleteKeyspaceRequest.builder()
                   .keyspaceName(keyspaceName)
                   .build();

           keyClient.deleteKeyspace(deleteKeyspaceRequest);

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「DeleteKeyspace」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun deleteKeyspace(keyspaceNameVal: String?) {
    val deleteKeyspaceRequest =
        DeleteKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        keyClient.deleteKeyspace(deleteKeyspaceRequest)
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「DeleteKeyspace」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
```

```
        return cls(keyspaces_client)


    def delete_keyspace(self):
        """
        Deletes the keyspace.
        """
        try:
            self.keyspaces_client.delete_keyspace(keyspaceName=self.ks_name)
            self.ks_name = None
        except ClientError as err:
            logger.error(
                "Couldn't delete keyspace %s. Here's why: %s: %s",
                self.ks_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「DeleteKeyspace」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **DeleteTable**で を使用する

次のサンプルコードは、DeleteTable を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTable(string keyspaceName, string tableName)
    {
        var response = await _amazonKeyspaces.DeleteTableAsync(
            new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「DeleteTable」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    public static void deleteTable(KeyspacesClient keyClient, String
 keyspaceName, String tableName) {
        try {
            DeleteTableRequest tableRequest = DeleteTableRequest.builder()
                    .keyspaceName(keyspaceName)
                    .tableName(tableName)
                    .build();

            keyClient.deleteTable(tableRequest);

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「DeleteTable」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun deleteTable(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
    val tableRequest =
        DeleteTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
```

```
        keyClient.deleteTable(tableRequest)
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「DeleteTable」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def delete_table(self):
        """
        Deletes the table from the keyspace.
        """
```

```
        try:
            self.keyspaces_client.delete_table(
                keyspaceName=self.ks_name, tableName=self.table_name
            )
            self.table_name = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table %s. Here's why: %s: %s",
                self.table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DeleteTable」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **GetKeyspace**で を使用する

次のサンプルコードは、GetKeyspace を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get data about a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
    public async Task<string> GetKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.GetKeyspaceAsync(
            new GetKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「GetKeyspace」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   public static void checkKeyspaceExistence(KeyspacesClient keyClient, String
keyspaceName) {
       try {
           GetKeyspaceRequest keyspaceRequest = GetKeyspaceRequest.builder()
                   .keyspaceName(keyspaceName)
                   .build();

           GetKeyspaceResponse response =
keyClient.getKeyspace(keyspaceRequest);
           String name = response.keyspaceName();
           System.out.println("The " + name + " KeySpace is ready");

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
```

```
            System.exit(1);
        }
    }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「GetKeyspace」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun checkKeyspaceExistence(keyspaceNameVal: String?) {
    val keyspaceRequest =
        GetKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }
    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response: GetKeyspaceResponse =
 keyClient.getKeyspace(keyspaceRequest)
        val name = response.keyspaceName
        println("The $name KeySpace is ready")
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「GetKeyspace」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def exists_keyspace(self, name):
        """
        Checks whether a keyspace exists.

        :param name: The name of the keyspace to look up.
        :return: True when the keyspace exists. Otherwise, False.
        """
        try:
            response = self.keyspaces_client.get_keyspace(keyspaceName=name)
            self.ks_name = response["keyspaceName"]
            self.ks_arn = response["resourceArn"]
            exists = True
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Keyspace %s does not exist.", name)
                exists = False
            else:
                logger.error(
                    "Couldn't verify %s exists. Here's why: %s: %s",
                    name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return exists
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「GetKeyspace」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **GetTable**で を使用する

次のサンプルコードは、GetTable を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
            new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response;
    }
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「GetTable」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    public static void checkTable(KeyspacesClient keyClient, String keyspaceName,
String tableName)
            throws InterruptedException {
        try {
            boolean tableStatus = false;
            String status;
            GetTableResponse response = null;
            GetTableRequest tableRequest = GetTableRequest.builder()
                    .keyspaceName(keyspaceName)
                    .tableName(tableName)
                    .build();
```

```
            while (!tableStatus) {
                response = keyClient.getTable(tableRequest);
                status = response.statusAsString();
                System.out.println(". The table status is " + status);

                if (status.compareTo("ACTIVE") == 0) {
                    tableStatus = true;
                }
                Thread.sleep(500);
            }

            List<ColumnDefinition> cols =
response.schemaDefinition().allColumns();
            for (ColumnDefinition def : cols) {
                System.out.println("The column name is " + def.name());
                System.out.println("The column type is " + def.type());
            }

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「GetTable」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun checkTable(
    keyspaceNameVal: String?,
```

```
    tableNameVal: String?,
) {
    var tableStatus = false
    var status: String
    var response: GetTableResponse? = null

    val tableRequest =
        GetTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }
    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        while (!tableStatus) {
            response = keyClient.getTable(tableRequest)
            status = response!!.status.toString()
            println(". The table status is $status")
            if (status.compareTo("ACTIVE") == 0) {
                tableStatus = true
            }
            delay(500)
        }
        val cols: List<ColumnDefinition>? =
 response!!.schemaDefinition?.allColumns
        if (cols != null) {
            for (def in cols) {
                println("The column name is ${def.name}")
                println("The column type is ${def.type}")
            }
        }
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「GetTable」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def get_table(self, table_name):
        """
        Gets data about a table in the keyspace.

        :param table_name: The name of the table to look up.
        :return: Data about the table.
        """
        try:
            response = self.keyspaces_client.get_table(
                keyspaceName=self.ks_name, tableName=table_name
            )
            self.table_name = table_name
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Table %s does not exist.", table_name)
                self.table_name = None
                response = None
            else:
                logger.error(
                    "Couldn't verify %s exists. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return response
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「GetTable」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **ListKeyspaces**で を使用する

次のサンプルコードは、ListKeyspaces を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Lists all keyspaces for the account.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task ListKeyspaces()
    {
        var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

        Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
        Console.WriteLine(new string('-', Console.WindowWidth));
        await foreach (var keyspace in paginator.Keyspaces)
        {

Console.WriteLine($"{keyspace.KeyspaceName,-30}\t{keyspace.ResourceArn}");
        }
    }
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「ListKeyspaces」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    public static void listKeyspacesPaginator(KeyspacesClient keyClient) {
        try {
            ListKeyspacesRequest keyspacesRequest =
ListKeyspacesRequest.builder()
                    .maxResults(10)
                    .build();
```

```
            ListKeyspacesIterable listRes =
 keyClient.listKeyspacesPaginator(keyspacesRequest);
            listRes.stream()
                    .flatMap(r -> r.keyspaces().stream())
                    .forEach(content -> System.out.println(" Name: " +
 content.keyspaceName()));

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「ListKeyspaces」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun listKeyspacesPaginator() {
    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        keyClient
            .listKeyspacesPaginated(ListKeyspacesRequest {})
            .transform { it.keyspaces?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name: ${obj.keyspaceName}")
            }
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「ListKeyspaces」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def list_keyspaces(self, limit):
        """
        Lists the keyspaces in your account.

        :param limit: The maximum number of keyspaces to list.
        """
        try:
            ks_paginator = self.keyspaces_client.get_paginator("list_keyspaces")
            for page in ks_paginator.paginate(PaginationConfig={"MaxItems":
 limit}):
                for ks in page["keyspaces"]:
                    print(ks["keyspaceName"])
                    print(f"\t{ks['resourceArn']}")
```

```
        except ClientError as err:
            logger.error(
                "Couldn't list keyspaces. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「ListKeyspaces」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **ListTables**で を使用する

次のサンプルコードは、ListTables を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Lists the Amazon Keyspaces tables in a keyspace.
    /// </summary>
```

```
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>A list of TableSummary objects.</returns>
    public async Task<List<TableSummary>> ListTables(string keyspaceName)
    {
        var response = await _amazonKeyspaces.ListTablesAsync(new
ListTablesRequest { KeyspaceName = keyspaceName });
        response.Tables.ForEach(table =>
        {

Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
        });

        return response.Tables;
    }
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「ListTables」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    public static void listTables(KeyspacesClient keyClient, String keyspaceName)
{
        try {
            ListTablesRequest tablesRequest = ListTablesRequest.builder()
                    .keyspaceName(keyspaceName)
                    .build();

            ListTablesIterable listRes =
keyClient.listTablesPaginator(tablesRequest);
            listRes.stream()
                    .flatMap(r -> r.tables().stream())
```

```
                    .forEach(content -> System.out.println(" ARN: " +
 content.resourceArn() +
                            " Table name: " + content.tableName()));

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「ListTables」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun listTables(keyspaceNameVal: String?) {
    val tablesRequest =
        ListTablesRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        keyClient
            .listTablesPaginated(tablesRequest)
            .transform { it.tables?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println(" ARN: ${obj.resourceArn} Table name: ${obj.tableName}")
            }
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「ListTables」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def list_tables(self):
        """
        Lists the tables in the keyspace.
        """
        try:
            table_paginator = self.keyspaces_client.get_paginator("list_tables")
            for page in table_paginator.paginate(keyspaceName=self.ks_name):
                for table in page["tables"]:
                    print(table["tableName"])
```

```
                print(f"\t{table['resourceArn']}")
        except ClientError as err:
            logger.error(
                "Couldn't list tables in keyspace %s. Here's why: %s: %s",
                self.ks_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「ListTables」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **RestoreTable**で を使用する

次のサンプルコードは、RestoreTable を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Restores the specified table to the specified point in time.
```

```
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to restore.</param>
    /// <param name="timestamp">The time to which the table will be restored.</
param>
    /// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
    public async Task<string> RestoreTable(string keyspaceName, string tableName,
 string restoredTableName, DateTime timestamp)
    {
        var request = new RestoreTableRequest
        {
            RestoreTimestamp = timestamp,
            SourceKeyspaceName = keyspaceName,
            SourceTableName = tableName,
            TargetKeyspaceName = keyspaceName,
            TargetTableName = restoredTableName
        };

        var response = await _amazonKeyspaces.RestoreTableAsync(request);
        return response.RestoredTableARN;
    }
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「RestoreTable」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    public static void restoreTable(KeyspacesClient keyClient, String
 keyspaceName, ZonedDateTime utc) {
        try {
            Instant myTime = utc.toInstant();
```

```
            RestoreTableRequest restoreTableRequest =
 RestoreTableRequest.builder()
                    .restoreTimestamp(myTime)
                    .sourceTableName("Movie")
                    .targetKeyspaceName(keyspaceName)
                    .targetTableName("MovieRestore")
                    .sourceKeyspaceName(keyspaceName)
                    .build();

            RestoreTableResponse response =
 keyClient.restoreTable(restoreTableRequest);
            System.out.println("The ARN of the restored table is " +
 response.restoredTableARN());

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「RestoreTable」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun restoreTable(
    keyspaceName: String?,
    utc: ZonedDateTime,
) {
    // Create an aws.smithy.kotlin.runtime.time.Instant value.
    val timeStamp =
        aws.smithy.kotlin.runtime.time
```

```
            .Instant(utc.toInstant())
    val restoreTableRequest =
        RestoreTableRequest {
            restoreTimestamp = timeStamp
            sourceTableName = "MovieKotlin"
            targetKeyspaceName = keyspaceName
            targetTableName = "MovieRestore"
            sourceKeyspaceName = keyspaceName
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.restoreTable(restoreTableRequest)
        println("The ARN of the restored table is ${response.restoredTableArn}")
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「RestoreTable」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
```

```
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def restore_table(self, restore_timestamp):
        """
        Restores the table to a previous point in time. The table is restored
        to a new table in the same keyspace.

        :param restore_timestamp: The point in time to restore the table. This
time
                                  must be in UTC format.
        :return: The name of the restored table.
        """
        try:
            restored_table_name = f"{self.table_name}_restored"
            self.keyspaces_client.restore_table(
                sourceKeyspaceName=self.ks_name,
                sourceTableName=self.table_name,
                targetKeyspaceName=self.ks_name,
                targetTableName=restored_table_name,
                restoreTimestamp=restore_timestamp,
            )
        except ClientError as err:
            logger.error(
                "Couldn't restore table %s. Here's why: %s: %s",
                restore_timestamp,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return restored_table_name
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「RestoreTable」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **UpdateTable**で を使用する

次のサンプルコードは、UpdateTable を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Updates the movie table to add a boolean column named watched.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to change.</param>
    /// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
    public async Task<string> UpdateTable(string keyspaceName, string tableName)
    {
        var newColumn = new ColumnDefinition { Name = "watched", Type =
"boolean" };
        var request = new UpdateTableRequest
        {
            KeyspaceName = keyspaceName,
            TableName = tableName,
            AddColumns = new List<ColumnDefinition> { newColumn }
        };
        var response = await _amazonKeyspaces.UpdateTableAsync(request);
        return response.ResourceArn;
```

```
    }
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「UpdateTable」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   public static void updateTable(KeyspacesClient keyClient, String keySpace,
String tableName) {
       try {
           ColumnDefinition def = ColumnDefinition.builder()
                   .name("watched")
                   .type("boolean")
                   .build();

           UpdateTableRequest tableRequest = UpdateTableRequest.builder()
                   .keyspaceName(keySpace)
                   .tableName(tableName)
                   .addColumns(def)
                   .build();

           keyClient.updateTable(tableRequest);

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「UpdateTable」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun updateTable(
    keySpace: String?,
    tableNameVal: String?,
) {
    val def =
        ColumnDefinition {
            name = "watched"
            type = "boolean"
        }

    val tableRequest =
        UpdateTableRequest {
            keyspaceName = keySpace
            tableName = tableNameVal
            addColumns = listOf(def)
        }

    KeyspacesClient.fromEnvironment { region = "us-east-1" }.use { keyClient ->
        keyClient.updateTable(tableRequest)
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「UpdateTable」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def update_table(self):
        """
        Updates the schema of the table.

        This example updates a table of movie data by adding a new column
        that tracks whether the movie has been watched.
        """
        try:
            self.keyspaces_client.update_table(
                keyspaceName=self.ks_name,
                tableName=self.table_name,
                addColumns=[{"name": "watched", "type": "boolean"}],
            )
```

```
        except ClientError as err:
            logger.error(
                "Couldn't update table %s. Here's why: %s: %s",
                self.table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の「UpdateTable」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK でのこのサービスの使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Keyspaces (Apache Cassandra 向け) のライブラリとツール

このセクションでは、Amazon Keyspaces (Apache Cassandra 向け) のライブラリ、コード例およびツールについて説明します。

トピック

- ライブラリと例
- ハイライトされたサンプルおよびデベロッパーツールのリポジトリ

ライブラリと例

Amazon Keyspaces のオープンソースライブラリとデベロッパーツールは、AWS と AWS サンプルリポジトリの GitHub にあります。

Amazon Keyspaces (Apache Cassandra 向け) のデベロッパーツールキット

このリポジトリには、Amazon Keyspaces の便利なデベロッパーツールを備えた Docker イメージがあります。例えば、ベストプラクティスを含む CQLSHRC ファイル、cqlsh のオプションの AWS 認証拡張、一般的なタスクを実行するヘルパーツールが含まれています。このツールキットは Amazon Keyspaces 用に最適化されていますが、Apache Cassandra クラスターでも機能します。

https://github.com/aws-samples/amazon-keyspaces-toolkit.

Amazon Keyspaces (Apache Cassandra 向け) の事例

このリポジトリは、Amazon Keyspaces サンプルコードの公式リストです。このリポジトリは言語別にセクションに細分されます (「事例」を参照)。各言語には独自の事例サブセクションがあります。これらの事例では、アプリケーションを構築する際に使用できる一般的な Amazon Keyspaces サービスの実装とパターンが示されています。

https://github.com/aws-samples/amazon-keyspaces-examples/.

AWS 署名バージョン 4 (SigV4) 認証プラグイン

プラグインを使用すると、 AWS Identity and Access Management (IAM) ユーザーとロールを使用して Amazon Keyspaces へのアクセスを管理できます。

Java: https://github.com/aws/aws-sigv4-auth-cassandra-java-driver-plugin。

Node.js: https://github.com/aws/aws-sigv4-auth-cassandra-nodejs-driver-plugin。

Python: https://github.com/aws/aws-sigv4-auth-cassandra-python-driver-plugin。

Go: https://github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin。

ハイライトされたサンプルおよびデベロッパーツールのリポジトリ

以下は、Amazon Keyspaces (Apache Cassandra 向け) に役立つコミュニティツールのセレクションです。

Amazon Keyspaces のプロトコルバッファ

Amazon Keyspaces でプロトコルバッファ (Protobuf) を使用すると、Apache Cassandra ユーザー定義タイプ (UDT) に代わる定義タイプを提供できます。Protobuf は、構造化データのシリアル化に使用する無料のオープンソースのクロスプラットフォームデータ形式です。アプリケーションやプログラミング言語全体で構造化データを保持しながら、CQL BLOB データ型とリファクタリング UDT を使用して Protobuf データを保存できます。

このリポジトリには、Amazon Keyspaces に接続し、新しいテーブルを作成し、Protobufメッセージを含む行を挿入するコード例が用意されています。その後、その行は強一貫性を保ちながら読み込まれます。

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/protobuf-user-defined-types

AWS CloudFormation Amazon Keyspaces (Apache Cassandra 向け) メトリクス用の Amazon CloudWatch ダッシュボードを作成するための テンプレート

このリポジトリには、Amazon Keyspaces の CloudWatch メトリクスをすばやくセットアップするための AWS CloudFormation テンプレートが用意されています。このテンプレートを使用すると、

デプロイ可能な事前構築済み CloudWatch ダッシュボードに一般的に使用されるメトリクスを取り入れることができるので、より簡単に使用を開始できます。

https://github.com/aws-samples/amazon-keyspaces-cloudwatch-cloudformation-templates.

での Amazon Keyspaces (Apache Cassandra 向け) の使用 AWS Lambda

リポジトリには、Lambda から Amazon Keyspaces に接続する方法が示されている事例が含まれています。以下に一部の例を示します。

C#/.NET: https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/dotnet/datastax-v3/connection-lambda。

Java: https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/connection-lambda。

Python Lambda から Amazon Keyspaces をデプロイして使用する方法を示した Lambda 例がもう 1 つあり、次のリポジトリから入手できます。

https://github.com/aws-samples/aws-keyspaces-lambda-python

Spring と Amazon Keyspaces (Apache Cassandra 向け) の使用

これは Spring Boot とともに Amazon Keyspaces を使用する方法の例です。

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/spring

Scala と Amazon Keyspaces (Apache Cassandra 向け) の使用

これは、Scala とともに SigV4 認証プラグインを使用して Amazon Keyspaces に接続する方法を示した例です。

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/scala/datastax-v4/connection-sigv4

での Amazon Keyspaces (Apache Cassandra 向け) の使用 AWS Glue

これは、 で Amazon Keyspaces を使用する方法を示す例です AWS Glue。

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/scala/datastax-v4/aws-glue

Amazon Keyspaces (Apache Cassandra 向け) Cassandra クエリ言語 (CQL) から AWS CloudFormation コンバーターへ

本パッケージは、Apache Cassandra クエリ言語 (CQL) スクリプトを AWS CloudFormation (CloudFormation) テンプレートに変換するコマンドラインツールを実装するものです。このツールを使えば、Amazon Keyspaces のスキーマを CloudFormation スタックで簡単に管理できます。

https://github.com/aws/amazon-keyspaces-cql-to-cfn-converter.

Java 向け Apache Cassandra ドライバーの Amazon Keyspaces (Apache Cassandra 向け) ヘルパー

このリポジトリには、DataStax Java ドライバーを Amazon Keyspaces (Apache Cassandra 向け) で使用する場合のドライバーポリシー、例、およびベストプラクティスが含まれています。

https://github.com/aws-samples/amazon-keyspaces-java-driver-helpers.

Amazon Keyspaces (Apache Cassandra 向け) 圧縮の簡単なデモ

このリポジトリは、パフォーマンスの向上およびスループットとストレージのコスト削減を実現できる、大きなオブジェクトの圧縮、保存、読み取り/書き込み方法を示しています。

https://github.com/aws-samples/amazon-keyspaces-compression-example.

Amazon Keyspaces (Apache Cassandra 向け) と Amazon S3 Codec のデモ

カスタムの Amazon S3 Codec では、Amazon S3 オブジェクトに対する UUID ポインターのユーザー設定可能な透過的マッピングがサポートされています。

https://github.com/aws-samples/amazon-keyspaces-large-object-s3-demo.

Amazon Keyspaces (Apache Cassandra 向け) を使用した設計とアーキテクチャに関するベストプラクティス

このセクションでは、Amazon Keyspaces の使用時に最大限のパフォーマンスを実現し、スループットコストを最小限に止めるための推奨事項を短時間に見極めることができます。

目次

- NoSQL 設計の主な相違点と設計原則
  - リレーショナルデータ設計と NoSQL の相違点
  - NoSQL 設計の 2 つの重要な概念
  - NoSQL 設計へのアプローチ
- サーバーレス環境のクライアントドライバー接続を最適化する
  - Amazon Keyspaces における接続の働き
  - Amazon Keyspaces で接続を設定する方法
  - Amazon Keyspaces で接続の再試行ポリシーを設定する方法
  - Amazon Keyspaces の VPC エンドポイント経由接続の設定方法
  - Amazon Keyspaces で接続をモニタリングする方法。
  - Amazon Keyspaces での接続エラーの処理方法
- データモデリングのベストプラクティス: データモデル設計時の推奨事項
  - Amazon Keyspaces でパーティションキーを効果的に使用する方法
    - 書き込みシャーディングを使用してワークロードをパーティション間で均等に分散させる
      - 複合パーティションキーと乱数値を使用したシャーディング
      - 複合パーティションキーと計算値を使用したシャーディング
- Amazon Keyspaces テーブルのコスト最適化
  - テーブルレベルでコストを評価する
    - 1 つの Amazon Keyspaces テーブルのコストを表示する方法
    - Cost Explorer のデフォルトビュー
    - Cost Explorer でのテーブルタグの使用方法および適用方法
  - テーブルのキャパシティモードの評価
    - 利用可能なテーブルキャパシティモード
    - オンデマンドキャパシティモードを選択する場合

    - プロビジョンドキャパシティモードを選択する場合
    - テーブルキャパシティモードを選択する際に考慮すべきその他の要素
- テーブルの Application Auto Scaling 設定を評価する
    - Application Auto Scaling 設定の理解
    - 目標使用率が低い (50% 未満) テーブルを特定する方法
    - 季節変動のあるワークロードに対処する方法
    - パターンが不明な急増するワークロードに対処する方法
    - リンクされたアプリケーションのワークロードに対応する方法
- 未使用のリソースを特定して Amazon Keyspaces のコストを最適化する
    - 未使用のリソースを特定する方法
    - 未使用のテーブルリソースを特定する
    - 未使用のテーブルリソースをクリーンアップする
    - 未使用のポイントインタイムリカバリ (PITR) バックアップのクリーンアップ
- テーブルの使用パターンを評価してパフォーマンスとコストを最適化する
    - 強力な整合性のある読み込みオペレーションの数を減らす
    - 有効期限 (TTL) の有効化
- 適切なサイズのプロビジョニングを行うために、プロビジョンドキャパシティを評価する
    - Amazon Keyspaces テーブルから消費メトリクスを取得する方法
    - プロビジョニング不足の Amazon Keyspaces テーブルを識別する方法
    - 過剰にプロビジョニングされた Amazon Keyspaces テーブルを識別する方法

NoSQL 設計の主な相違点と設計原則

Amazon Keyspaces などの NoSQL データベースシステムでは、キーと値のペアやドキュメントストレージなど、データ管理に代替モデルを使用します。リレーショナルデータベース管理システム (RDBMS) から Amazon Keyspaces のような NoSQL データベースシステムに切り替えるときは、他のシステムとの主な違いと独自の設計アプローチを理解することが重要です。

トピック

- リレーショナルデータ設計と NoSQL の相違点
- NoSQL 設計の 2 つの重要な概念
- NoSQL 設計へのアプローチ

リレーショナルデータ設計と NoSQL の相違点

リレーショナルデータベースシステム (RDBMS) と NoSQL データベースにはそれぞれ異なる長所と短所があります。

- RDBMS では、データは柔軟にクエリできますが、クエリは比較的コストが高く、トラフィックが多い状況ではスケールがうまくいかない場合があります (the section called “データモデリング” 参照)。
- 一方、Amazon Keyspaces のような NoSQL データベースでは、一定の方式では、データを効率的にクエリできますが、それ以外の方式では、クエリのコストが高くなり、処理速度が遅くなりがちです。

これらの相違点により、2 つのシステム間でデータベース設計が異なるものになります。

- RDBMS では、実装の詳細やパフォーマンスを気にせずに柔軟に設計できます。クエリの最適化は一般的にスキーマ設計には影響しませんが、正規化は重要です。
- Amazon Keyspaces では、最も一般的で重要なクエリをできるだけ速く、経済的に処理するために、具体的にスキーマを設計します。データ構造は、ビジネスユースケースの特定の要件に合わせて調整されています。

NoSQL 設計の 2 つの重要な概念

NoSQL 設計では、RDBMS 設計とは異なる考え方が必要です。RDBMS の場合は、アクセスパターンを考慮せずに正規化されたデータモデルを作成できます。その後、新しい課題とクエリの要件が発生したら、そのデータモデルを拡張することができます。各タイプのデータを独自のテーブルに整理できます。

NoSQL の設計の違い

- Amazon Keyspaces の場合は対照的に、答えが必要な疑問点が分かるまで、スキーマの設計を開始しないでください。ビジネス上の問題とアプリケーションのユースケースを理解することが不可欠です。
- Amazon Keyspaces アプリケーションでは、できるだけ少ないテーブルで済ませる必要があります。テーブルの数が少なければ、スケーラビリティが向上し、必要な権限の管理業務が減少し、Amazon Keyspaces アプリケーションのオーバーヘッドが削減されます。また、バックアップコストを全体的に低く抑えるのにも役立ちます。

NoSQL 設計へのアプローチ

Amazon Keyspaces アプリケーション設計の最初のステップでは、システムで対応すべき具体的なクエリパターンを見極めます。

始める前に、特にアプリケーションのアクセスパターンの 3 つの基本的な特性を理解することが重要です。

- データサイズ: 一度に収容されて、リクエストされるデータの量を把握できれば、最も効果的なデータパーティションの方式を決定できます。
- データシェイプ: クエリが処理される際 (RDBMS システムのように) データを再形成するのではなく、データベースの形状がクエリ処理に対応するように、NoSQL データベースでデータを整理します。これは、スピードとスケーラビリティを向上させる重要な要素です。
- データ速度: Amazon Keyspaces では、クエリ処理に使用できる物理パーティションの数を増やし、それらのパーティション間で効率的にデータを分散させてスケーリングします。ピーク時のクエリのロードを事前に把握することは、I/O キャパシティを最大限に活用するためにデータを分割する方法を決定する上で役立ちます。

特定のクエリ要件を特定したら、パフォーマンスを管理する一般的な原則に従ってデータを整理できます。

- 関連データをまとめて保存します。 20 年前のルーティングテーブルの最適化に関する研究では、関連するデータをまとめて 1 つの場所にまとめておく「参照の局所性」が、応答時間を短縮する上で最も重要な要素であることがわかりました。これは、今日の NoSQL システムにも同様に当てはまります。関連するデータを近くに置くことはコストとパフォーマンスに大きな影響を与えます。関連するデータ項目を複数のテーブルに分散するのではなく、NoSQL システム内の関連項目を可能な限り近くにまとめる必要があります。

  原則として、Amazon Keyspaces アプリケーションではできるだけテーブル数を少なくする必要があります。

  例外として、大キャパシティの時系列データが必要な場合や、データセットのアクセスパターンが大きく異なる場合などがあります。反転されたインデックスを含む単一のテーブルでは通常、シンプルなクエリを使用してアプリケーションで必要とされる複雑な階層データ構造を構築および取得できます。
- ソート順を使用します。 キーの設計が原因でソートされている場合は、関連項目をグループ化して、効率的にクエリできます。これは重要な NoSQL 設計方法です。

- クエリを分散します。　大キャパシティのクエリがデータベースの一部に集中しないことも重要です。I/O キャパシティを超えるおそれがあります。その代わり、「ホットスポット」を避け、できるだけ多くのパーティションにトラフィックを均等に分散するように、データキーを設計する必要があります。

このような一般原則は、Amazon Keyspaces でデータを効率的にモデル化するための一般的なデザインパターンに書き換えられます。

サーバーレス環境のクライアントドライバー接続を最適化する

Amazon Keyspaces との通信には、既存の任意の Apache Cassandra クライアントドライバーを使用できます。Amazon Keyspaces はサーバーレスサービスなので、アプリケーションのスループットニーズに合わせてクライアントドライバーの接続設定を最適化することをおすすめします。このトピックでは、アプリケーションに必要な接続数の計算方法や、接続のモニタリングやエラー処理などのベストプラクティスを紹介します。

トピック

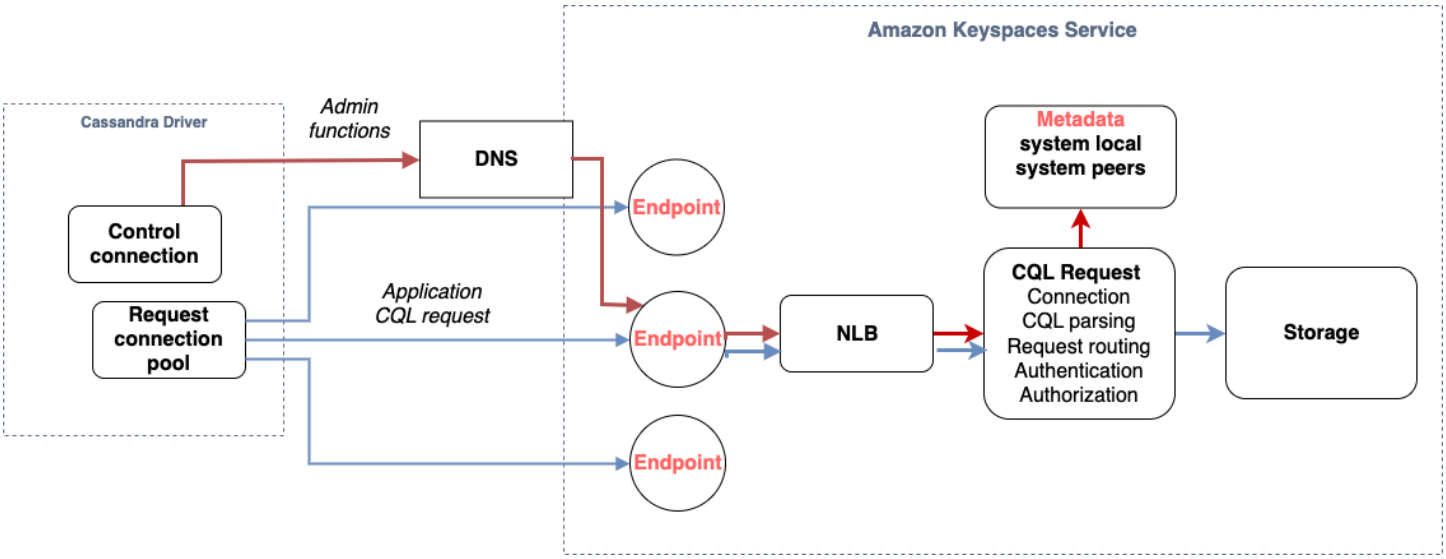
- Amazon Keyspaces における接続の働き
- Amazon Keyspaces で接続を設定する方法
- Amazon Keyspaces で接続の再試行ポリシーを設定する方法
- Amazon Keyspaces の VPC エンドポイント経由接続の設定方法
- Amazon Keyspaces で接続をモニタリングする方法。
- Amazon Keyspaces での接続エラーの処理方法

Amazon Keyspaces における接続の働き

このセクションでは、Amazon Keyspaces におけるクライアントドライバー接続の働きの概要を説明します。Cassandra クライアントドライバーの設定を誤ると Amazon Keyspaces で `PerConnectionRequestExceeded` イベントが発生するおそれがあるので、これらのエラーや同様の接続エラーを回避するため、クライアントドライバー設定で適切な接続数を設定する必要があります。

Amazon Keyspaces に接続する場合、ドライバーには初期接続を確立するためのシードエンドポイントが必要です。Amazon Keyspaces は、利用可能な多数のエンドポイントのいずれかに DNS で初期接続をルーティングします。エンドポイントはネットワークロードバランサーにアタッチされ、フ

リート内のリクエストハンドラーの 1 つとの接続がロードバランサーによって確立されます。最初の接続が確立すると、クライアントドライバーは利用可能なすべてのエンドポイントに関する情報を `system.peers` テーブルから収集します。この情報から、クライアントドライバーはリストされたエンドポイントとの追加の接続を作成できます。クライアントドライバーが作成できる接続数は、クライアントドライバー設定で指定されたローカル接続数が上限になります。デフォルトでは、ほとんどのクライアントドライバーが、エンドポイントごとに 1 つの接続を確立し、Cassandra までの接続プールを確立し、その接続プールに対してクエリのロードバランスを行います。同じエンドポイントに複数の接続を確立できますが、ネットワークロードバランサーの背後では、それらの接続は、さまざまなリクエストハンドラーにつながっている場合があります。パブリックエンドポイント経由で接続するとき、`system.peers` テーブルに記載された 9 つのエンドポイントのそれぞれに 1 つの接続を確立すると、異なるリクエストハンドラーに対して 9 つの接続が作成されます。



Amazon Keyspaces で接続を設定する方法

Amazon Keyspaces は、TCP 接続 1 つにつき 1 秒あたり最大 3,000 の CQL クエリに対応しています。ドライバーが確立できる接続数には制限がないため、オーバーヘッド、トラフィックバースト、負荷分散を考慮して、1 接続あたり 1 秒あたり 500 件の CQL リクエストのみを目標にすることをおすすめします。以下の手順に従って、ドライバーの接続がアプリケーションのニーズに合わせて正しく設定されていることを確認してください。

この数を増やすには、接続プールでドライバーが維持管理している各 IP アドレスの接続数の増加をおすすめします。

- ほとんどの Cassandra ドライバーで、Cassandra までの接続プールが確立され、その接続プールでクエリのロードバランスが行われます。ほとんどのドライバーは、デフォルト動作で、エンドポイントごとに 1 本の接続を確立します。Amazon Keyspaces では 9 つのピア IP アドレスが

ドライバーに公開されているため、ほとんどのドライバーのデフォルト動作に従えば、接続数は 9 本になります。Amazon Keyspaces は、TCP 接続 1 本につき 1 秒あたり最大 3,000 の CQL クエリに対応しているため、デフォルト設定を使用するドライバーの最大 CQL クエリスループットは、1 秒あたり 27,000 件の CQL クエリになります。ドライバーのデフォルト設定を使用すると、1 本の接続で、1 秒あたり 3,000 CQL クエリを超える処理が必要になる可能性があります。その場合、`PerConnectionRequestExceeded` イベントが発生するおそれがあります。

- `PerConnectionRequestExceeded` イベントを回避するには、エンドポイントごとの追加接続を作成してスループットが分散するようにドライバーを設定する必要があります。
- Amazon Keyspaces のベストプラクティスとしては、各接続でサポートできる想定 CQL クエリ数を毎秒 500 件としています。
- したがって、使用可能な 9 つのエンドポイントに分散される 1 秒あたり CQL クエリ数を推定 27,000 件にする必要がある本番アプリケーションでは、エンドポイントごとに 6 本の接続を設定する必要があります。これにより、各接続が 1 秒あたり 500 件以下のリクエストを処理できます。

アプリケーションのニーズに基づいて、ドライバーに設定する必要がある IP アドレスごとの接続数を計算します。

アプリケーションのエンドポイントごとに設定が要な接続数を決定するため、次の例で考えてみましょう。例えば、10,000`INSERT`、5,000`SELECT`、5,000`DELETE` 件の操作からなる、1 秒あたり 20,000 件の CQL クエリのサポートが求められているアプリケーションがあるとします。Java アプリケーションは Amazon Elastic Container Service (Amazon ECS) の 3 つのインスタンスで実行されており、各インスタンスは Amazon Keyspaces への 1 件のセッションを確立します。ドライバーに設定する必要がある接続数を推定できる計算では、次の入力を使用します。

1. アプリケーションがサポートする必要がある 1 秒あたりのリクエスト数。
2. 利用可能なインスタンスの数を、メンテナンスや障害を考慮して 1 つ減らした数です。
3. 利用可能なエンドポイント数。パブリックエンドポイント経由で接続している場合、利用可能なエンドポイントは 9 つになります。VPC エンドポイントを使用している場合は、リージョンに応じて 2 ~ 5 つのエンドポイントを使用できます。
4. Amazon Keyspaces のベストプラクティスとして、接続ごとに 1 秒あたり 500 件の CQL クエリを使用します。
5. 結果は切り上げてください。

この例では、計算式は次のようになります。

```
20,000 CQL queries / (3 instances - 1 failure) / 9 public endpoints / 500 CQL queries
 per second = ROUND(2.22) = 3
```

この計算から、このドライバー設定ではエンドポイントごとに 3 本のローカル接続を指定する必要があります。リモート接続の場合は、エンドポイントごとに 1 本のみの接続を設定します。

Amazon Keyspaces で接続の再試行ポリシーを設定する方法

Amazon Keyspaces への接続の再試行ポリシーを設定する場合は、Amazon Keyspaces の再試行ポリシー `AmazonKeyspacesExponentialRetryPolicy` を実装することをお勧めします。この再試行ポリシーは、Amazon Keyspaces への複数の接続をまたいで再試行する場合に、ドライバーの `DefaultRetryPolicy` よりも適しています。

`AmazonKeyspacesExponentialRetryPolicy` では、接続の再試行回数をニーズに応じて設定できます。デフォルトでは、`AmazonKeyspacesExponentialRetryPolicy` の再試行回数は 3 に設定されています。

もう 1 つの利点として、Amazon Keyspaces の再試行ポリシーは、サービスで返された元の例外を返すため、リクエストの試行が失敗した理由がわかります。デフォルトの再試行ポリシーは汎用的な `NoHostAvailableException` しか返しません。そのため、リクエスト失敗の原因が詳しくわからない場合があります。

`AmazonKeyspacesExponentialRetryPolicy` を使用してリクエスト再試行ポリシーを設定する場合は、再試行回数を少なく設定し、返された例外をアプリケーションコードで処理することをお勧めします。

再試行ポリシーを実装するコード例については、Github の Amazon Keyspaces 再試行ポリシーを参照してください。

Amazon Keyspaces の VPC エンドポイント経由接続の設定方法

プライベート VPC エンドポイントを介して接続する場合、通常は、3 つのエンドポイントを利用できます。VPC エンドポイントの数は、アベイラビリティーゾーンの数と、割り当てられた VPC 内のサブネットの数に応じて、リージョンごとに異なる場合があります。米国東部 (バージニア北部) リージョンには 5 つのアベイラビリティーゾーンがあり、最大 5 つの Amazon Keyspaces エンドポイントを設定できます。米国西部 (北カリフォルニア) リージョンには 2 つのアベイラビリティーゾーンがあり、最大 2 つの Amazon Keyspaces エンドポイントを使用できます。エンドポイントの数は規模には影響しませんが、ドライバー設定で確立する必要のある接続数は増えます。次の例を考

えます。アプリケーションは 20,000 件の CQL クエリをサポートする必要があり、インスタンスごとに Amazon Keyspaces へのセッションが 1 件確立される Amazon ECS 上の 3 つのインスタンスで実行されています。唯一の違いは、異なる で使用可能なエンドポイントの数です AWS リージョン。

米国東部 (バージニア北部) リージョンで必要な接続数:

```
20,000 CQL queries / (3 instances - 1 failure) / 5 private VPC endpoints / 500 CQL
 queries per second = 4 local connections
```

米国西部 (北カリフォルニア) リージョンで必要な接続数:

```
20,000 CQL queries / (3 instances - 1 failure) / 2 private VPC endpoints / 500 CQL
 queries per second = 10 local connections
```

> **Important**
>
> プライベート VPC エンドポイントを使用するとき、Amazon Keyspaces で使用可能な VPC エンドポイントを動的に検出して `system.peers` テーブルに入力するには、Amazon Keyspaces には追加の権限が必要です。詳細については、「the section called “インターフェイス VPC エンドポイント情報を含む `system.peers` テーブルエントリの入力”」を参照してください。

別の VPC エンドポイントを使用してプライベート VPC エンドポイントから Amazon Keyspaces にアクセスする場合 AWS アカウント、1 つの Amazon Keyspaces エンドポイントしか表示されない可能性があります。繰り返しますが、これは Amazon Keyspaces へのスループットの規模には影響しませんが、場合によっては、ドライバー設定の接続数を増やす必要があります。この例では、使用可能な 1 つのエンドポイントに同じ計算を行っています。

```
20,000 CQL queries / (3 instances - 1 failure) / 1 private VPC endpoints / 500 CQL
 queries per second = 20 local connections
```

共有 VPC を使用した Amazon Keyspaces へのクロスアカウントアクセスの詳細については、「the section called “共有 VPC のクロスアカウントアクセスの設定”」を参照してください。

Amazon Keyspaces で接続をモニタリングする方法。

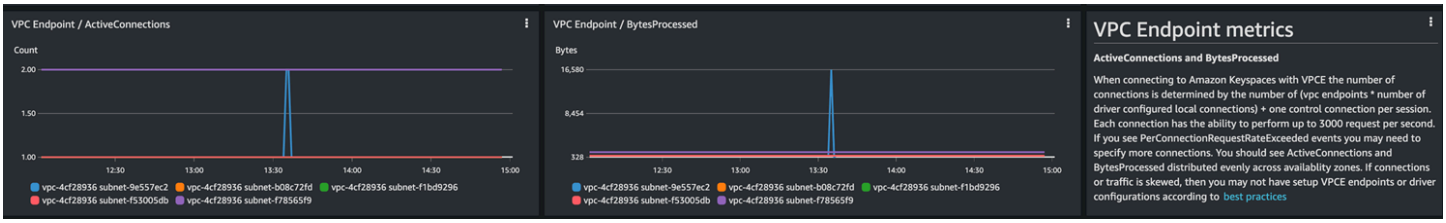
アプリケーションが接続されているエンドポイントの数がすぐにわかるように、検出されたピアの数を `system.peers` テーブルに記録できます。次の例は、接続が確立されるとピア数を出力する Java コードの例です。

```
ResultSet result = session.execute(new SimpleStatement("SELECT * FROM system.peers"));

logger.info("number of Amazon Keyspaces endpoints:" + result.all().stream().count());
```

> **Note**
>
> CQL コンソールまたは AWS コンソールは VPC 内にデプロイされないため、パブリックエンドポイントを使用します。そのため、VPCE の外部にあるアプリケーションから `system.peers` クエリを実行すると、多くの場合、ピアは 9 つになります。このとき、各ピアの IP アドレスを印刷しておくと役立つ場合があります。

VPCE Amazon CloudWatch メトリクスを設定すると、VPC エンドポイントを使用する際のピア数も確認できます。CloudWatch では、VPC エンドポイントに対して確立された接続の数を確認できます。Cassandra ドライバーは、CQL クエリを送信する各エンドポイントの接続と、システムテーブル情報を収集するための制御接続を確立します。以下の画像は、ドライバー設定で 1 つの接続を設定して Amazon Keyspaces に接続した後の VPC エンドポイントの CloudWatch メトリクスを示しています。このメトリクスには、1 本のコントロール接続と 5 本の接続 (アベイラビリティーゾーン全体でエンドポイントごとに 1 つ) からなる 6 本のアクティブな接続が表示されています。



CloudWatch グラフを使用して接続数のモニタリングを開始するには、Amazon Keyspaces AWS CloudFormation テンプレートリポジトリの GitHub で利用可能なこのテンプレートをデプロイします。 https://github.com/aws-samples/amazon-keyspaces-cloudwatch-cloudformation-templates

Amazon Keyspaces での接続エラーの処理方法

接続あたりのリクエストが 3,000 件を超えると、Amazon Keyspaces は `PerConnectionRequestExceeded` イベントを返し、Cassandra ドライバーは `WriteTimeout` 例外や `ReadTimeout` 例外を受け取ります。その場合は、Cassandra の再試行ポリシーまたはアプリケーションで、指数バックオフを設定してこの例外を再試行してください。このとき、追加のリクエストを送信しないように、指数バックオフを設定する必要があります。

デフォルト再試行ポリシーでは、クエリプランの `try next host` の再試行が試みられます。場合によっては、Amazon Keyspaces には VPC エンドポイントの接続に使用できるエンドポイントが 1 ~ 3 つあるため、アプリケーションログには `NoHostAvailableException` 以外に `WriteTimeout` 例外と `ReadTimeout` 例外も表示される場合があります。Amazon Keyspaces が用意した再試行ポリシーを使用できます。この再試行ポリシーは、同じエンドポイントですが、異なる接続間で再試行します。

GitHub の Java の指数再試行ポリシーの例は、Amazon Keyspaces Java コード例リポジトリにあります。その他の言語サンプルは、Github の Amazon Keyspaces コード例リポジトリにあります。

データモデリングのベストプラクティス: データモデル設計時の推奨事項

Amazon Keyspaces (Apache Cassandra 向け) を使用する際にパフォーマンスを最適化し、コストを最小限に抑えるには、効果的なデータモデリングが不可欠です。このトピックでは、アプリケーションのデータアクセスパターンに適したデータモデルを設計する上で重要となる考慮事項と推奨事項を取り上げます。

- パーティションキー設計 – パーティションキーは、Amazon Keyspaces の複数のパーティション間でどのようにデータを分散させるかを決める上で重要な役割を果たします。適切なパーティションキーを選択することが、クエリのパフォーマンスやスループットコストを大きく左右する可能性があります。このセクションでは、パーティション間で読み取り/書き込みアクティビティを均等に分散させるためのパーティションキー設計の戦略について説明します。
- 主な考慮事項:
  - アクティビティの均等分散 – スループットコストを最小限に抑え、バーストキャパシティを効果的に活用するために、読み取り/書き込みのアクティビティをすべてのパーティションに均等に分散させることを目指します。
  - アクセスパターン – アプリケーションのプライマリデータのアクセスパターンに合わせて、パーティションキーを設計します。

- パーティションサイズ – 肥大化するパーティションはパフォーマンスに支障をきたし、コスト増の原因となりかねないため、作成しないようにします。

NoSQL Workbench を使用すれば、データモデルの可視化と設計が簡単になります。

トピック

- Amazon Keyspaces でパーティションキーを効果的に使用する方法

Amazon Keyspaces でパーティションキーを効果的に使用する方法

Amazon Keyspaces テーブルの各行を一意に識別するプライマリキーは、データを保存するパーティションを決定する 1 つ以上のパーティションキー列と、パーティション内のデータのクラスター化とソートの方法を定義する 1 つ以上のクラスタリング列 (オプション) で構成されます。

パーティションキーによって、データが保存されるパーティションの数と、これらのパーティション間でデータを分散させる方法が決まるため、パーティションキーの選択方法が、クエリのパフォーマンスに大きな影響を与える可能性があります。通常は、ディスク上のすべてのパーティション全体でアクティビティが均一になるようにアプリケーションを設計する必要があります。

アプリケーションの読み取り/書き込みアクティビティをすべてのパーティションに均等に分散させることで、スループットコストを最小限に抑えることができます。これは、オンデマンドおよび読み取り/書き込みのプロビジョンドキャパシティモードにも適用されます。例えば、プロビジョンドキャパシティモードを使用している場合、アプリケーションに必要なアクセスパターンの決定と、各テーブルに必要な読み取りキャパシティユニット (RCU) と書き込みキャパシティユニット (WCU) の合計の見積もりが可能です。Amazon Keyspaces では、指定されたパーティションのトラフィックが 3,000 RCU と 1,000 WCU を超えない限り、プロビジョニングされたスループットによってアクセスパターンはサポートされます。

パーティションの読み取りまたは書き込みスループットが持続的に高い場合、トラフィックパターンに応じて、Amazon Keyspaces は自動的にパーティションを 2 つの新しいパーティションに分割することがあります。各新しいパーティションには元のパーティションの行のサブセットが含まれ、両方のパーティションにスループットが均等に分散されます。

Amazon Keyspaces では、バーストキャパシティを指定して、パーティションごとのスループットプロビジョニングに柔軟性をもたらします。詳細については、the section called “バーストキャパシティの使用” を参照してください。

トピック

- 書き込みシャーディングを使用してワークロードをパーティション間で均等に分散させる

書き込みシャーディングを使用してワークロードをパーティション間で均等に分散させる

Amazon Keyspaces のパーティションで効率的な書き込みの分散を行う方法として、スペースの拡張があります。これにはさまざまな方法があります。乱数を書き込んだパーティションキー列を追加して、パーティション間で行を分散させることができます。または、クエリ対象に基づいて計算された数値を使用することができます。

複合パーティションキーと乱数値を使用したシャーディング

パーティション全体に均等にロードを分散させるための戦略として、乱数を書き込んだパーティションキー列を追加する方法があります。次に、より大きなスペース全体の書き込みをランダム化します。

例えば、次のテーブルに、日付を表すパーティションキーが 1 つあるとします。

```
CREATE TABLE IF NOT EXISTS tracker.blogs (
   publish_date date,
   title text,
   description int,
   PRIMARY KEY (publish_date));
```

このテーブルをパーティション間で均等に分散するために、乱数が保存されている追加のパーティションキー列 shard を含めることができます。例:

```
CREATE TABLE IF NOT EXISTS tracker.blogs (
   publish_date date,
   shard int,
   title text,
   description int,
   PRIMARY KEY ((publish_date, shard)));
```

データを挿入するときに、shard 列に対して 1～200 の乱数を選択するとします。これにより、(2020-07-09, 1) や (2020-07-09, 2) などから (2020-07-09, 200) までの複合パーティションキーバリューが生成されます。パーティションキーをランダム化しているため、毎日のテーブルへの書き込みは複数のパーティション間で均等に分散されます。これにより、並列性が強化され、全体的なスループットが向上します。

ただし、特定の日のすべての行を読み取るには、すべてのシャードの行をクエリしてから結果をマージする必要があります。例えば、まず、パーティションキーバリュー (2020-07-09, 1) に対して SELECT ステートメントを発行したとします。次に、(2020-07-09, 2) などから (2020-07-09, 200) までに対して別の SELECT ステートメントを発行します。最終的には、アプリケーションを使用して、これらすべての SELECT ステートメントの結果をマージする必要があります。

複合パーティションキーと計算値を使用したシャーディング

ランダム化の方法は、書き込みスループットを大幅に向上させることができます。しかし、行が書き込まれたときに shard 列にどの値が書き込まれたのかわからないため、特定の行の読み取りが難しくなります。個々の行の読み取りを容易にするために、別の戦略を使用することができます。パーティション間で行を分散させるには、乱数を使用せずに、クエリする項目に基づいて計算できる数値を使用します。

前の例では、パーティションキーで今日の日付が使用されています。各行にはアクセス可能な title 列があり、日付別に加えてタイトル別に行を頻繁に検索する必要があるとします。アプリケーションでは、テーブルに行が書き込まれる前に、タイトルに基づいてハッシュ値が計算され、それを使用して shard 列が入力されます。この計算では、1 から 200 までの数値が生成され、ランダムな方法と同様に、完全に均一に分散されます。

この計算はシンプルな計算で十分です (タイトルの文字の UTF-8 コードポイント値を乗算して、それを 200 で割った余りに + 1 するなど)。複合パーティションキーバリューは、日付と計算結果を組み合わせた値になります。

この方法を使用すると、書き込みがパーティションキーバリュー全体に均一に分散されます。したがって、物理パーティション全体にも均一に分散されます。これで、特定の title 値のパーティションキーバリューを計算できるため、特定の行と日付に対して簡単に SELECT ステートメントを実行できます。

特定の日のすべての行を読み取るには、(2020-07-09, N) キー (ここで N は 1〜200) を SELECT する必要があり、アプリケーションはすべての結果をマージする必要があります。この方法には、すべてのワークロードを利用する単一の「ホット」パーティションキーバリューを使用しないというメリットがあります。

Amazon Keyspaces テーブルのコスト最適化

このセクションでは、既存の Amazon Keyspaces テーブルのコストを最適化する方法のベストプラクティス説明します。次の戦略を見て、どのコスト最適化戦略がニーズに最も適しているかを確認し、それらに繰り返しアプローチする必要があります。各戦略では、コストに影響する可能性のある

要素、コストを最適化する機会を探す方法、節約に役立つこれらのベストプラクティスの実装方法に関する規範的なガイダンスの概要を示します。

トピック

- テーブルレベルでコストを評価する
- テーブルのキャパシティモードの評価
- テーブルの Application Auto Scaling 設定を評価する
- 未使用のリソースを特定して Amazon Keyspaces のコストを最適化する
- テーブルの使用パターンを評価してパフォーマンスとコストを最適化する
- 適切なサイズのプロビジョニングを行うために、プロビジョンドキャパシティを評価する

テーブルレベルでコストを評価する

内にある Cost Explorer ツール AWS Management Console を使用すると、読み取り、書き込み、ストレージ、バックアップ料金など、コストをタイプ別に分類して確認できます。また、これらのコストを月や日などの期間別にまとめて表示することもできます。

Cost Explorer のよくある課題の 1 つは、特定のテーブル単体のコストを簡単には確認できないことです。Cost Explorer では、特定のテーブルのコストだけをフィルタリングまたはグループ化して表示することができません。Amazon Keyspaces コンソールでは、テーブルごとの [請求可能なテーブルサイズ (バイト)] メトリクスを各テーブルの [モニタリング] タブで確認できます。テーブルごとのコスト関連情報がさらに必要な場合は、このセクションで紹介する方法でタグ付けを利用し、Cost Explorer で個々のテーブルのコスト分析を実行してください。

トピック

- 1 つの Amazon Keyspaces テーブルのコストを表示する方法
- Cost Explorer のデフォルトビュー
- Cost Explorer でのテーブルタグの使用方法および適用方法

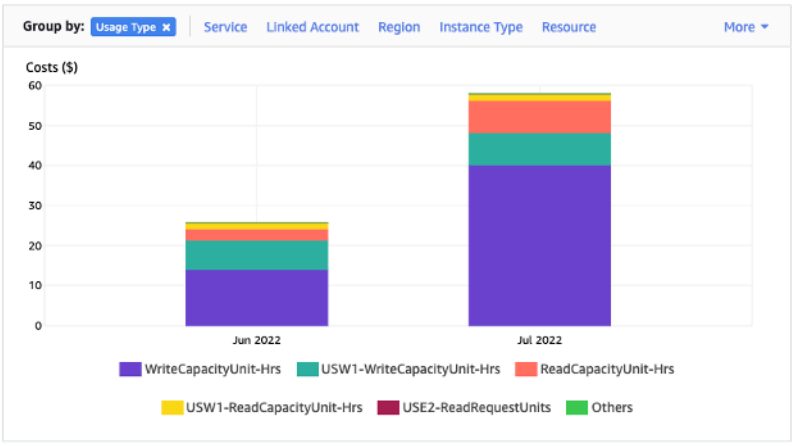
1 つの Amazon Keyspaces テーブルのコストを表示する方法

Amazon Keyspaces テーブルに関する基本情報 (プライマリキースキーマ、請求可能なテーブルサイズ、キャパシティ関連のメトリクスなど) はコンソールで確認できます。テーブルのサイズから、そのテーブルの月々のストレージコストを計算できます。たとえば、米国東部 (バージニア北部) では、GB あたり 0.25 USD AWS リージョンです。

テーブルがプロビジョンドキャパシティモードの場合、現在のリードキャパシティユニット (RCU) とライトキャパシティユニット (WCU) の設定も返します。この情報を基に、テーブルの現在の読み取りと書き込みのコストを計算できます。これらのコストは、テーブルに Amazon Keyspaces 自動スケーリングを設定している場合は特に、変動する可能性があります。

Cost Explorer のデフォルトビュー

Cost Explorer のデフォルトビューには、スループットやストレージなど、消費されたリソースのコストを示すグラフが表示されます。月ごとや日ごとの合計など、期間ごとにこれらのコストをグループ化することを選択できます。ストレージ、読み取り、書き込み、その他のカテゴリーのコストも分類して比較できます。



Cost Explorer でのテーブルタグの使用方法および適用方法

デフォルトでは、Cost Explorer は複数のテーブルのコストを合計するため、特定の 1 つのテーブルのコストの概要が表示されません。ただし、AWS リソースへのタグ付けを使用すると、メタデータタグで各テーブルを識別できます。タグとは、プロジェクトや部門に属するすべてのリソースの識別など、さまざまな目的に使用できるキーと値のペアです。詳細については、「the section called “タグの操作”」を参照してください。

この例では、MyTable という名前のテーブルを使用します。

1. table_name のキーと MyTable の値を使用してタグを設定します。
2. Cost Explorer 内でタグをアクティブ化し、タグ値でフィルタリングすると、各テーブルのコストをより明確に把握できます。

> **Note**
>
> タグが Cost Explorer に表示され始めるまでには、1～2 日かかる場合があります。

メタデータタグは、コンソールで自分で設定することも、CQL、、 AWS CLIまたは AWS SDK を使用してプログラムで設定することもできます。組織の新しいテーブル作成プロセスの一環として、table_name タグの設定を義務付けることを検討してください。詳細については、「the section called “コスト配分レポートの作成”」を参照してください。

テーブルのキャパシティモードの評価

このセクションでは、Amazon Keyspaces テーブルで、適切なキャパシティモードを選択する方法の概要を説明します。各モードは、スループットの変化に対する応答性や使用に対する請求方法という点で、さまざまなワークロードのニーズを満たすように調整されています。決定を下す際には、これらの要素のバランスを取る必要があります。

トピック

- 利用可能なテーブルキャパシティモード
- オンデマンドキャパシティモードを選択する場合
- プロビジョンドキャパシティモードを選択する場合
- テーブルキャパシティモードを選択する際に考慮すべきその他の要素

利用可能なテーブルキャパシティモード

Amazon Keyspaces テーブルを作成するとき合、オンデマンドキャパシティモードとプロビジョンドキャパシティモードのいずれかを選択する必要があります。詳細については、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。

オンデマンドキャパシティモード

オンデマンドキャパシティモードでは、Amazon Keyspaces テーブルのキャパシティをプランニングまたはプロビジョニングする必要がありません。このモードでは、リクエストにテーブルが即座に対応します。いかなるリソースもスケールアップやスケールダウンの必要はありません (テーブルの過去のピークスループットの最大 2 倍まで対応可能)。

オンデマンドテーブルは、テーブルに対する実際のリクエスト数をカウントして請求されるため、お支払いいただくのは、プロビジョニングされたリクエスト数ではなく、実際に使用したリクエスト分のみです。

プロビジョンドキャパシティモード

プロビジョンドキャパシティモードは従来型のモデルで、リクエストに対して利用できるテーブルのキャパシティを、直接、または Application Auto Scaling で定義できます。特定のキャパシティが指定された時間にテーブルにプロビジョニングされるため、請求はリクエスト数ではなく、プロビジョニングされたキャパシティに基づいて行われます。割り当てられたキャパシティを超えると、テーブルはリクエストを拒否し、アプリケーションのユーザーのエクスペリエンスが低下する可能性があります。

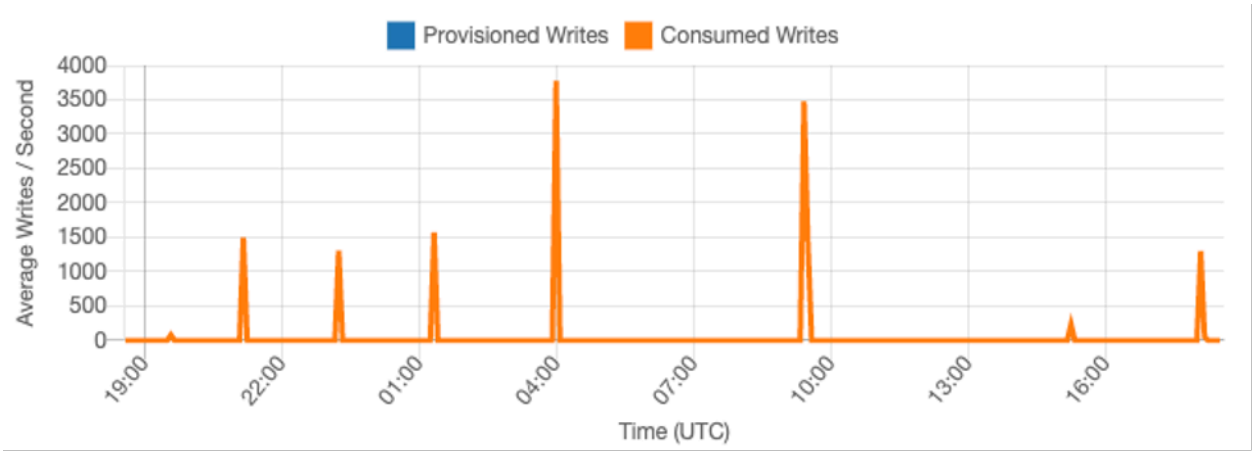
プロビジョンドキャパシティモードでは、テーブルをオーバープロビジョニングとアンダープロビジョニングの双方の間で調整して、スループットキャパシティエラーの発生率の軽減と、コスト最適化の両方を実現する必要があります。

オンデマンドキャパシティモードを選択する場合

コストを最適化する際、次のグラフに示すような想定外のワークロードがある場合は、オンデマンドモードが最適です。

この種のワークロードには、次のような要素が影響します。

- リクエストのタイミングが予測できない (トラフィックの急増につながる)
- リクエストの量が変動する (バッチワークロードに起因)
- 一定時間、ゼロまで低下する、またはピーク時の 18% を下回る (開発環境またはテスト環境に起因)



上記のような特性のワークロードでは、Application Auto Scaling を使用してテーブルがトラフィックの急増に対応できる十分なキャパシティを維持すると、望ましくない結果が生じるおそれがあります。その場合、テーブルの過剰なプロビジョニングによって必要以上のコストが発生するか、テーブルのプロビジョニングが不十分でリクエストに不必要な低キャパシティスループットエラーが発生するおそれがあります。このような場合は、オンデマンドテーブルの方が適しています。

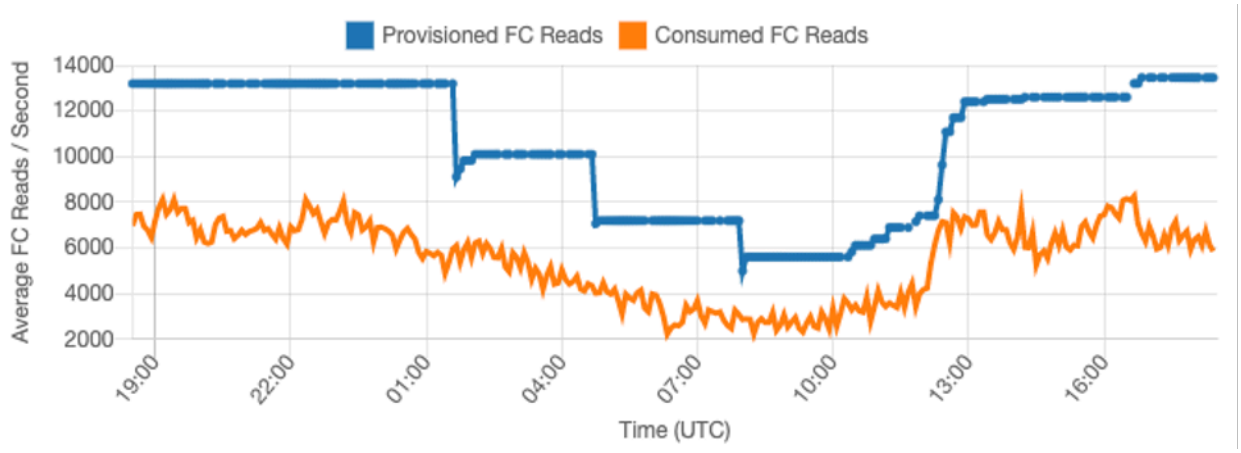
オンデマンドテーブルはリクエストに応じて請求されるため、コスト最適化のためにテーブルレベルでこれ以上の対応は不要です。オンデマンドテーブルを定期的に評価して、ワークロードに上記の現象が引き続き残っていることを確認してください。ワークロードが安定したら、コスト最適化のためにプロビジョンドモードへの変更を検討してください。

プロビジョンドキャパシティモードを選択する場合

プロビジョンドキャパシティモードに最適なワークロードは、以下のグラフに示すように、使用パターンを予測しやすい場合です。

予測可能なワークロードには、次のような要素があります。

- 特定の時間または 1 日のトラフィックが予測可能/周期的
- トラフィックの急増が短期間で限定的



一定時間内または 1 日のトラフィック量が安定しているため、プロビジョンドキャパシティは、テーブルで実際に消費されるキャパシティに比較的近い値に設定できます。プロビジョンドキャパシティテーブルのコストの最適化は、最終的には、テーブル上で `ThrottledRequests` イベントを増やさずに、プロビジョンドキャパシティ (青色の線) を消費キャパシティ (オレンジ色の線) にできるだけ近づける練習になります。2 つの線の間にある空白は、未使用のキャパシティであると同時に、スループットキャパシティ不足エラーによるユーザーエクスペリエンスの低下に備えた保険にもなります。

Amazon Keyspaces は、Application Auto Scaling を備えており、あなたに代わって自動的にプロビジョンドキャパシティテーブルのバランスを調整してくれます。1 日を通して消費されたキャパシティを追跡し、いくつかの変数に基づいてテーブルのプロビジョンドキャパシティを設定できます。

キャパシティユニットの最小数

テーブルの最小キャパシティを設定すると、スループットキャパシティ不足の エラーの発生を抑えることはできますが、これによってテーブルのコストが削減されるわけではありません。テーブルの使用量が少ない期間が続いた後に突然使用量が急増した場合、最小値を設定しておくと、Application Auto Scalingによって低すぎるテーブルキャパシティが設定されるのを防ぐことができます。

キャパシティユニットの最大数

テーブルの最大キャパシティを設定すると、テーブルが意図したより大きくスケールしてしまうことを制限できます。大規模な負荷テストが望ましくない開発テーブルやテストテーブルには最大値の適用を検討してください。最大数はどのテーブルにも設定できますが、本番環境で使用する場合は、不意のスループットキャパシティ不足エラーを防ぐため、この設定はテーブルのベースラインと照らし合わせて定期的に評価してください。

目標使用率

テーブルの目標使用率を設定することは、プロビジョンドキャパシティテーブルのコストを最適化するための主な手段です。ここでパーセント値を低く設定すると、テーブルでオーバープロビジョニングされるキャパシティ増により、コストが増加しますが、スループットキャパシティ不足エラーのリスクは軽減できます。パーセント値を高く設定すると、テーブルの過剰プロビジョニングの割合を軽減できますが、スループットキャパシティ不足エラーが発生するリスクが高まります。

テーブルキャパシティモードを選択する際に考慮すべきその他の要素

2 つのキャパシティモードのどちらかを選択する際には、考慮すべき点が他にもいくつかあります。

2 つのテーブルモードのどちらかを選択する際は、この追加の割引でテーブルのコストに与えられる影響を考慮してください。多くの場合、比較的予測が困難なワークロードでも、リザーブドキャパシティを利用すればオーバープロビジョニングされたプロビジョンドキャパシティテーブルで実行する方がコスト効率が良くなる場合があります。

ワークロードの予測可能性の向上

状況によっては、ワークロードに予測可能なパターンと予測不可能なパターンの両方があるように思われる場合があります。これはオンデマンドテーブルで簡単にサポートできますが、ワークロードの予測不可能なパターンを改善できれば、コストも低くなる可能性があります。

これらのパターンの最も一般的な原因の 1 つは、バッチインポートです。このタイプのトラフィックでは、ワークロードを実行するとスループットキャパシティ不足エラーが発生してしまうほど、テーブルのベースラインキャパシティを超えることがよくあります。このようなワークロードをプロビジョンドキャパシティテーブルで実行し続ける場合は、次のオプションを検討してください。

- 予定した時間にバッチが行われる場合は、バッチの実行前にアプリケーションオートスケーリングキャパシティを増やすようにスケジュールできます。
- バッチがランダムに行われる場合は、できるだけ速く実行することよりも、実行する時間を延長することを検討してください。
- Application Auto Scaling でテーブルキャパシティの調整を開始できるまで、低インポート速度で開始し、数分かけてゆっくりと速度を上げていくランプアップ時間をインポートに追加します。

テーブルの Application Auto Scaling 設定を評価する

このセクションでは、Amazon Keyspaces テーブルの Application Auto Scaling 設定を評価する方法を概説します。Amazon Keyspaces Application Auto Scaling は、アプリケーショントラフィックと目標使用率メトリクスに基づいてテーブルのスループットを管理する機能です。これにより、テーブルがアプリケーションパターンに必要なキャパシティを確保できます。

Application Auto Scaling サービスは現在のテーブル使用率をモニタリングし、目標使用率値である `TargetValue` と比較します。割り当てられたキャパシティを増減する時期になると、通知されます。

トピック

- Application Auto Scaling 設定の理解
- 目標使用率が低い (50% 未満) テーブルを特定する方法
- 季節変動のあるワークロードに対処する方法
- パターンが不明な急増するワークロードに対処する方法
- リンクされたアプリケーションのワークロードに対応する方法

Application Auto Scaling 設定の理解

目標使用率、初期ステップ、最終値の正しい値を定義するには、運用チームの関与が必要です。これにより、Application Auto Scaling ポリシーのトリガーに使用されるアプリケーションの使用履歴に基づいて値を適切に定義することができます。目標使用率は、Application Auto Scaling ルールが適用される前の期間に達成する必要がある合計キャパシティの割合です。

高い目標使用率 (90% 前後) を設定するときは、Application Auto Scaling が起動する前に、一定期間、トラフィックが 90% を超える必要があります。アプリケーションが常に一定で、トラフィックが急増しない場合を除いて、高い目標使用率を使用しないでください。

非常に低い使用率 (50% 未満) を設定する場合は、アプリケーションが Application Auto Scaling ポリシーをトリガーする前に、プロビジョニングされたキャパシティの 50% に達する必要があります。アプリケーショントラフィックが非常に速い速度で増加しない限り、これは通常、キャパシティの未使用およびリソースの浪費につながります。

目標使用率が低い (50% 未満) テーブルを特定する方法

AWS CLI または のいずれかを使用して AWS Management Console 、Amazon Keyspaces リソース内の Application Auto Scaling ポリシー`TargetValues`の をモニタリングおよび識別できます。

> **Note**
>
> プロビジョンドキャパシティモードのマルチリージョンテーブルで Amazon Keyspaces 自動スケーリングを有効にする場合は、Amazon Keyspaces の API オペレーションを使用して自動スケーリングを設定してください。Amazon Keyspaces で自動的に呼び出される基盤の Application Auto Scaling API オペレーションには、マルチリージョンの機能がありません。詳細については、「the section called “マルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定の表示”」を参照してください。

AWS CLI

1. リソースのリスト全体を返すには、次のコマンドを実行します。

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    cassandra
   ```

   このコマンドでは、すべての Amazon Keyspaces リソースに発行された Application Auto Scaling ポリシーのリスト全体が返ります。特定のテーブルからのみリソースを取得する場合は、`–resource-id parameter` を追加できます。例:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    cassandra --resource-id "keyspace/keyspace-name/table/table-name”
   ```

2. 特定のテーブルの Auto Scaling ポリシーのみを返すには、次のコマンドを実行します。

```
aws application-autoscaling describe-scaling-policies --service-namespace
 cassandra --resource-id "keyspace/keyspace-name/table/table-name”
```

Application Auto Scaling ポリシーで求める値は以下のとおりです。たとえば、オーバープロビジョニングを避けるため、目標値を 50% 以上に設定しようと考えているとします。次のような結果を取得します。

```
{
    "ScalingPolicies": [
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/keyspaces/table/table-name-scaling-policy",
            "PolicyName": $<full-gsi-name>”,
            "ServiceNamespace": "cassandra",
            "ResourceId": "keyspace/keyspace-name/table/table-name",
            "ScalableDimension": "cassandra:index:WriteCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "KeyspacesWriteCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:48.641000+10:00"
        },
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/keyspaces/table/table-name/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName":$<full-table-name>”,
            "ServiceNamespace": "cassandra",
            "ResourceId": "keyspace/keyspace-name/table/table-name",
            "ScalableDimension": "cassandra:index:ReadCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "CassandraReadCapacityUtilization"
```

```
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:47.820000+10:00"
        }
    ]
}
```

AWS Management Console

1. にログイン AWS Management Console し、「 の開始方法 AWS Management Console」の CloudWatch サービスページに移動します。 AWS リージョン 必要に応じて適切な を選択します。
2. 左のナビゲーションバーで、[Tables (テーブル)] を選択します。[Tables (テーブル)] ページで、テーブルの [Name (名)] を選択します。
3. [Capacity (キャパシティ)] タブの [Table Details (テーブルの詳細)] ページで、テーブルの Application Auto Scaling 設定を確認します。

目標使用率が 50% 以下の場合は、テーブルの使用率メトリクスを調べて、プロビジョニングが不十分か過剰かを確認する必要があります。

季節変動のあるワークロードに対処する方法

次のシナリオを考えてみましょう。アプリケーションはほとんどの場合最小平均値で動作していますが、目標使用率は低いため、アプリケーションは 1 日の特定の時間に発生するイベントに迅速に対応でき、十分なキャパシティがあり、スロットリングを回避できます。これは、アプリケーションが通常の勤務時間 (午前 9 時 ~ 午後 5 時) には非常にビジーで、勤務時間外には基本的なレベルで動作する場合に一般的なシナリオです。一部のユーザーは午前 9 時前に接続を開始するため、アプリケーションはこの低いしきい値を使用しますが、ピーク時には必要なキャパシティに対応して迅速に増加します。

このシナリオは次のようなものです。

- 午後 5 時から午前 9 時までの間、ConsumedWriteCapacityUnits ユニットの単位は 90 から 100 の間にとどまる

- ユーザーが午前 9 時前にアプリケーションに接続し始めると、キャパシティユニットが大幅に増加する (最大値は 1500 WCU)
- 勤務時間中のアプリケーション使用量は、平均して 800 ~ 1,200 の間で推移する

前述のシナリオがアプリケーションに当てはまる場合は、スケジュールした Application Auto Scaling の使用を検討してください。この場合、テーブルにはアプリケーションの Application Auto Scaling ルールを設定できますが、目標使用率はそれほど高くなく、必要な間隔で追加のキャパシティのみをプロビジョニングできます。

を使用して、以下のステップ AWS CLI を実行して、時刻と曜日に基づいて実行されるスケジュールされた自動スケーリングルールを作成できます。

1. Application Auto Scalingで Amazon Keyspaces テーブルをスケーラブルな目標として登録します。スケーラブルな目標は、 Application Auto Scaling によるスケールアウトやスケールインが可能なリソースです。

```
aws application-autoscaling register-scalable-target \
    --service-namespace cassandra \
    --scalable-dimension cassandra:table:WriteCapacityUnits \
    --resource-id keyspace/keyspace-name/table/table-name \
    --min-capacity 90 \
    --max-capacity 1500
```

2. 要件に従ってスケジュールされたアクションをセットアップします。

   このシナリオに対応するには、2 つのルールが必要です。1 つはスケールアップ用、もう 1 つはスケールダウン用です。スケジュールしたアクションをスケールアップするための最初のルールを次の例で示します。

```
aws application-autoscaling put-scheduled-action \
    --service-namespace cassandra \
    --scalable-dimension cassandra:table:WriteCapacityUnits \
    --resource-id keyspace/keyspace-name/table/table-name \
    --scheduled-action-name my-8-5-scheduled-action \
    --scalable-target-action MinCapacity=800,MaxCapacity=1500 \
    --schedule "cron(45 8 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

   スケジュールされたアクションをスケールダウンするための 2 つ目のルールを次の例で示します。

```
aws application-autoscaling put-scheduled-action \
    --service-namespace cassandra \
    --scalable-dimension cassandra:table:WriteCapacityUnits \
    --resource-id keyspace/keyspace-name/table/table-name \
    --scheduled-action-name my-5-8-scheduled-down-action \
    --scalable-target-action MinCapacity=90,MaxCapacity=1500 \
    --schedule "cron(15 17 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

3. 次のコマンドを実行して、両方のルールがアクティブになっていることを確認します。

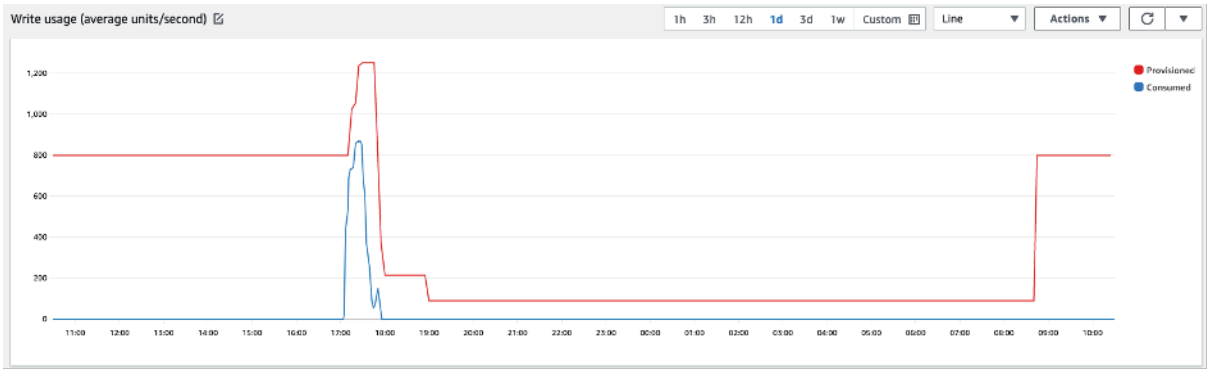
```
aws application-autoscaling describe-scheduled-actions --service-namespace
 cassandra
```

次のような結果が表示されます。

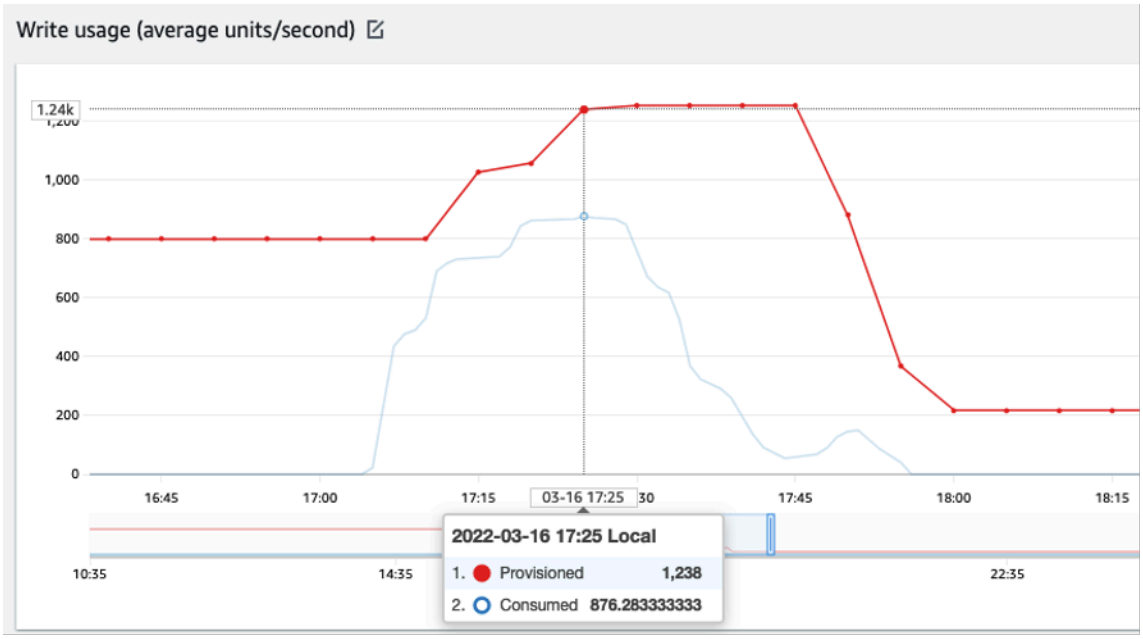
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/keyspaces/
table/table-name:scheduledActionName/my-5-8-scheduled-down-action",
            "ServiceNamespace": "cassandra",
            "Schedule": "cron(15 17 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "keyspace/keyspace-name/table/table-name",
            "ScalableDimension": "cassandra:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 90,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:30:25.100000+10:00"
        },
        {
            "ScheduledActionName": "my-8-5-scheduled-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/keyspaces/
table/table-name:scheduledActionName/my-8-5-scheduled-action",
            "ServiceNamespace": "cassandra",
            "Schedule": "cron(45 8 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
```

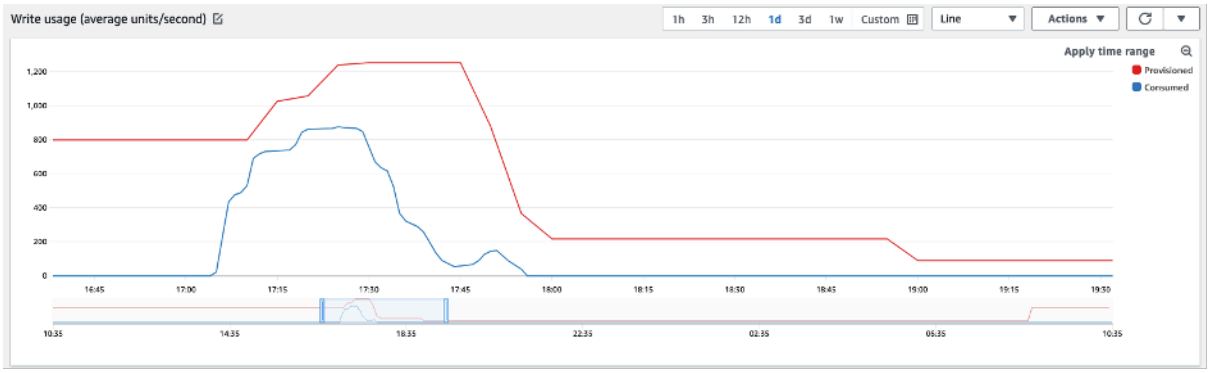
```
            "ResourceId": "keyspace/keyspace-name/table/table-name",
            "ScalableDimension": "cassandra:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 800,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:28:57.816000+10:00"
        }
    ]
}
```

次の図は、常に 70% の目標使用率を維持するサンプルワークロードを示しています。オートスケーリングルールが適用され、スループットが低下していないことがわかりますか。



拡大すると、アプリケーションにスパイクが発生して Auto Scaling のしきい値が 70% に達し、Auto Scaling が強制的に開始され、テーブルに必要な追加キャパシティが提供されたことがわかります。スケジュールされた自動スケーリングアクションは最大値と最小値に影響するため、それらの値を設定しておくのはユーザーの責任です。





パターンが不明な急増するワークロードに対処する方法

このシナリオでは、アプリケーションのパターンがまだわからないため、アプリケーションの目標使用率が非常に低く、ワークロードに低いキャパシティスループットエラーが生じないようにする必要があります。

代わりにオンデマンドキャパシティモードを使用することを検討してください。オンデマンドテーブルは、トラフィックパターンが不明で急増するワークロードに最適です。オンデマンドキャパシティモードでは、アプリケーションがテーブルに対して実行するデータの読み取りと書き込みに対して、リクエストごとに料金を支払います。Amazon Keyspaces はワークロードの増減に即座に対応するため、アプリケーションに期待する読み取りと書き込みのスループットを指定する必要はありません。

リンクされたアプリケーションのワークロードに対応する方法

このシナリオでは、アプリケーションは他のシステムに依存します。例えば、バッチ処理シナリオでは、アプリケーションロジックのイベントに応じてトラフィックが大幅に急増する可能性があります。

特定のニーズに応じてテーブルのキャパシティと `TargetValues` を増やすことができる、これらのイベントに反応するカスタムの Application Auto Scaling ロジックの開発を検討してください。Λ Amazon EventBridge や Step Functions などの AWS サービスを組み合わせて使用することで、特定のアプリケーションのニーズに対応できます。

未使用のリソースを特定して Amazon Keyspaces のコストを最適化する

このセクションでは、未使用のリソースを定期的に評価する方法の概要を説明します。アプリケーションの要件が変化に対応できるよう、未使用のリソースの使用によって不要な Amazon Amazon Keyspaces のコストが生じないようにする必要があります。以下に説明する手順では、Amazon CloudWatch メトリクスで未使用のリソースを特定し、コストを削減する措置を講じます。

Amazon CloudWatch で Amazon Keyspaces をモニタリングすると、raw データが収集され、Amazon Keyspaces からほぼリアルタイムで読み取り可能なメトリクスに加工することができます。これらの統計は一定期間保持されるため、履歴情報にアクセスして使用率をより正確に調べることができます。デフォルトでは、 Amazon Keyspaces メトリクスデータは CloudWatch に自動的に送信されます。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch とは」および「メトリクスの保持」を参照してください。

トピック

- 未使用のリソースを特定する方法
- 未使用のテーブルリソースを特定する
- 未使用のテーブルリソースをクリーンアップする
- 未使用のポイントインタイムリカバリ (PITR) バックアップのクリーンアップ

未使用のリソースを特定する方法

未使用のテーブルを特定するには、現在から30 日間の CloudWatch メトリクスを調べ、特定のテーブルに対してアクティブな読み込みまたは書き込みがあるかどうかを調べます。

`ConsumedReadCapacityUnits`

指定された期間に消費された読み込みキャパシティユニットの数。消費されたキャパシティの使用量を追跡できます。テーブルの読み取り消費キャパシティの合計を取得できます。

`ConsumedWriteCapacityUnits`

指定された期間に消費された書き込みキャパシティユニットの数。消費されたキャパシティの使用量を追跡できます。テーブルの書き込み消費キャパシティの合計を取得できます。

未使用のテーブルリソースを特定する

Amazon CloudWatch は、モニタリングサービスとオブザーバビリティサービスです。Amazon CloudWatch では、未使用のリソースを特定できる Amazon Keyspaces テーブルのメトリクスが得られます。CloudWatch メトリクスは、 AWS Management Console と AWS Command Line Interfaceを使って表示できます。

AWS Command Line Interface

を使用してテーブルメトリクスを表示するには AWS Command Line Interface、次のコマンドを使用できます。

1. まず、テーブルの読み込みを評価します。

> **Note**
>
> そのテーブル名がアカウント内で一意でない場合は、キー空間名も指定する必要があります。

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/Cassandra --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

テーブルが未使用と誤って識別されないようにするには、長期間にわたってメトリクスの評価を行います。たとえば、30 日間と、適切な開始時刻と終了時刻の範囲、および 86400 などで適切な期間を選択します。

返されるデータで、[合計] が 0 を超える場合は、評価対象のテーブルがその期間に受信した読み込みトラフィックを示します。

次の結果は、評価期間中に読み込みトラフィックを受信したテーブルを示しています。

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

次の結果は、評価期間中に読み込みトラフィックを受信しなかったテーブルを示しています。

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. 次に、テーブルの書き込みを評価します。

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedWriteCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/Cassandra --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

テーブルが未使用と誤って識別されないようにするには、長期間にわたってメトリクスを評価する必要があります。30 日間などの適切な開始時刻と終了時刻の範囲、および 86400 などの適切な期間を選択します。

返されるデータで、[Sum (合計)] が 0 を超える場合は、評価対象のテーブルがその期間に受信した読み込みトラフィックを示します。

次の結果は、評価期間中に書き込みトラフィックを受信したテーブルを示しています。

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

次の結果は、評価期間中に書き込みトラフィックを受信しなかったテーブルを示しています。

```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

AWS Management Console

次のステップでは、 AWS Management Consoleでリソースの使用率を評価します。

1. にログイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ の CloudWatch サービスページに移動します。必要に応じて、コンソールの右上 AWS リージョン にある適切な を選択します。
2. 左のナビゲーションバーで、[メトリクス] セクションを探し、[すべてのメトリクス] を選択します。
3. これにより、2 つのパネルで構成されるダッシュボードが開きます。上部のパネルには、現在グラフ化されているメトリクスが表示されます。下部のパネルでは、グラフ化できるメトリクスを選択できます。下部のパネルで Amazon Keyspaces を選択します。

4. Amazon Keyspaces メトリクスの選択パネルで [テーブルメトリクス] カテゴリを選択し、現在のリージョンのテーブルのメトリクスを表示します。
5. メニューを下にスクロールしてテーブル名を確認し、テーブルの `ConsumedReadCapacityUnits` と `ConsumedWriteCapacityUnits` メトリクスを選択します。
6. [グラフ化したメトリクス (2)] タブを選択し、[統計] 列を [合計] に設定します。
7. テーブルが誤って未使用と識別されないように、長期間にわたってテーブルメトリクスの評価を行います。グラフパネルの上部で、テーブルを評価するための適切な時間枠 (1 か月など) を選択します。[カスタム] を選択し、ドロップダウンで [1 か月間] を選択し、[適用] を選択します。
8. テーブルのグラフ化されたメトリクスを評価して、使用されているかどうかを判断します。0 を超えるメトリクスは、評価期間中にテーブルが使用されたことを示します。読み込みと書き込みの両方が 0 の平らなグラフは、テーブルが未使用であることを示します。

未使用のテーブルリソースをクリーンアップする

未使用のテーブルリソースを特定したら、次の方法でその継続的なコストを削減できます。

> **Note**
>
> 未使用のテーブルを特定したものの、将来アクセスする必要が生じた場合に備えて使用できるようにしておきたい場合は、オンデマンドモードに切り替えることを検討してください。それ以外の場合、テーブルの削除を検討してください。

キャパシティモード

Amazon Keyspaces テーブル内のデータの読み込み、書き込み、保存には料金がかかります。

Amazon Keyspaces には 2 種類のキャパシティモードがあり、テーブルの読み書き処理については特定の請求オプション (オンデマンドとプロビジョンド) があります。読み取り/書き込みキャパシティモードは、読み取りおよび書き込みスループットの課金方法とキャパシティの管理方法を制御します。

オンデマンドモードのテーブルでは、アプリケーションで実行することが予測される読み込みおよび書き込みスループットを指定する必要はありません。Amazon Keyspaces では、読み込みリクエスト単位と書き込みリクエスト単位に関して、テーブルに対してアプリケーションが実行する読み込み

と書き込みに料金が請求されます。テーブルにアクティビティがない場合、スループットに対する支払いは発生しませんが、ストレージ料金は発生します。

テーブルの削除

未使用のテーブルを発見し、それを削除する場合は、まずデータのバックアップか、エクスポートを行うことを検討してください。

を介してバックアップを実行すると、コールドストレージ階層化を活用 AWS Backup できるため、コストをさらに削減できます。ライフサイクルを使用してバックアップをコールドストレージに移動する方法については、バックアッププランの管理ドキュメントを参照してください。

テーブルをバックアップしたら、 AWS Management Console または AWS Command Line Interface を使用してテーブルを削除できます。

未使用のポイントインタイムリカバリ (PITR) バックアップのクリーンアップ

Amazon Keyspaces にはポイントインタイムリカバリ機能があり、35 日間の継続的バックアップを利用できるため、意図しない書き込みや削除を防ぐことができます。PITR バックアップには、関連する追加コストがあります。

不要になった可能性のあるバックアップがテーブルで有効になっているかどうかを、the section called “ポイントインタイムリカバリによるバックアップと復元” のマニュアルを参照して確認してください。

テーブルの使用パターンを評価してパフォーマンスとコストを最適化する

このセクションでは、あなたがAmazon Keyspaces テーブルを効率的に使用しているかどうかを評価する方法の概要を説明します。Amazon Keyspaces には最適ではない特定の使用パターンがあり、これらはパフォーマンスとコストの両方の観点から最適化する余地があります。

トピック

- 強力な整合性のある読み込みオペレーションの数を減らす
- 有効期限 (TTL) の有効化

強力な整合性のある読み込みオペレーションの数を減らす

Amazon Keyspaces では、リクエストごとに読み込み整合性を設定できます。デフォルトでは、読み込みリクエストは結果的に整合性が保たれます。結果整合性のある読み込みは、最大 4 KB のデータに対して 0.5 RCU が課金されます。

分散型ワークロードのほとんどの部分は柔軟性があり、最終的な一貫性を許容できます。ただし、強力な整合性のある読み込みを必要とするアクセスパターンもあり得ます。強力な整合性のある読み込みには、最大 4 KB のデータに対して 1 RCU の料金が課金されるため、読み込みコストは実質的に 2 倍になります。Amazon Keyspaces では、同じテーブルで両方の整合性モデルを柔軟に使用できます。

ワークロードとアプリケーションコードを評価して、強力な整合性のある読み込みが必要な場合にのみ使用されているかどうかを確認できます。

有効期限 (TTL) の有効化

有効期限 (TTL) を使用すると、テーブルのデータが自動的に期限切れになり、アプリケーションロジックが簡素化され、ストレージの料金を最適化できます。不要になったデータは、設定した有効期限の値に基づいて、テーブルから自動的に削除されます。

適切なサイズのプロビジョニングを行うために、プロビジョンドキャパシティを評価する

このセクションでは、Amazon Keyspaces テーブルのプロビジョニングが適切なサイズであるかどうかを評価する方法の概要を説明します。ワークロードの変化に応じて、運用手順を適切に変更する必要があります。特に Amazon Keyspaces テーブルがプロビジョニングモードで設定されていて、テーブルの過剰なプロビジョニングや、プロビジョニング不足のリスクがある場合はそうです。

このセクションで説明する手順では、本番アプリケーションをサポートしている Amazon Keyspaces テーブルから取得すべき統計情報が必要です。アプリケーションの動作を理解するには、アプリケーションから得られるデータの季節性を把握するのに十分な期間を定義する必要があります。たとえば、アプリケーションに週単位のパターンが見られる場合は、3 週間の期間を設定すれば、アプリケーションのスループットニーズ分析に十分な余裕が得られます。

何から始めればよいかわからない場合は、以下の計算に少なくとも 1 か月分のデータ使用量を使用してください。

キャパシティを評価する際、Amazon Keyspaces テーブルでは、読み込みキャパシティユニット (RCU) と書き込みキャパシティユニット (WCU) を個別に設定できます。

トピック

- Amazon Keyspaces テーブルから消費メトリクスを取得する方法

- プロビジョニング不足の Amazon Keyspaces テーブルを識別する方法
- 過剰にプロビジョニングされた Amazon Keyspaces テーブルを識別する方法

Amazon Keyspaces テーブルから消費メトリクスを取得する方法

テーブルキャパシティを評価するには、次の CloudWatch メトリクスをモニタリングし、適切なディメンションを選択してテーブル情報を取得します。

読み込みキャパシティユニット	書き込みキャパシティユニット
ConsumedReadCapacityUnits	ConsumedWriteCapacityUnits
ProvisionedReadCapacityUnits	ProvisionedWriteCapacityUnits
ReadThrottleEvents	WriteThrottleEvents

これを行うには、 AWS CLI または を使用します AWS Management Console。

AWS CLI

テーブル消費メトリクスを取得する前に、まず CloudWatch API を使用して履歴データポイントを取得する必要があります。

まず、write-calc.json と read-calc.json の 2 つのファイルを作成します。これらのファイルは、テーブルの計算を表します。下の表に示すように、一部のフィールドを環境に合わせて更新する必要があります。

> **Note**
>
> そのテーブル名がアカウント内で一意でない場合は、キー空間名も指定する必要があります。

フィールド名	定義	例
<table-name>	分析するテーブルの名前	サンプルテーブル

フィールド名	定義	例
<period>	目標使用率の評価に使用する期間 (秒単位)	1 時間の場合は 3600 と指定します
<start-time>	ISO8601 形式で指定された評価間隔の開始	2022-02-21T23:00:00
<end-time>	ISO8601 形式で指定された評価間隔の終了	2022-02-22T06:00:00

書き込み計算ファイルに、指定された日付範囲の期間にプロビジョニングされた WCU と消費された WCU の数が取得されます。また、分析に使用できる使用率も生成されます。write-calc.json ファイルの完全な内容は次の例のようになります。

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/Cassandra",
          "MetricName": "ProvisionedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/Cassandra",
```

```
          "MetricName": "ConsumedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>""
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedWCU/PERIOD(consumedWCU)",
      "Label": "Consumed WCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedWCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

読み込み計算ファイルにも同様のメトリクスを使用します。このファイルには、指定された日付範囲にプロビジョニングされた RCU と、消費された RCU の数が取得されます。read-calc.json ファイルの内容は以下のようになります。

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedRCU",
      "MetricStat": {
        "Metric": {
```

```
          "Namespace": "AWS/Cassandra",
          "MetricName": "ProvisionedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/Cassandra",
          "MetricName": "ConsumedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedRCU/PERIOD(consumedRCU)",
      "Label": "Consumed RCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedRCU)",
      "Label": "Utilization Percentage",
```

```
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

ファイルを作成したら、使用率データの取得を開始できます。

1. 書き込み使用率データを取得するには、次のコマンドを実行します。

```
aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
```

2. 読み込み使用率データを取得するには、次のコマンドを実行します。

```
aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
```

両方のクエリの結果、分析に使用できる JSON 形式の一連のデータポイントが得られます。結果は、指定したデータポイントの数、期間、および特定のワークロードデータによって異なります。場合によっては、次の例のようになります。

```
{
    "MetricDataResults": [
        {
            "Id": "utilizationPercentage",
            "Label": "Utilization Percentage",
            "Timestamps": [
                "2022-02-22T05:00:00+00:00",
                "2022-02-22T04:00:00+00:00",
                "2022-02-22T03:00:00+00:00",
                "2022-02-22T02:00:00+00:00",
                "2022-02-22T01:00:00+00:00",
                "2022-02-22T00:00:00+00:00",
                "2022-02-21T23:00:00+00:00"
            ],
            "Values": [
                91.55364583333333,
                55.066631944444445,
                2.6114930555555556,
```

```
                24.9496875,
                40.94725694444445,
                25.61819444444444,
                0.0
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

> **Note**
>
> 短い期間の範囲と長時間の範囲を指定する場合、デフォルトで 24 に設定されているスクリプトの `MaxDatapoints` 値を変更する必要があるかもしれません。これは、1 時間あたり 1 データポイント、1 日あたり 24 データポイントに相当します。

AWS Management Console

1. にログイン AWS Management Console し、 の開始方法 AWS Management Consoleの CloudWatch サービスページに移動します。 AWS リージョン 必要に応じて適切な を選択します。
2. 左のナビゲーションバーの [メトリクス] セクションで、[すべてのメトリクス] を選択します。
3. これにより、2 つのパネルで構成されたダッシュボードが開きます。上のパネルにはグラフィックが表示され、下のパネルにはグラフ化するメトリクスが表示されます。Amazon Keyspaces パネルを選択します。
4. サブパネルから [テーブルメトリクス] カテゴリを選択します。現在の のテーブルが表示されます AWS リージョン。
5. メニューを下にスクロールしてテーブル名を確認し、`ConsumedWriteCapacityUnits` および `ProvisionedWriteCapacityUnits` の書き込みオペレーションメトリクスを選択します。

> **Note**
>
> この例では書き込みオペレーションメトリクスについて説明していますが、これらの手順を使用して読み込みオペレーションメトリクスをグラフ化することもできます。

6. 数式を変更するには、[Graphed metrics (2)] (グラフ化したメトリクス (2)) タブを選択します。デフォルトでは、CloudWatch はグラフの統計関数 [Average (平均)] を選択します。
7. グラフ化された両方のメトリクスを選択した状態 (左側のチェックボックス) で、[Add math (算術の追加)] メニュー、[Common (共通)] の順に選択し、次に [Percentage (パーセンテージ)] 関数を選択します。この手順を 2 回繰り返します。

   [Percentage (パーセンテージ)] 関数を初めて選択する場合:

   [Percentage (パーセンテージ)] 関数を 2 回目に選択する場合:
8. この時点で、下部のメニューに 4 つのメトリクスが表示されているはずです。`ConsumedWriteCapacityUnits` の計算に取り掛かりましょう。一貫性を保つには、AWS CLI セクションで使用した名前と一致する必要があります。[m1 ID] をクリックし、この値を [ConsumedWCU] に変更します。
9. 統計を Average から Sum に変更します。このアクションにより、ANOMALY_DETECTION_BAND という名前の別のメトリクスが自動的に作成されます。この手順の範囲は、新しく生成された [ad1 メトリクス] のチェックボックスをオフにすれば無視できます。
10. 手順 8 を繰り返して m2 ID の名前を ProvisionedWCU に変更します。統計は [Average (平均)] に設定したままにします。
11. [Expression1] ラベルを選択し、値を [m1] に更新し、ラベルを [Consumed WCUs] に更新します。

> **Note**
>
> データを正しく視覚化するには、必ず [m1] (左側のチェックボックス) と [ProvisionedWCU] のみを選択してください。[Details (詳細)] をクリックし、数式を [consumedWCU/PERIOD(consumedWCU)] に変更して、数式を更新します。このステップで別の [ANOMALY_DETECTION_BAND] メトリクスを生成することもありますが、この手順の範囲では無視できます。

12. これで 2 つのグラフィックができたはずです。1 つはテーブル上にあなたがプロビジョニングした WCU を反映し、もう 1 つは消費された WCU を反映します。
13. Expression2 グラフィック (e2) を選択して、パーセンテージ数式を更新します。ラベルと ID の名前を utilizationPercentage に変更します。100*(m1/provisionedWCU) と一致するように数式の名前を変更します。
14. 使用率パターンを視覚化するには、utilizationPercentage 以外のすべてのメトリクスのチェックボックスをオフにします。デフォルトの間隔は 1 分に設定されていますが、必要に応じて自由に変更できます。

得られる結果は、ワークロードの実際のデータによって異なります。使用率が 100% を超える間隔では、スループットキャパシティエラーイベントが発生しやすくなります。Amazon Keyspaces にはバーストキャパシティ機能がありますが、バーストキャパシティがなくなると、100% を超えるものはすべて低スループットキャパシティエラーイベントが発生します。

プロビジョニング不足の Amazon Keyspaces テーブルを識別する方法

ほとんどのワークロードでは、テーブルでプロビジョンドキャパシティの 80% 以上が絶えず消費されている場合、そのテーブルはプロビジョニング不足と見なされます。

バーストキャパシティは Amazon Keyspaces の機能の 1 つで、当初のプロビジョニング量よりも多くの (表で定義されている 1 秒あたりのプロビジョニングスループットを超える) RCU/WCU を一時的に消費できるようにするものです。バーストキャパシティは、特別なイベントや使用量の急増によるトラフィックの急激な増加を吸収するために作成されました。このバーストキャパシティには制限があります。詳細については、「the section called “バーストキャパシティの使用”」を参照してください。未使用の RCU と WCU がなくなってから、プロビジョニングされたキャパシティよりも多くのキャパシティを消費しようとすると、低キャパシティスループットエラーイベントが発生する場合があります。アプリケーショントラフィックの使用率が 80% に近づくと、低キャパシティスループットエラーイベントのリスクが大幅に高まります。

80% 使用率ルールは、データの季節性やトラフィックの増加によって異なります。次のシナリオを考えてみます。

- 過去 12 か月間、トラフィックの使用率が約 90% で安定していれば、テーブルのキャパシティは適切であると言えます
- アプリケーショントラフィックが 3 か月以内に毎月 8% の割合で増加した場合、100% に到達します

- アプリケーショントラフィックが 4 か月余りで 5% の割合で増加している場合でも、100% に到達します

上記のクエリの結果から、使用率の全体像がわかります。これらを参考にして、必要に応じてテーブルのキャパシティを増やす方法を選択するのに役立つ他のメトリクス (月間または毎週の増加率など) をさらに評価してください。運用チームと協力して、ワークロードとテーブルに適したパーセンテージを定義してください。

データを毎日または毎週分析すると、データに偏りが生じる特別なシナリオがあります。たとえば、季節性のアプリケーションで、勤務時間中に使用量が急増する (ただし、勤務時間外はほぼゼロになる) 場合は、Appliceation Auto Scaling をスケジュールして時間帯 (および曜日) を指定してプロビジョンドキャパシティを増やすと効果的です。季節性がそれほど顕著でない場合は、繁忙期に対応できるようにキャパシティを増やすかわりに、Amazon Keyspaces テーブルの Auto Scaling 設定を利用することもできます。

過剰にプロビジョニングされた Amazon Keyspaces テーブルを識別する方法

上記のスクリプトから取得したクエリ結果から、初期分析を実行するために必要なデータポイントが得られます。データセットの使用率が複数の間隔で 20% 未満の値を示す場合は、テーブルが過剰にプロビジョニングされている可能性があります。WCU と RCU の数を減らす必要があるかどうかをさらに明確にするには、その間隔で他の測定値を見直す必要があります。

テーブルに使用頻度の低い間隔が複数ある場合は、Application Auto Scaling をスケジュールするか、使用率に基づくテーブルのデフォルトの Application Auto Scaling ポリシーを設定すると、Application Auto Scaling ポリシーを有効活用できます。

使用率が低いのにスロットル率 (間隔内の Max(ThrottleEvents)/Min(ThrottleEvents)) が高いワークロードがある場合、これは、特定の日 (または一日の時刻) にトラフィックが大幅に増加しても、それ以外は一般的にトラフィックが常に少ない、非常にスパイクの多いワークロードの存在時に発生することがあります。このようなシナリオでは、スケジュールした Application Auto Scaling を使用すると有益な場合があります。

Amazon Keyspaces (Apache Cassandra 向け) のトラブルシューティング

このガイドでは、Amazon Keyspaces (Apache Cassandra 向け) の使用時のさまざまなシナリオを取り上げ、問題のトラブルシューティング手順を説明します。一般的なエラー、接続の問題、キャパシティ管理の問題、データ定義言語 (DDL) エラーの解決に関する情報を紹介します。

- 一般的なエラー
  - `NoNodeAvailableException`、`NoHostAvailableException`、`AllNodesFailedException` など、上位の例外をトラブルシューティングする。
  - Java ドライバー例外の根本原因のエラーを特定する。
  - 再試行ポリシーを実装し、接続を正しく設定する。
- 接続の問題
  - `cqlsh` または Apache Cassandra クライアントドライバーを使用した Amazon Keyspaces エンドポイントへの接続時に起きるエラーを解決する。
  - VPC エンドポイント接続、Cassandra-stress 接続、IAM 設定のエラーをトラブルシューティングする。
  - データインポート中の接続損失に対処する。
- キャパシティ管理エラー
  - テーブル、パーティション、接続に関連するキャパシティ不足エラーを把握し、解決する。
  - 関連する Amazon Keyspaces メトリクスを Amazon CloudWatch Logs でモニタリングする。
  - 接続とスループットを最適化してパフォーマンスを向上させる。
- データ定義言語 (DDL) エラー
  - キースペースとテーブルの作成時、アクセス時、または復元時のエラーをトラブルシューティングする。
  - カスタムの Time to Live (TTL) 設定、列の制限、範囲の削除に関連する障害に対処する。
  - 大量の削除ワークロードに関する考慮事項。

IAM アクセスに関するトラブルシューティングガイダンスについては、「the section called “トラブルシューティング”」を参照してください。セキュリティに関する詳しいベストプラクティスについては、「the section called “セキュリティに関するベストプラクティス”」を参照してください。

トピック

- Amazon Keyspaces の一般的なエラーのトラブルシューティング
- Amazon Keyspaces の接続エラーのトラブルシューティング
- Amazon Keyspaces のキャパシティ管理エラーのトラブルシューティング
- Amazon Keyspaces のデータ定義言語エラーのトラブルシューティング

Amazon Keyspaces の一般的なエラーのトラブルシューティング

一般的なエラーが発生していませんか。以下は、一般的なシナリオとその解決方法です。

一般的なエラー

次のいずれかの上位の例外が発生している場合、さまざまな原因が考えられます。

- `NoNodeAvailableException`
- `NoHostAvailableException`
- `AllNodesFailedException`

これらはクライアントドライバーが生成する例外であり、コントロール接続の確立時や、読み取り/書き込み/準備/実行/バッチリクエストの実行時に発生する可能性があります。

コントロール接続の確立中にエラーが起きた場合は、アプリケーションで指定されたすべての接続先に到達できないということです。読み取り/書き込み/準備/実行クエリの実行中にエラーが起きれば、それは、そのリクエストが再試行回数をすべて使い果たしたということです。デフォルトの再試行ポリシーを使用している場合、再試行のたびに別のノードで試行されます。

上位の Java ドライバー例外の根本原因のエラーを特定する方法

これらの一般的なエラーは、接続に問題がある場合や、読み取り/書き込み/準備/実行のオペレーションの実行時に発生することがあります。分散型システムでは一時的な障害が想定されるため、リクエストを再試行することで対処する必要があります。接続エラーの発生時に、Java ドライバーは自動では再試行しません。そのため、アプリケーションでドライバー接続を確立する際は、再試行ポリシーを実装することを推奨します。接続のベストプラクティスの詳細については、「the section called “Connections”」を参照してください。

デフォルトでは、Java ドライバーはすべてのリクエストに対して `idempotence` (べき等性) を false に設定します。つまり、読み取り/書き込み/準備のリクエストが失敗しても、Java ドライバーは自動

的には再試行しません。idempotence を true に設定して、リクエストが失敗した場合は再試行するようにドライバーに指示する方法は、いくつかあります。ここでは、Java アプリケーションで単一のリクエストに対してプログラムでべき等性を設定する一例を紹介します。

```
Statement s = new SimpleStatement("SELECT * FROM my_table WHERE id = 1");
s.setIdempotent(true);
```

または、次の例に示すように、Java アプリケーション全体のデフォルトのべき等性をプログラムで設定できます。

```
// Make all statements idempotent by default:
cluster.getConfiguration().getQueryOptions().setDefaultIdempotence(true);
//Set the default idempotency to true in your Cassandra configuration
basic.request.default-idempotence = true
```

また、アプリケーションレベルで再試行ポリシーを作成する方法もお勧めします。この場合、アプリケーションで NoNodeAvailableException をキャッチしてリクエストを再試行する必要があります。再試行は 10 回行い、エクスポネンシャルバックオフを使用して、10 ミリ秒から始めて最長 100 ミリ秒まで延ばし、すべての再試行の合計時間を 1 秒に収めることを推奨します。

もう 1 つの選択肢は、Java ドライバー接続の確立時に Amazon Keyspaces のエクスポネンシャルバックオフ再試行ポリシーを適用することです。このポリシーは Github で公開されています。

デフォルトの再試行ポリシーを使用する場合は、複数のノードへの接続が確立されていることを確認してください。それには、Amazon Keyspaces で次のクエリを使用します。

```
SELECT * FROM system.peers;
```

このクエリの応答が空の場合、Amazon Keyspaces で単一のノードを使用しているということです。デフォルトの再試行ポリシーを使用している場合、再試行は行われません。デフォルトの再試行は必ず別のノードで行われるためです。VPC エンドポイントを介した接続の確立の詳細については、「the section called “VPC エンドポイント接続”」を参照してください。

Datastax 4.x Cassandra ドライバーを使用して Amazon Keyspaces への接続を確立する方法を説明したステップバイステップのチュートリアルについては、「the section called “Java 4.x の認証プラグイン”」を参照してください。

Amazon Keyspaces の接続エラーのトラブルシューティング

接続に問題がありませんか。以下は、一般的なシナリオとその解決方法です。

Amazon Keyspaces エンドポイントへの接続中のエラー

接続が失敗し、接続エラーが発生すると、さまざまなエラーメッセージが表示される可能性があります。次のセクションでは、最も一般的なシナリオを取り上げます。

トピック

- cqlsh を使用して Amazon Keyspaces に接続できない
- Cassandra クライアントドライバーを使用して Amazon Keyspaces に接続することができない

cqlsh を使用して Amazon Keyspaces に接続できない

cqlsh を使用して Amazon Keyspaces エンドポイントに接続しようとすると、**`Connection error`** が発生して接続が失敗します。

cqlsh が正しく設定されていない場合、Amazon Keyspaces テーブルに接続しようとすると接続は失敗します。次のセクションでは、cqlsh を使用して接続の確立を試みた時に接続エラーが発生するという最も一般的な設定問題の例を示します。

> **Note**
>
> VPC から Amazon Keyspaces に接続しようとする場合は、追加のアクセス権限が必要です。VPC エンドポイントを使用して接続を正常に設定するには、the section called “VPC エンドポイントによる接続” の手順に従ってください。

cqlsh を使って Amazon Keyspaces に接続しようとすると、接続の **`timed out`** エラーが発生します。

これは正しいポートを指定しなかった場合に発生する可能性がある問題で、発生した場合は次のエラーが表示されます。

```
#  cqlsh cassandra.us-east-1.amazonaws.com 9140 -u "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.199': error(None,
 "Tried connecting to [('3.234.248.199', 9140)]. Last error: timed out")})
```

この問題を解決するには、接続にポート 9142 を使用しているか確認してください。

cqlsh を使ってAmazon Keyspaces に接続しようとすると、**`Name or service not known`** というエラーが発生します。

これは、使用しているエンドポイントのスペルが間違っている場合、またはそのエンドポイントが存在しない場合に発生する可能性がある問題です。以下は、エンドポイント名のスペルが間違っている場合の例です。

```
#  cqlsh cassandra.us-east-1.amazon.com 9142 -u "USERNAME" -p "PASSWORD" --ssl
Traceback (most recent call last):
  File "/usr/bin/cqlsh.py", line 2458, in >module>
    main(*read_options(sys.argv[1:], os.environ))
  File "/usr/bin/cqlsh.py", line 2436, in main
    encoding=options.encoding)
  File "/usr/bin/cqlsh.py", line 484, in __init__
    load_balancing_policy=WhiteListRoundRobinPolicy([self.hostname]),
  File "/usr/share/cassandra/lib/cassandra-driver-internal-only-3.11.0-bb96859b.zip/
cassandra-driver-3.11.0-bb96859b/cassandra/policies.py", line 417, in __init__
socket.gaierror: [Errno -2] Name or service not known
```

パブリックエンドポイントを使用して接続している場合、この問題を解決するには、the section called “サービスエンドポイント” から使用可能なエンドポイントを選択し、そのエンドポイントの名前に問題がないことを確認してください。VPC エンドポイントを使用して接続する場合は、cqlsh 設定の VPC エンドポイント情報が正しいことを確認してください。

cqlsh を使用して Amazon Keyspaces に接続しようとすると、**OperationTimedOut** というエラーが発生します。

Amazon Keyspaces では、接続時の強力なセキュリティを確保するために、SSL を有効にする必要があります。SSL パラメータが見つからない場合は、次のエラーが表示されることがあります。

```
# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD"
Connection error: ('Unable to connect to any servers', {'3.234.248.192':
 OperationTimedOut('errors=Timed out creating connection (5 seconds),
 last_host=None',)})
#
```

この問題を解決するには、cqlsh 接続コマンドに次のフラグを追加してください。

```
--ssl
```

cqlsh を使用して Amazon Keyspaces に接続しようとすると、**SSL transport factory requires a valid certfile to be specified** というエラーが発生します。

この場合、SSL/TLS 証明書へのパスが欠落しているため、次のエラーが発生します。

```
# cat .cassandra/cqlshrc
[connection]
port = 9142
factory = cqlshlib.ssl.ssl_transport_factory
#


# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD" --ssl
Validation is enabled; SSL transport factory requires a valid certfile to be specified.
 Please provide path to the certfile in [ssl] section as 'certfile' option in /
root/.cassandra/cqlshrc (or use [certfiles] section) or set SSL_CERTFILE environment
 variable.
#
```

この問題を解決するには、コンピュータの certfile へのパスを追加します。

```
certfile =  path_to_file/sf-class2-root.crt
```

cqlsh を使用して Amazon Keyspaces に接続しようとすると、**No such file or directory** というエラーが発生します。

これは、コンピュータの証明書ファイルへのパスが間違っている場合に発生する可能性がある問題で、発生した場合は次のエラーが表示されます。

```
# cat .cassandra/cqlshrc
[connection]
port = 9142
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
validate = true
certfile = /root/wrong_path/sf-class2-root.crt
#



# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.192': IOError(2, 'No
 such file or directory')})
#
```

この問題を解決するには、コンピュータの certfile へのパスが正しいか確認してください。

cqlsh を使用して Amazon Keyspaces に接続しようとすると、**[X509] PEM lib** というエラーが発生します。

これは SSL/TLS 証明書ファイル sf-class2-root.crt が無効である場合に発生する可能性がある問題で、発生した場合は次のエラーが表示されます。

```
# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.241':
 error(185090057, u"Tried connecting to [('3.234.248.241', 9142)]. Last error: [X509]
 PEM lib (_ssl.c:3063)")})
#
```

この問題を解決するには、次のコマンドを使用して Starfield デジタル証明書をダウンロードしてください。sf-class2-root.crt をローカルまたはホームディレクトリに保存してください。

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

cqlsh を使用して Amazon Keyspaces に接続しようとすると、**unknown** SSL というエラーが発生します。

これは SSL/TLS 証明書ファイル sf-class2-root.crt が空である場合に発生する可能性がある問題で、発生した場合は次のエラーが表示されます。

```
# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.220': error(0,
 u"Tried connecting to [('3.234.248.220', 9142)]. Last error: unknown error
 (_ssl.c:3063)")})
#
```

この問題を解決するには、次のコマンドを使用して Starfield デジタル証明書をダウンロードしてください。sf-class2-root.crt をローカルまたはホームディレクトリに保存してください。

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

cqlsh を使用して Amazon Keyspaces に接続しようとすると、**SSL: CERTIFICATE_VERIFY_FAILED** というエラーが発生します。

これは SSL/TLS 証明書ファイルを検証できなかった場合に発生する可能性がある問題で、発生した場合は次のエラーが表示されます。

```
Connection error: ('Unable to connect to any servers', {'3.234.248.223':
 error(1, u"Tried connecting to [('3.234.248.223', 9142)]. Last error: [SSL:
 CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:727)")})
```

この問題を解決するには、次のコマンドを使用して証明書ファイルをダウンロードしてください。`sf-class2-root.crt` をローカルまたはホームディレクトリに保存してください。

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

cqlsh を使用して Amazon Keyspaces に接続しようとすると、**`Last error: timed out`** というエラーが発生します。

これは、Amazon EC2 セキュリティグループで Amazon Keyspaces のアウトバウンドルールを設定しなかった場合に発生する可能性がある問題で、発生した場合は次のエラーが表示されます。

```
# cqlsh cassandra.us-east-1.amazonaws.com 9142 -u  "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.206': error(None,
 "Tried connecting to [('3.234.248.206', 9142)]. Last error: timed out")})
#
```

この問題の原因が ではなく Amazon EC2 インスタンスの設定であることを確認するには`cqlsh`、 を使用してキースペースへの接続を試行します。例えば AWS CLI、次のコマンドを使用します。

```
aws keyspaces list-tables --keyspace-name 'my_keyspace'
```

このコマンドもタイムアウトする場合は、Amazon EC2 インスタンスが正しく設定されていません。

Amazon Keyspaces にアクセスするための十分なアクセス許可があることを確認するには、 AWS CloudShell を使用して に接続できます`cqlsh`。これで接続が確立される場合は、Amazon EC2 インスタンスの設定が必要です。

この問題を解決するには、Amazon Keyspaces への接続を許可するアウトバウンドルールが Amazon EC2 インスタンスにあるかどうかを確認します。ない場合は、EC2 インスタンスの新しいセキュリティグループを作成し、Amazon Keyspaces リソースへのアウトバウンドトラフィックを許可するルールを追加する必要があります。Amazon Keyspaces へのトラフィックを許可するようにアウトバウンドルールを更新するには、[タイプ] ドロップダウンメニューから [CQLSH/CASSANDRA] を選択してください。

アウトバウンドトラフィックルールを追加した新しいセキュリティグループを作成したら、そのグループをインスタンスに追加する必要があります。インスタンスを選択し、[アクション]、[セキュリティ]、[セキュリティグループを変更] の順に選択します。アウトバウンドルール付きの新しいセキュリティグループを追加しますが、デフォルトのグループも引き続き使用可能にしておいてください。

EC2 アウトバウンドルールの表示方法と編集方法の詳細については、「Amazon EC2 ユーザーガイド」の「セキュリティグループへのルールの追加」を参照してください。

cqlsh を使用して Amazon Keyspaces に接続しようとすると、**Unauthorized** というエラーが発生します。

これは、IAM ユーザーポリシーで Amazon Keyspaces のアクセス許可が欠如している場合に発生する可能性がある問題で、発生した場合は次のエラーが表示されます。

```
# cqlsh cassandra.us-east-1.amazonaws.com 9142 -u  "testuser-at-12345678910" -p
 "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.241':
 AuthenticationFailed('Failed to authenticate to 3.234.248.241: Error from server:
 code=2100 [Unauthorized] message="User arn:aws:iam::12345678910:user/testuser has no
 permissions."',)})
#
```

この問題を解決するには、IAM ユーザー `testuser-at-12345678910` が Amazon Keyspaces へのアクセス許可を取得しているか確認してください。Amazon Keyspaces へのアクセスを許可する IAM ポリシーの例については、「the section called “アイデンティティベースのポリシーの例”」を参照してください。

IAM アクセスに関するトラブルシューティングガイダンスについては、「the section called “トラブルシューティング”」を参照してください。

cqlsh を使用して Amazon Keyspaces に接続しようとすると、**Bad credentials** というエラーが発生します。

これは、ユーザー名またはパスワードが間違っている場合に発生する可能性がある問題で、発生した場合は次のエラーが表示されます。

```
# cqlsh cassandra.us-east-1.amazonaws.com 9142 -u  "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.248':
 AuthenticationFailed('Failed to authenticate to 3.234.248.248: Error from server:
```

```
 code=0100 [Bad credentials] message="Provided username USERNAME and/or password are
 incorrect"',)})
#
```

この問題を解決するには、コードの *USERNAME* (ユーザー名) と *PASSWORD* (パスワード) が、サービス固有の認証情報を生成したときに取得したユーザー名とパスワードと一致しているか確認してください。

> **Important**
>
> cqlsh を使用して接続を試みると引き続きエラーが表示される場合は、--debug オプションでコマンドを再実行し、 サポートへの連絡時に詳細な出力を含めてください。

Cassandra クライアントドライバーを使用して Amazon Keyspaces に接続することができない

次のセクションでは、Cassandra クライアントドライバーとの接続時に発生する最も一般的なエラーを示します。

DataStax Java ドライバーを使用して Amazon Keyspaces テーブルに接続しようとすると、**NodeUnavailableException** というエラーが発生します。

リクエストの試行に使用した接続が切断されると、次のエラーが発生します。

```
[com.datastax.oss.driver.api.core.NodeUnavailableException: No connection
 was available to Node(endPoint=vpce-22ff22f2f22222fff-aa1bb234.cassandra.us-
west-2.vpce.amazonaws.com/11.1.1111.222:9142, hostId=1a23456b-
c77d-8888-9d99-146cb22d6ef6, hashCode=123ca4567)]
```

この問題を解決するには、ハートビート値を確認し、値が 30 秒よりも長い場合は短くします。

```
advanced.heartbeat.interval = 30 seconds
```

次に、関連するタイムアウトを確認し、値が 5 秒以上に設定されていることを確認します。

```
advanced.connection.init-query-timeout = 5 seconds
```

ドライバーと SigV4 プラグインを使用して Amazon Keyspaces に接続しようとすると、**AttributeError** というエラーが発生します。

認証情報が正しく設定されていない場合は、次のエラーが発生します。

```
cassandra.cluster.NoHostAvailable: ('Unable to connect to any servers',
 {'44.234.22.154:9142': AttributeError("'NoneType' object has no attribute
 'access_key'")})
```

この問題を解決するには、SigV4 プラグインの使用時に IAM ユーザーまたはロールに関連付けられた認証情報を渡すことになっているか確認します。SigV4 プラグインには次の認証情報が必要です。

- AWS_ACCESS_KEY_ID – IAM ユーザーまたはロールに関連付けられた AWS アクセスキーを指定します。
- AWS_SECRET_ACCESS_KEY - アクセスキーに関連付けられるシークレットキーを指定します。これは、基本的にアクセスキーの「パスワード」です。

アクセスキーと SigV4 プラグインの詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

ドライバーを使用して Amazon Keyspaces テーブルに接続しようとすると、**PartialCredentialsError** というエラーが発生します。

AWS_SECRET_ACCESS_KEY が欠落している場合、次のエラーが発生する可能性があります。

```
cassandra.cluster.NoHostAvailable: ('Unable to connect to any servers',
 {'44.234.22.153:9142':
 PartialCredentialsError('Partial credentials found in config-file, missing:
 aws_secret_access_key')})
```

この問題を解決するには、SigV4 プラグインを使用する場合に AWS_ACCESS_KEY_ID と AWS_SECRET_ACCESS_KEY の両方を渡すことになっているか確認してください。アクセスキーと SigV4 プラグインの詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

ドライバーを使用して Amazon Keyspaces テーブルに接続しようとすると、**Invalid signature** というエラーが発生します。

署名に必要なコンポーネントのいずれかが誤っているか、セッションに対して正しく定義されていないことが原因として考えられます。

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- AWS_DEFAULT_REGION

次のエラーは、無効なアクセスキーの例です。

```
cassandra.cluster.NoHostAvailable: ('Unable to connect to any servers',
 {'11.234.11.234:9142':
 AuthenticationFailed('Failed to authenticate to 11.234.11.234:9142: Error from server:
 code=0100
 [Bad credentials] message="Authentication failure: Invalid signature"')})
```

この問題を解決するには、アクセスキーと AWS リージョン が、SigV4 プラグインが Amazon Keyspaces にアクセスするように正しく設定されていることを確認します。アクセスキーと SigV4 プラグインの詳細については、「the section called " AWS 認証用の IAM 認証情報を作成する"」を参照してください。

VPC エンドポイント接続が正しく機能しない

VPC エンドポイントを使用して Amazon Keyspaces に接続しようとすると、トークンマップエラーが発生するか、スループットが低下します。

これは、VPC エンドポイント接続が正しく設定されていない場合に発生する可能性があります。

この問題を解決するには、以下の設定の詳細を確認してください。Amazon Keyspaces 用のインターフェイス VPC エンドポイントを介して接続を設定する方法に関する詳しいチュートリアルについては、「the section called "VPC エンドポイントによる接続"」を参照してください。

1. Amazon Keyspaces への接続に使用する IAM エンティティに、次の例に示すようにユーザーテーブルへの読み取り/書き込みアクセス権と、システムテーブルへの読み取りアクセス権があることを確認します。

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

```
            "cassandra:Modify"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

2. Amazon Keyspaces への接続に使用する IAM エンティティに、次の例に示すように、Amazon EC2 インスタンスの VPC エンドポイント情報にアクセスするために必要な読み取り許可があることを確認します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListVPCEndpoints",
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcEndpoints"
         ],
         "Resource":"*"
      }
   ]
}
```

> **Note**
>
> 管理ポリシー AmazonKeyspacesReadOnlyAccess_v2 と AmazonKeyspacesFullAccess には、Amazon Keyspaces が Amazon EC2 インスタンスにアクセスして、使用可能なインターフェイス VPC エンドポイントに関する情報を読み取るためのアクセス権限が設定されています。

VPC エンドポイントの詳細については、「the section called “Amazon Keyspaces 用インターフェイス VPC エンドポイントの使用”」を参照してください。

3. Java ドライバーの SSL 設定が、この例のようにホスト名の検証を false に設定していることを確認します。

```
hostname-validation = false
```

エージェント設定の詳細については、「the section called “ステップ 2: ドライバーを設定する”」を参照してください。

4. VPC エンドポイントが正しく設定されていることを確認するには、VPC 内から次のステートメントを実行します。

> **Note**
>
> ローカルの開発者環境や Amazon Keyspaces CQL エディタを使用してこの設定を確認することはできません。これらはパブリックエンドポイントを使用するためです。

```
SELECT peer FROM system.peers;
```

出力はこの例のようになります。VPC のセットアップと AWS リージョンに応じて、プライベート IP アドレスを持つ 2~6 個のノードが返されます。

```
peer
---------------
 192.0.2.0.15
 192.0.2.0.24
 192.0.2.0.13
 192.0.2.0.7
 192.0.2.0.8

(5 rows)
```

cassandra-stress を使用して接続することができない

cassandra-stress コマンドを使用して Amazon Keyspaces に接続しようとすると、**SSL context** というエラーが発生します。

これは、Amazon Keyspaces に接続しようとしたのに trustStore が正しく設定されていない場合に発生します。Amazon Keyspaces では、クライアントとの安全な接続を確保するために Transport Layer Security (TLS) を使用する必要があります。

この場合は次のエラーが表示されます。

```
Error creating the initializing the SSL Context
```

この問題を解決するには、このトピック the section called “[開始する前に]” の指示に従って trustStore をセットアップしてください。

trustStore の設定が完了すると、次のコマンドを使用して接続できるようになります。

```
./cassandra-stress user profile=./profile.yaml n=100 "ops(insert=1,select=1)"
 cl=LOCAL_QUORUM -node "cassandra.eu-north-1.amazonaws.com" -port native=9142
 -transport ssl-alg="PKIX" truststore="./cassandra_truststore.jks" truststore-
password="trustStore_pw" -mode native cql3 user="user_name" password="password"
```

IAM アイデンティティを使用して接続することができない

IAM アイデンティティを使用して Amazon Keyspaces に接続しようとすると、**Unauthorized** というエラーが発生します。

これは、ポリシーを実装せず、ユーザーに必要なアクセス許可を付与していない状態で、IAM アイデンティティ (IAM ユーザーなど) を使用して Amazon Keyspaces テーブルに接続しようとした場合に発生する問題です。

この場合は次のエラーが表示されます。

```
Connection error: ('Unable to connect to any servers', {'3.234.248.202':
 AuthenticationFailed('Failed to authenticate to 3.234.248.202:
Error from server: code=2100 [Unauthorized] message="User
 arn:aws:iam::1234567890123:user/testuser has no permissions."',)})
```

この問題を解決するには、IAM ユーザーのアクセス許可を確認してください。ほとんどのドライバーで接続が構築されるときにシステムのキースペース/テーブルが読み取られるため、標準的なドライバーで接続する場合には、最低条件として、ユーザーはそのシステムテーブルへの SELECT アクセス権を取得しておく必要があります。

例えば、Amazon Keyspaces システムおよびユーザーテーブルへのアクセスを許可する IAM ポリシーについては、「the section called “Amazon Keyspaces テーブルへのアクセス”」を参照してください。

IAM に特化したトラブルシューティングセクションを確認するには、「the section called “トラブルシューティング”」を参照してください。

cqlsh を使ってデータをインポートしようとすると、Amazon Keyspaces テーブルへの接続が失われる

cqlsh を使用して Amazon Keyspaces にデータをアップロードしようとすると、接続エラーが発生します。

サーバーから cqlsh クライアントに何らかの種類のエラーが 3 回連続で送信されると、Amazon Keyspaces への接続が失敗します。cqlsh クライアントで処理が失敗すると、次のメッセージが表示されます。

```
Failed to import 1 rows: NoHostAvailable - , will retry later, attempt 3 of 100
```

このエラーを解決するには、インポートするデータが Amazon Keyspaces のテーブルスキーマと一致していることを確認する必要があります。インポートファイルで解析エラーが発生していないか確認してください。INSERT ステートメントを使用してエラーを切り離すことで、1 行のデータの使用を試すことができます。

クライアントにより接続の再確立が自動的に試行されます。

Amazon Keyspaces のキャパシティ管理エラーのトラブルシューティング

サーバーレスキャパシティに関して問題がありますか。以下は、一般的なシナリオとその解決方法です。

サーバーレス容量エラー

このセクションでは、サーバーレスキャパシティ管理に関連するエラーの認識方法と解決方法について説明します。例えば、アプリケーションがプロビジョンドスループット性能を超えると、キャパシティ不足イベントが発生することがあります。

Apache Cassandra は、ノード群で実行するように設計されたクラスタベースのソフトウェアであるため、スループットキャパシティなどのサーバーレス機能に関連する例外メッセージはありませ

ん。ほとんどのドライバーでは Apache Cassandra で利用可能なエラーコードしか認識されないため、Amazon Keyspaces では、互換性を維持するために同一のエラーコードセットを使用します。

基盤となるキャパシティイベントに Cassandra エラーをマッピングする場合には、Amazon CloudWatch を使用して関連する Amazon Keyspaces メトリクスを監視することができます。クライアント側のエラーを発生させるキャパシティ不足イベントは、イベントの原因となっているリソースに基づいて、次の 3 つのグループに分類できます。

- テーブル – テーブルに対して [Provisioned] (プロビジョンド) キャパシティモードを選択した場合に、アプリケーションがプロビジョンドスループットを超えると、キャパシティ不足エラーが発生する可能性があります。詳細については、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。
- パーティション — 指定されたパーティションのトラフィックが 3,000 RCU または 1,000 WCU を超えると、キャパシティ不足イベント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、パーティション間でトラフィックを均一に分散させることをお勧めします。詳細については、「the section called “データモデリング”」を参照してください。
- 接続 — 接続 1 つあたりの 1 秒あたりの最大オペレーション数のクォータを超えると、スループットが不十分になる可能性があります。スループットを高めるために、ドライバーによる接続を設定する際に、デフォルト接続の数を増やすことができます。

  Amazon Keyspaces で接続を設定する方法については、「the section called “接続を設定する方法”」を参照してください。VPC エンドポイントを利用した接続の最適化の詳細については、「the section called “VPC エンドポイント接続”」を参照してください。

クライアント側エラーを返すキャパシティ不足イベントの原因となっているリソースを特定するために、Amazon Keyspaces コンソールでダッシュボードを確認できます。デフォルトでは、コンソールには、テーブルの [Capacity] (キャパシティ) の [Capacity and related metrics] (キャパシティと関連メトリクス) セクションに、最も一般的なキャパシティとトラフィックに関する CloudWatch メトリクスの集約ビューがあります。

Amazon CloudWatch を使用して独自のダッシュボードを作成するには、次の Amazon Keyspaces メトリクスを確認します。

- `PerConnectionRequestRateExceeded` — 接続ごとのリクエストレートのクォータを超える Amazon Keyspaces へのリクエスト。Amazon Keyspaces への各クライアント接続は、1 秒あたり最大 3000 の CQL リクエストに対応できます。複数の接続を作成すれば、1 秒あたり 3000 を超えるリクエストを実行できます。

- `ReadThrottleEvents` — テーブルの読み取りキャパシティを超える Amazon Keyspaces へのリクエスト。
- `StoragePartitionThroughputCapacityExceeded` — パーティションのスループットキャパシティを超える Amazon Keyspaces ストレージパーティションへのリクエスト。Amazon Keyspaces ストレージパーティションは、最大 1000 WCU/WRU/秒および最大 3000 RCU/RRU/秒に対応できます。これらの例外を軽減するために、データモデルを確認して、読み取り/書き込みトラフィックをより多くのパーティションに分散させることをお勧めします。
- `WriteThrottleEvents` — テーブルの書き込みキャパシティを超える Amazon Keyspaces へのリクエスト。

CloudWatch の詳細については、「the section called “CloudWatch によるモニタリング”」を参照してください。Amazon Keyspaces で利用可能なすべての CloudWatch メトリクスの一覧については、「the section called “メトリクスとディメンション”」を参照してください。

> **Note**
>
> Amazon Keyspaces でよく見られるすべてのメトリクスを表示したカスタムダッシュボードの使用を開始する場合、AWS サンプル リポジトリの GitHub で使用できるようにあらかじめ構築されている CloudWatch テンプレートを使用できます。

トピック

- クライアントドライバーから NoHostAvailable というキャパシティ不足エラーが送られてくる
- データのインポート中に書き込みタイムアウトエラーが表示される
- キースペースまたはテーブルの実際のストレージサイズが表示されない

クライアントドライバーから **NoHostAvailable** というキャパシティ不足エラーが送られてくる

テーブルに対して **Read_Timeout** または **Write_Timeout** という例外が表示されています。

キャパシティが不足している Amazon Keyspaces テーブルに対して書き込みや読み取りを繰り返し試みると、ドライバー固有のクライアント側エラーが発生する可能性があります。

CloudWatch を使用して、プロビジョンドスループットメトリクス、実際のスループットメトリクス、テーブルのキャパシティ不足イベントを監視してください。例えば、十分なス

ループットキャパシティがない読み取りリクエストは、Read_Timeout 例外が発生して失敗し、ReadThrottleEvents メトリクスに投稿されます。十分なスループットキャパシティがない書き込みリクエストは、Write_Timeout 例外が発生して失敗し、WriteThrottleEvents メトリクスに投稿されます。これらのメトリクスの詳細については、「the section called “メトリクスとディメンション”」を参照してください。

この問題を解決するには、以下のオプションのいずれかを検討してください。

- テーブルのプロビジョンドスループットを増やす。これは、アプリケーションにより消費されるスループットキャパシティの最大量です。詳細については、「the section called “読み取りキャパシティユニットと書き込みキャパシティユニット”」を参照してください。
- オートスケーリングを使用してサービスによってスループットキャパシティが管理されるようにします。詳細については、「the section called “自動スケーリングによるスループットキャパシティの管理”」を参照してください。
- テーブルに対して [On-demand] (オンデマンド) キャパシティモードを選択します。詳細については、「the section called “オンデマンドキャパシティモードを設定する”」を参照してください。

アカウントのデフォルトのキャパシティクォータを増やす必要がある場合は、「クォータ」を参照してください。

パーティションキャパシティの超過に関連するエラーが表示されています。

StoragePartitionThroughputCapacityExceeded エラーが表示される場合は、パーティションのキャパシティが一時的に超過しています。この場合、アダプティブキャパシティまたはオンデマンドキャパシティによって自動的に処理される可能性があります。これらのエラーを軽減するために、データモデルを見直し、読み取り/書き込みトラフィックをより多くのパーティションに分散させることをお勧めします。Amazon Keyspaces ストレージパーティションは、最大 1000 WCU/WRU/秒、最大 3000 RCU/RRU/秒に対応できます。データモデルを改良して読み取り/書き込みトラフィックをより多くのパーティションに分散させる方法の詳細については、「the section called “データモデリング”」を参照してください。

Write_Timeout 例外は、同じ論理パーティションに静的データと非静的データを含める同時書き込みオペレーションのレートが上昇した場合にも発生する可能性があります。トラフィックにおいて、同じ論理パーティション内に静的データと非静的データを含める複数の同時書き込みオペレーションが実行されることが予想される場合、静的データと非静的データを別々に書き込むことをお勧めします。データを別々に書き込むと、スループットコストの最適化にも役立ちます。

接続リクエストレートの超過に関連するエラーが表示されています。

`PerConnectionRequestRateExceeded` が表示される場合は、次のいずれかが原因です。

- セッションごとに設定された接続数が不足している可能性があります。
- VPC エンドポイントのアクセス許可が正しく設定されていないと、使用可能なピアよりも接続数が少なくなる可能性があります。VPC エンドポイントポリシーの詳細については、「the section called “Amazon Keyspaces 用インターフェイス VPC エンドポイントの使用”」を参照してください。
- 4.x ドライバーを使用している場合は、ホスト名の検証が有効になっているかどうかを確認してください。ドライバーでは、デフォルトで TLS ホスト名の検証が有効になっています。この設定により、Amazon Keyspaces がシングルノードクラスターとしてドライバーに表示されます。ホスト名の検証をオフにすることをお勧めします。

接続とスループットが最適化されるように、次のベストプラクティスに従うことをお勧めします。

- CQL クエリのスループットチューニングを設定します。

  Amazon Keyspaces は、TCP 接続 1 つにつき 1 秒あたり最大 3,000 の CQL クエリに対応していますが、ドライバーが確立できる接続数に制限はありません。

  ほとんどのオープンソース Cassandra ドライバーで、Cassandra への接続プールが確立され、その接続プールでクエリのロードバランスが行われます。Amazon Keyspaces では 9 つのピア IP アドレスをドライバーに開示します。ほとんどのドライバーのデフォルト動作は、各ピア IP アドレスに対して 1 つの接続を確立することです。したがって、デフォルト設定を使用するドライバーの最大 CQL クエリスループットは、1 秒あたり 27,000 CQL クエリになります。

  この数を増やすには、ドライバーが接続プールで維持している IP アドレスあたりの接続数を増やすことをお勧めします。例えば、IP アドレスあたりの最大接続数を 2 に設定すると、ドライバーの最大スループットが 1 秒あたり 54,000 CQL クエリの 2 倍になります。
- 単一ノード接続を最適化します。

  ほとんどのオープンソース Cassandra ドライバーでは、デフォルトで、セッションの確立時に `system.peers` テーブルでアドバタイズされた各 IP アドレスへに対して、1 つ以上の接続が確立されます。ただし、設定によっては、ドライバーにより単一の Amazon Keyspaces IP アドレスに接続されることがあります。これは、ドライバーにより、ピアノードの SSL ホスト名の検証が試行されている場合や (DataStax Java ドライバーなど)、VPC エンドポイントを通じて接続されている場合に発生します。

複数の IP アドレスに接続するドライバーと同じ可用性とパフォーマンスを確保するには、次の操作の実行をお勧めします。

- 希望するクライアントのスループットに応じて、各 IP の接続数を 9 以上に増やす。
- 同じノードに対して再試行が実行されるようにするカスタム再試行ポリシーを作成します。詳細については「」を参照してください。

  the section called “再試行ポリシーを設定する方法”.
- VPC エンドポイントを使用する場合は、Amazon Keyspaces への接続に使用する IAM エンティティに、エンドポイントとネットワークインターフェイスの情報について VPC をクエリするためのアクセス許可を付与します。これにより、ロードバランシングが改善され、読み取り/書き込みスループットが向上します 詳細については、「???」を参照してください。

データのインポート中に書き込みタイムアウトエラーが表示される

`cqlsh COPY` コマンドを使用してデータをアップロードしているときに、タイムアウトエラーが表示されます。

```
Failed to import 1 rows: WriteTimeout - Error from server: code=1100 [Coordinator node
 timed out waiting for replica nodes' responses]
 message="Operation timed out - received only 0 responses." info={'received_responses':
 0, 'required_responses': 2, 'write_type': 'SIMPLE', 'consistency':
 'LOCAL_QUORUM'}, will retry later, attempt 1 of 100
```

Amazon Keyspaces では、スループットキャパシティ不足により書き込みリクエストが失敗した場合に、`ReadTimeout` 例外と `WriteTimeout` 例外を使用してその失敗が示されます。キャパシティ不足の例外を診断するために、Amazon Keyspaces は Amazon CloudWatch で次のメトリクスを公開しています。

- `WriteThrottleEvents`
- `ReadThrottledEvents`
- `StoragePartitionThroughputCapacityExceeded`

データロード中に発生したキャパシティ不足エラーを解決するには、ワーカーあたりの書き込みレートまたは総取り込みレートを引き下げ、行のアップロードを再試行してください。詳細については、「the section called “ステップ 4: `cqlsh COPY FROM` を設定する”」を参照してください。より堅牢なデータアップロードオプションについては、DSBulk の使用を検討してください。DSBulk

は GitHub リポジトリから入手できます。手順については、「the section called “DSBulk を使用したデータのロード”」を参照してください。

キースペースまたはテーブルの実際のストレージサイズが表示されない

キースペースまたはテーブルの実際のストレージサイズが表示されません。

テーブルのストレージサイズの詳細については、「the section called “テーブルレベルでコストを評価する”」を参照してください。テーブル内の行サイズを計算することから始めて、ストレージサイズを見積もることもできます。行サイズの計算に関する詳しい手順については、「the section called “行サイズの推定”」を参照してください。

Amazon Keyspaces のデータ定義言語エラーのトラブルシューティング

リソースの作成に問題がありますか。以下は、一般的なシナリオとその解決方法です。

データ定義言語エラー

Amazon Keyspaces では、キースペースとテーブルの非同期的な作成や削除など、データ定義言語 (DDL) オペレーションが同期なしで実行されます。アプリケーションの準備が整っていない時点でアプリケーションによりリソースの使用が試行されると、そのオペレーションは失敗します。

で新しいキースペースとテーブルの作成ステータスをモニタリングできます。これは AWS Management Console、キースペースまたはテーブルが保留中またはアクティブであることを示します。システムスキーマテーブルのクエリを実行して、新しいキースペースまたはテーブルの作成ステータスをプログラムにより監視することもできます。キースペースまたはテーブルは、使用可能な状態になると、システムスキーマに表示されます。

> **Note**
>
> を使用してキースペースの作成を最適化するには AWS CloudFormation、このユーティリティを使用して CQL スクリプトを CloudFormation テンプレートに変換します。このツールは GitHub リポジトリから入手できます。

トピック

- 新しいキースペースを作成したが、表示またはアクセスできない

- 新しいテーブルを作成したが、表示またはアクセスできない
- Amazon Keyspaces ポイントインタイムリカバリ (PITR、point-in-time recovery) を使用してテーブルを復元することができない
- INSERT/UPDATE を使用してカスタム有効期限 (TTL) 設定を編集しようとすると、オペレーションが失敗する
- Amazon Keyspaces テーブルにデータをアップロードしようとすると、列数の超過に関するエラーが発生する
- Amazon Keyspaces テーブルのデータを削除しようとすると、その範囲のために削除が失敗する

新しいキースペースを作成したが、表示またはアクセスできない

新しいキースペースへのアクセスを試行しているアプリケーションからエラーが送られてきた。

非同期的作成中である新規の Amazon Keyspaces キースペースにアクセスしようとすると、エラーが発生します。以下にエラーの例を示します。

```
InvalidRequest: Error from server: code=2200 [Invalid query] message="unconfigured
 keyspace mykeyspace"
```

新しいキースペースが使用可能な状態になるタイミングをチェックするための推奨設計パターンとは、Amazon Keyspaces のシステムスキーマテーブル (system_schema_mcs.*) のポーリングです。

詳細については、「the section called “キースペースの作成ステータスの確認”」を参照してください。

新しいテーブルを作成したが、表示またはアクセスできない

新しいテーブルへのアクセスを試行しているアプリケーションからエラーが送られてきました。

非同期的作成中である新規の Amazon Keyspaces テーブルにアクセスしようとすると、エラーが発生します。例えば、まだ使用できる状態ではないテーブルをクエリしようとすると失敗して `unconfigured table` というエラーが表示されます。

```
InvalidRequest: Error from server: code=2200 [Invalid query] message="unconfigured
 table mykeyspace.mytable"
```

`sync_table()` でテーブルを表示しようとすると失敗して `KeyError` が表示されます。

```
KeyError: 'mytable'
```

新しいテーブルが使用可能な状態になるタイミングをチェックするための推奨設計パターンとは、Amazon Keyspaces のシステムスキーマテーブル (system_schema_mcs.*) のポーリングです。

これは作成中のテーブルの出力例です。

```
user-at-123@cqlsh:system_schema_mcs> select table_name,status from
 system_schema_mcs.tables where keyspace_name='example_keyspace' and
 table_name='example_table';

table_name | status

------------+----------

example_table | CREATING

(1 rows)
```

これはアクティブなテーブルの出力例です。

```
user-at-123@cqlsh:system_schema_mcs> select table_name,status from
 system_schema_mcs.tables where keyspace_name='example_keyspace' and
 table_name='example_table';

table_name | status

------------+----------

example_table | ACTIVE

(1 rows)
```

詳細については、「the section called “テーブルの作成ステータスの確認”」を参照してください。

Amazon Keyspaces ポイントインタイムリカバリ (PITR、point-in-time recovery) を使用してテーブルを復元することができない

ポイントインタイムリカバリ (PITR) を使用して Amazon Keyspaces テーブルの復元を試みた際に、復元プロセスが開始しても正常に完了しない場合は、この特定のテーブルの復元プロセスに必要なアクセス許可の一部が設定されていない可能性があります。

Amazon Keyspaces では、ユーザーアクセス許可に加えて、復元プロセス中にプリンシパルに代わってアクションを実行するためのアクセス許可が必要になる場合があります。これは、テーブルがカスタマーマネージドキーで暗号化されている場合や、着信トラフィックを制限する IAM ポリシーを使用している場合です。

例えば、IAM ポリシーで条件キーを使用してソーストラフィックを特定のエンドポイントまたは IP 範囲に制限している場合、復元オペレーションは失敗します。プリンシパルの代わりに Amazon Keyspaces によってテーブルの復元オペレーションが実行されるようにするには、IAM ポリシーに `aws:ViaAWSService` グローバル条件キーを追加する必要があります。

テーブルを復元するためのアクセス許可の詳細については、「the section called “復元用の IAM アクセス許可の設定”」を参照してください。

INSERT/UPDATE を使用してカスタム有効期限 (TTL) 設定を編集しようとすると、オペレーションが失敗する

カスタム TTL 値を挿入または更新しようとすると、次のエラーが発生してオペレーションが失敗することがあります。

```
TTL is not yet supported.
```

`INSERT` または `UPDATE` のオペレーションを使用して行または列のカスタム TTL 値を指定するには、まずそのテーブルの TTL を有効にする必要があります。`ttl` カスタムプロパティを使用すればテーブルの TTL を有効にすることができます。

テーブルのカスタム TTL 設定を有効にする方法の詳細については、「the section called “テーブルのカスタム TTL の更新”」を参照してください。

Amazon Keyspaces テーブルにデータをアップロードしようとすると、列数の超過に関するエラーが発生する

データをアップロードしていて、同時に更新できる列数を超過した。

このエラーは、テーブルスキーマが最大サイズの 350 KB を超えた場合に発生します。詳細については、「クォータ」を参照してください。

Amazon Keyspaces テーブルのデータを削除しようとすると、その範囲のために削除が失敗する

パーティションキーでデータを削除しようとすると、削除範囲エラーが表示されます。

このエラーは、1 回の削除オペレーションで 1,000 行以上を削除しようとしたときに発生します。

```
Range delete requests are limited by the amount of items that can be deleted in a
 single range.
```

詳細については、「the section called “範囲削除”」を参照してください。

1 つのパーティション内で 1,000 行以上を削除する場合は、次のオプションを検討してください。

- パーティションごとの削除 — パーティションの大部分が 1,000 行未満の場合は、パーティションごとにデータを削除できます。パーティションに含まれている行が 1,000 行以上である場合は、代わりにクラスタリング列ごとに削除してください。
- クラスタリング列ごとの削除 — モデルに複数のクラスタリング列が含まれている場合、列階層を使用して複数の行を削除できます。クラスタリング列はネスト構造であり、最上位の列に対してオペレーションを行うことで、多くの行を削除できます。
- 行ごとの削除 — 行を繰り返し処理し、そのフルプライマリキーで各行を削除できます (パーティション列とクラスタリング列)。
- ベストプラクティスとして、パーティション間での行の分割を検討してください。Amazon Keyspaces では、テーブルパーティション間でスループットを分散させることをお勧めします。これにより、物理リソース間でデータとアクセスが均等に分散され、最高のスループットが得られます。詳細については、「the section called “データモデリング”」を参照してください。

高いワークロードに対して削除オペレーションを計画する場合は、次の推奨事項も考慮してください。

- Amazon Keyspaces では、パーティションに含めることができる行数は事実上無制限です。これにより、従来の Cassandra ガイダンスである 100 MB よりも「広く」パーティションのスケールを調整できます。時系列や台帳については、時間の経過とともにデータをギガバイト単位で増加させることは珍しくありません。
- Amazon Keyspaces では、高いワークロードの削除オペレーションを実行する必要がある場合に考慮すべき圧縮戦略やトゥームストーンはありません。読み取りパフォーマンスに影響を与えることなく、必要な数のデータを削除できます。

Amazon Keyspaces (Apache Cassandra 向け) のモニタリング

モニタリングは、Amazon Keyspaces およびその他の AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。 には、Amazon Keyspaces をモニタリングし、問題が発生したときに報告し、必要に応じて自動アクションを実行するための以下のモニタリングツール AWS が用意されています。

- Amazon Keyspaces には、アカウント内のすべてのテーブルに集約されたレイテンシーとエラー AWS Management Console を示す事前設定されたダッシュボードが に用意されています。
- Amazon CloudWatch は、 AWS リソースと で実行されるアプリケーションを AWS リアルタイムでモニタリングします。カスタマイズしたダッシュボードでメトリクスの収集と追跡ができます。たとえば、さまざまなタイミングとロード条件でパフォーマンスを測定すれば、お客様の環境における Amazon Keyspaces の通常パフォーマンスのベースラインを作成できます。Amazon Keyspaces のモニタリングでは、過去のモニタリングデータを保存し、現在のパフォーマンスデータと比較することで、パフォーマンスの通常パターンと異常パターンを特定し、問題に対処する方法を考案できます。ベースラインを確立するには、少なくとも、システムエラーをモニタリングする必要があります。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。
- Amazon CloudWatch アラームは、指定した期間。1 つのメトリクスを監視し、特定のしきい値に関する複数の期間にわたるメトリクスの値に基づいて、1 つ以上のアクションを実行します。たとえば、Amazon Keyspaces をプロビジョニングモードで使用し、アプリケーションの自動スケーリングを行う場合、Application Auto Scaling のポリシーを評価するために、Amazon Simple Notification Service (Amazon SNS) が送信する通知がアクションはになります。

  CloudWatch アラームは、特定の状態にあるという理由だけではアクションを呼び出しません。状態が変わって、変わった状態が指定期間にわたって維持される必要があります。詳細については、「Amazon CloudWatch による Amazon Keyspaces のモニタリング」を参照してください。
- Amazon CloudWatch Logs では、Amazon Keyspaces テーブル、CloudTrail、およびその他のソースからのログファイルに対するモニタリング、保存、アクセスの操作ができます。CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることも可能です。詳細については、『Amazon CloudWatch Logs ユーザーガイド』を参照してください。
- AWS CloudTrailは、 AWS アカウント により、またはそのアカウントに代わって行われた API コールや関連イベントを取得し、指定した Amazon S3 バケットにログファイルを配信します。

AWSを呼び出したユーザーとアカウント、呼び出し元の IP アドレス、および呼び出しの発生日時を特定できます。詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

Amazon EventBridge は、アプリケーションをさまざまなイベントソースのデータに簡単に接続できるようにするサーバーレスイベントバスサービスです。EventBridge は、独自のアプリケーション、Software-as-a-Service (SaaS) アプリケーション、および AWS のサービスからリアルタイムデータのストリームを配信し、そのデータを Lambda などのターゲットにルーティングします。これにより、サービスで発生したイベントをモニタリングし、イベント駆動型アーキテクチャを構築できます。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。

トピック

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

Amazon CloudWatch による Amazon Keyspaces のモニタリング

Amazon CloudWatch を使用して Amazon Keyspaces をモニタリングすることで、raw データを収集し、ほぼリアルタイムで読み取り可能なメトリクスに加工することができます。これらの統計は 15 か月間保持されるため、履歴情報にアクセスし、ウェブアプリケーションまたはサービスの動作をより的確に把握できます。

また、特定のしきい値を監視するアラームを設定し、これらのしきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

> **Note**
>
> Amazon Keyspaces の一般的なメトリクスを示す事前設定された CloudWatch ダッシュボードをすばやく開始するには、 から利用可能な AWS CloudFormation テンプレートを使用できますhttps://github.com/aws-samples/amazon-keyspaces-cloudwatch-cloudformation-templates。

トピック

- Amazon Keyspaces のメトリクスの使用方法を教えてください。
- Amazon Keyspaces のメトリクスとディメンション

- CloudWatch での Amazon Keyspaces メトリクスの表示
- Amazon Keyspaces をモニタリングするための CloudWatch アラームの作成

Amazon Keyspaces のメトリクスの使用方法を教えてください。

Amazon Keyspaces からレポートされるメトリクスには、さまざまな方法で分析できる情報が含まれています。以下のリストは、メトリクスの一般的な利用方法をいくつか示しています。ここで紹介するのは開始するための提案事項です。すべてを網羅しているわけではありません。メトリクスと保持の詳細については、「Metrics」(メトリクス) を参照してください。

目的	関連するメトリクス
どうすればシステムエラーの発生の有無を判断できますか。	`SystemErrors` をモニタリングすれば、サーバーエラーコードを発生させたリクエストがあるかどうかを判断できます。通常、このメトリクスはゼロであるべきです。そうでない場合は、調査することをお勧めします。
読み取りの平均プロビジョンドキャパシティと消費キャパシティを比較する方法を教えてください。	読み取りの平均プロビジョンドキャパシティと消費キャパシティをモニタリングするには 1. `ConsumedReadCapacityUnits` と `ProvisionedReadCapacityUnits` の [Period (期間)] をモニタリング間隔に設定します。 2. `ConsumedReadCapacityUnits` の [Statistic (統計)] を `Average` から `Sum` に変更します。 3. 新しい空の [Math expression (数式)] を作成します。 4. 新しい数式の [Details (詳細)] セクションで、`ConsumedReadCapacityUnits` の Id を入力し、そのメトリクスを、そのメトリクスの CloudWatch PERIOD 関数 (metric_id/(PERIOD(metric_id)) で割ります。 5. `ConsumedReadCapacityUnits` の選択を解除します。 これで、読み取りの平均消費キャパシティをプロビジョンドキャパシティと比較できます。基本的な算術関数の詳細と時系

目的	関連するメトリクス
	列の作成方法の詳細については、「Using metric math (メトリクス Math の使用)」 を参照してください。
書き込みの平均プロビジョンドキャパシティと消費キャパシティを比較する方法を教えてください。	書き込みのプロビジョンドキャパシティと消費キャパシティをモニタリングするには 1. `ConsumedWriteCapacityUnits` と `ProvisionedWriteCapacityUnits` の [Period (期間)] をモニタリング間隔に設定します。 2. `ConsumedWriteCapacityUnits` の [Statistic (統計)] を `Average` から `Sum` に変更します。 3. 新しい空の [Math expression (数式)] を作成します。 4. 新しい数式の [Details (詳細)] セクションで、`ConsumedWriteCapacityUnits` の Id を入力し、そのメトリクスを、そのメトリクスの CloudWatch PERIOD 関数 (metric_id/(PERIOD(metric_id)) で割ります。 5. `ConsumedWriteCapacityUnits` の選択を解除します。 これで、書き込みの平均消費キャパシティをプロビジョンドキャパシティと比較できます。基本的な算術関数の詳細と時系列の作成方法の詳細については、「Using metric math(メトリクス Math の使用)」を参照してください。

Amazon Keyspaces のメトリクスとディメンション

Amazon Keyspaces を操作すると、メトリクスとディメンションが Amazon CloudWatch に送信されます。メトリクス値はすべて、1 分ごとに集計されて報告されます。以下のセクションでは、Amazon Keyspaces で使用可能なすべてのメトリクスの概要を示します。

トピック

- Amazon Keyspaces のメトリクス
- Amazon Keyspaces メトリクスのディメンション

- Amazon Keyspaces 変更データキャプチャ (CDC) のメトリクス
- Amazon Keyspaces 変更データキャプチャ (CDC) メトリクスのディメンション

Amazon Keyspaces のメトリクス

Amazon CloudWatch では 1 分間隔で Amazon Keyspaces メトリクスが集計されます。

Average や Sum など、すべての統計が必ずしも常にすべてのメトリクスに適用可能であるとは限りません。ただし、これらの値はすべて、Amazon Keyspaces コンソール、CloudWatch コンソール AWS CLI、またはすべてのメトリクス AWS SDKs を使用して使用できます。次の表は、各メトリクスに適用可能な有効な統計のリストを示します。

メトリクス	説明
AccountMaxTableLevelReads	アカウントのテーブルで使用できる読み取りキャパシティユニットの最大数。オンデマンドテーブルの場合、この制限は、テーブルで使用できる最大の読み取りリクエストユニットです。 単位: Count 有効な統計: • Maximum — アカウントのテーブルで使用できる読み取りキャパシティユニットの最大数。
AccountMaxTableLevelWrites	アカウントのテーブルで使用できる書き込みキャパシティユニットの最大数。オンデマンドテーブルの場合、この制限は、テーブルで使用できる最大の書き込みリクエストユニットです。 単位: Count 有効な統計: • Maximum — アカウントのテーブルで使用できる書き込みキャパシティユニットの最大数。

メトリクス	説明
AccountProvisioned ReadCapacityUtilization	アカウントで利用されるプロビジョニング済み読み込み容量ユニットの割合。 単位: Percent 有効な統計: • Maximum — アカウントで使用されるプロビジョニング済み読み込み容量ユニットの最大割合。 • Minimum — アカウントで使用されるプロビジョニング済み読み込み容量ユニットの最小割合。 • Average — アカウントで使用されるプロビジョニング済み読み込み容量ユニットの平均割合。メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み読み込み容量ユニットをすばやく調整すると、この統計には実際の平均が反映されないことがあります。
AccountProvisioned WriteCapacityUtilization	アカウントで利用されるプロビジョニング済み書き込み容量ユニットの割合。 単位: Percent 有効な統計: • Maximum - アカウントで利用されるプロビジョニング済み書き込み容量ユニットの最大割合。 • Minimum — アカウントで使用されるプロビジョニング済み読み込み容量ユニットの最小割合。 • Average — アカウントで使用されるプロビジョニング済み書き込み容量ユニットの平均割合。メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み書き込み容量ユニットをすばやく調整すると、この統計には実際の平均値が反映されないことがあります。

メトリクス	説明
BillableTableSizeInBytes	テーブルの支払請求可能なサイズ (バイト単位)。これは、テーブル内のすべての行のエンコードされたサイズの合計です。このメトリクスは、テーブルストレージのコストを長期にわたって追跡するのに役立ちます。 単位: Bytes ディメンション: Keyspace, TableName 有効な統計: • Maximum — テーブルの最大ストレージサイズ。 • Minimum — テーブルの最小ストレージサイズ。 • Average — テーブルの平均ストレージサイズ。このメトリクスは 4 ~ 6 時間間隔で計算されます。

メトリクス	説明
ConditionalCheckFailedRequests	失敗したライトウェイトトランザクション (LWT) 書き込みリクエストの数。INSERT、UPDATE、および DELETE オペレーションを使用すると、オペレーションを続行する前に true と評価される必要がある論理条件を指定できます。この条件が false として評価された場合、ConditionalCheckFailedRequests は 1 ずつ増加されます。条件チェックで評価結果が false となった場合も、行のサイズに基づいて書き込みキャパシティユニットが消費されます。詳細については、「the section called “LWT のキャパシティ消費量の推定”」を参照してください。 単位: Count ディメンション: Keyspace, TableName 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
ConsumedReadCapacityUnits	指定した期間に消費された読み取りキャパシティユニットの数。詳細については、「Read/Write capacity mode」(読み取り/書き込みキャパシティモード) を参照してください。 **Note** 1 秒あたりの平均スループット使用率を把握するには、Sum 統計を使用して 1 分間の消費スループットを計算します。次に、その計算値を 1 分間の秒数 (60) で除算して、1 秒あたりの平均 ConsumedReadCapacityUnits を計算します (この平均では、その間に発生した読み取りアクティビティの大きく短いスパイクは強調されません)。読み取りの平均消費キャパシティとプロビジョンドキャパシティの比較については、「the section called “メトリクスの使用”」を参照してください。 単位: Count ディメンション: Keyspace, TableName 有効な統計: • Minimum — テーブルへの個々のリクエストによって消費される読み取りキャパシティユニットの最小数。 • Maximum — テーブルへの個々のリクエストによって消費される読み取りキャパシティユニットの最大数。 • Average — 消費されたリクエストごとの平均読み込み容量。

メトリクス	説明
	> **Note** `Average` 値は、サンプル値がゼロになる非活動期間によって影響を受けます。 • Sum — 消費された読み込み容量ユニットの合計。これは、`ConsumedReadCapacityUnits` メトリクスの最も有用な統計です。 • `SampleCount` — Amazon Keyspaces へのリクエストの数 (読み取りキャパシティが消費されなかった場合も含む)。 > **Note** `SampleCount` 値は、サンプル値がゼロになる非活動期間によって影響を受けます。

メトリクス	説明
ConsumedWriteCapacityUnits	指定した期間に消費された書き込みキャパシティユニットの数。テーブルの書き込み消費キャパシティの合計を取得できます。詳細については、「Read/Write capacity mode」(読み取り/書き込みキャパシティモード) を参照してください。 **Note** 1 秒あたりの平均スループット使用率を把握するには、Sum 統計を使用して 1 分間の消費スループットを計算します。次に、その計算値を 1 分間の秒数 (60) で除算して、1 秒あたりの平均 ConsumedWriteCapacityUnits を計算します (この平均では、その間に発生した書き込みアクティビティの大きく短いスパイクは強調されません)。書き込みの平均消費キャパシティとプロビジョンドキャパシティの比較については、「the section called “メトリクスの使用”」を参照してください。 単位: Count ディメンション: Keyspace, TableName 有効な統計: • Minimum — テーブルへの個々のリクエストによって消費される書き込みキャパシティユニットの最小数。 • Maximum — テーブルへの個々のリクエストによって消費される書き込みキャパシティユニットの最大数。 • Average — 消費されたリクエストごとの平均書き込み容量。

メトリクス	説明
	> **Note** `Average` 値は、サンプル値がゼロになる非活動期間によって影響を受けます。 • Sum — 消費された書き込み容量ユニットの合計。これは、`ConsumedWriteCapacityUnits` メトリクスの最も有用な統計です。 • `SampleCount` — Amazon Keyspaces へのリクエストの数 (書き込みキャパシティが消費されなかった場合も含む)。 > **Note** `SampleCount` 値は、サンプル値がゼロになる非活動期間によって影響を受けます。

メトリクス	説明
MaxProvisionedTableReadCapacityUtilization	アカウントのプロビジョニングされた最高の読み取りテーブルにより使用されている読み取りプロビジョンドキャパシティの割合。 単位: Percent 有効な統計: • Maximum - : プロビジョニング済み読み込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み読み込みテーブルが使用している最大割合。 • Minimum - プロビジョニング済み読み込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み読み込みテーブルが使用している最小割合。 • Average - プロビジョニング済み読み込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み読み込みテーブルが使用している平均割合。メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み読み込み容量ユニットをすばやく調整すると、この統計には実際の平均が反映されないことがあります。

メトリクス	説明
MaxProvisionedTableWriteCapacityUtilization	書き込みのプロビジョンドキャパシティのうち、アカウントのプロビジョニングされた最高の書き込みテーブルにより使用されている割合。 単位: Percent 有効な統計: • Maximum – 書き込みのプロビジョンドキャパシティのうち、アカウントのプロビジョニングされた最高の書き込みテーブルにより使用されている割合の最大値。 • Minimum – 書き込みのプロビジョンドキャパシティのうち、アカウントのプロビジョニングされた最高の書き込みテーブルにより使用されている割合の最小値。 • Average – 書き込みのプロビジョンドキャパシティのうち、アカウントのプロビジョニングされた最高の書き込みテーブルにより使用されている割合の平均値。メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み書き込み容量ユニットをすばやく調整すると、この統計には実際の平均値が反映されないことがあります。

メトリクス	説明
PerConnectionRequestRateExceeded	接続ごとのリクエストレートのクォータを超える Amazon Keyspaces へのリクエスト。Amazon Keyspaces への各クライアント接続は、1 秒あたり最大 3000 の CQL リクエストに対応できます。クライアントで、複数の接続を作成してスループットを向上させることができます。 マルチリージョンレプリケーションを使用している場合、レプリケートされた各書き込みもこのクォータに影響します。ベストプラクティスとして、PerConnectionRequestRateExceeded エラーを回避するためにテーブルへの接続数を増やすことをお勧めします。Amazon Keyspaces で使用できる接続数に制限はありません。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • SampleCount • Sum

メトリクス	説明
ProvisionedReadCapacityUnits	テーブルの読み込みプロビジョンドキャパシティユニットの数。 TableName ディメンションからテーブルの ProvisionedReadCapacityUnits が返されます。 単位: Count ディメンション: Keyspace, TableName 有効な統計: • Minimum — プロビジョニング済み読み込み容量の最小設定。ALTER TABLE を使用して読み込み容量を増やす場合、このメトリクスは、この期間中のプロビジョニング済み ReadCapacityUnits の最小値を示します。 • Maximum — プロビジョニング済み読み込み容量の最大設定。ALTER TABLE を使用して読み込み容量を減らす場合、このメトリクスは、この期間中のプロビジョニング済み ReadCapacityUnits の最大値を示します。 • Average — プロビジョニング済み読み込み容量の平均。ProvisionedReadCapacityUnits メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み読み込み容量ユニットをすばやく調整すると、この統計には実際の平均が反映されないことがあります。

メトリクス	説明
ProvisionedWriteCapacityUnits	テーブルの書き込みプロビジョンドキャパシティユニットの数。 TableName ディメンションからテーブルの ProvisionedWriteCapacityUnits が返されます。 単位: Count ディメンション: Keyspace, TableName 有効な統計: • Minimum — プロビジョニング済み書き込み容量の最小設定。ALTER TABLE を使用して書き込み容量を増やす場合、このメトリクスは、この期間中のプロビジョニング済み WriteCapacityUnits の最小値を示します。 • Maximum — プロビジョニング済み書き込み容量の最大設定。ALTER TABLE を使用して書き込み容量を減らす場合、このメトリクスは、この期間中のプロビジョニング済み WriteCapacityUnits の最大値を示します。 • Average — プロビジョニング済み書き込み容量の平均。ProvisionedWriteCapacityUnits メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み書き込み容量ユニットをすばやく調整すると、この統計には実際の平均値が反映されないことがあります。

メトリクス	説明
ReadThrottleEvents	テーブルの読み取りプロビジョンドキャパシティを超える Amazon Keyspaces へのリクエスト、またはアカウントレベルのクォータ、接続ごとのリクエスト、またはパーティションレベルのクォータ。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • SampleCount • Sum
ReplicationLatency	このメトリクスはマルチリージョンキースペースにのみ適用され、マルチリージョンキースペース内のあるレプリカテーブルから別のレプリカテーブルに updates、inserts、または deletes をレプリケートするのにかかった時間を測定します。 単位: Millisecond ディメンション: TableName, ReceivingRegion 有効な統計: • Average • Maximum • Minimum

メトリクス	説明
ReturnedItemCountBySelect	指定期間中に複数行の SELECT クエリによって返された行数 複数行の SELECT クエリは、完全テーブルスキャンや範囲限定クエリなどの完全修飾プライマリキーが含まれていないクエリです。 返された行数は、評価された行数と必ずしも一致しません。例えば、100 行のテーブルの ALLOW FILTERING を含む SELECT * をリクエストし、15 行が返されるように結果を絞り込む WHERE 句を指定した場合を考えてみましょう。この場合、SELECT からのレスポンスには、100 の ScanCount と返された 15 行の Count が含まれます。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
StoragePartitionThroughputCapacityExceeded	パーティションのスループットキャパシティを超える Amazon Keyspaces ストレージパーティションへのリクエスト。Amazon Keyspaces ストレージパーティションは、最大 1000 WCU/WRU/秒、最大 3000 RCU/RRU/秒に対応できます。これらの例外を軽減するために、データモデルを確認して、読み取り/書き込みトラフィックをより多くのパーティションに分散させることをお勧めします。 **Note** 論理的な Amazon Keyspaces パーティションは、複数のストレージパーティションにまたがることができ、そのサイズは事実上無制限です。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • SampleCount • Sum
SuccessfulRequestCount	指定された期間に、正常に処理されたレコードの数。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • SampleCount

メトリクス	説明
SuccessfulRequestLatency	指定期間中の成功した Amazon Keyspaces へのリクエスト。SuccessfulRequestLatency は 2 種類の異なる情報を提供できます。 • リクエストが成功するまでの経過時間 (Minimum、Maximum、Sum、または Average)。 • 成功したリクエストの数 (SampleCount)。 SuccessfulRequestLatency は Amazon Keyspaces 内のアクティビティのみを反映し、ネットワークレイテンシーやクライアント側のアクティビティは考慮しません。 単位: Milliseconds ディメンション: Keyspace, TableName, Operation 有効な統計: • Minimum • Maximum • Average • SampleCount

メトリクス	説明
SystemErrors	指定期間中に ServerError を生成する Amazon Keyspaces へのリクエスト。ServerError は通常、内部サービスエラーを示します。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • Sum • SampleCount
SystemReconciliationDeletes	クライアント側のタイムスタンプが有効になっている場合に、トゥームストーン化されたデータを削除するために消費される単位。各 SystemReconciliationDelete には、1 行あたり最大 1 KB のデータを削除または更新するのに十分なキャパシティがあります。例えば、2.5 KB のデータが保存されている 1 行について、その行を更新する場合や、その行内の 1 つ以上の列を同時に削除する場合には、3 つの SystemReconciliationDeletes が必要です。また、3.5 KB のデータが含まれている 1 行全体を削除する場合は、4 つの SystemReconciliationDeletes が必要です。 単位: Count ディメンション: Keyspace, TableName 有効な統計: • Sum — ある期間に消費された SystemReconciliationDeletes の総数。

メトリクス	説明
TTLDeletes	有効期限 (TTL) を使用して行のデータを削除または更新するために消費されたユニット。各 TTLDelete には、1 行あたり最大 1 KB のデータを削除または更新するのに十分なキャパシティがあります。例えば、2.5 KB のデータが保存されている 1 行について、その行を更新する場合や、その行内の 1 つ以上の列を同時に削除する場合には、3 つの TTL 削除が必要です。また、3.5 KB のデータが含まれている 1 行全体を削除する場合は、4 つの TTL 削除が必要です。 単位: Count ディメンション: Keyspace, TableName 有効な統計: • Sum — ある期間に消費された TTLDeletes の総数。

メトリクス	説明
UserErrors	指定期間中に InvalidRequest エラーを生成する Amazon Keyspaces へのリクエスト。 InvalidRequest エラーは通常、パラメータの無効な組み合わせ、存在しないテーブルの更新の試行、不正なリクエスト署名など、クライアント側のエラーを表します。 UserErrors は、現在の AWS リージョン と現在の に対する無効なリクエストの集計を表します AWS アカウント。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • Sum • SampleCount
WriteThrottleEvents	テーブルの書き込みプロビジョンドキャパシティを超える Amazon Keyspaces へのリクエスト、接続ごとのリクエスト、またはパーティションレベルのクォータ。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • SampleCount • Sum

Amazon Keyspaces メトリクスのディメンション

Amazon Keyspaces のメトリクスは、アカウント、テーブル名、オペレーションなどの値別に分類されます。CloudWatch コンソールを使用して、以下の表に示すいずれかのディメンションに従って Amazon Keyspaces データを取得できます。

ディメンション	説明
Keyspace	このディメンションによりデータが特定のキースペースに制限されます。この値は、現在のリージョンおよび現在の AWS アカウントの任意のキースペースになる場合があります。
Operation	このディメンションによりデータが Amazon Keyspaces CQL オペレーションの 1 つ (INSERT オペレーション、SELECT オペレーションなど) に制限されます。
TableName	このディメンションは、特定のテーブルにデータを制限します。この値は、現在のリージョンおよび現在の AWS アカウントアの任意のテーブル名になる場合があります。そのテーブル名がアカウント内で一意でない場合は、Keyspace も指定する必要があります。

Amazon Keyspaces 変更データキャプチャ (CDC) のメトリクス

Amazon Keyspaces 変更データキャプチャ (CDC) エンドポイントへの API コールをモニタリングする CloudWatch アカウントでは、次のメトリクスを使用できます。

メトリクス	説明
ReturnedRecordsCount	指定された期間中にGetRecords オペレーションによって返されたストリームレコードの数。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計:

メトリクス	説明
	• Minimum • Maximum • Average • SampleCount • Sum
SuccessfulRequestLatency	指定期間中の成功した Amazon Keyspaces へのリクエスト。SuccessfulRequestLatency は 2 種類の異なる情報を提供できます。 • リクエストが成功するまでの経過時間 (Minimum、Maximum、Sum、または Average)。 • 成功したリクエストの数 (SampleCount)。 SuccessfulRequestLatency は Amazon Keyspaces CDC APIs、ネットワークレイテンシーやクライアント側のアクティビティを考慮しません。 単位: Milliseconds ディメンション: Keyspace, TableName, Operation 有効な統計: • Minimum • Maximum • Average • SampleCount

メトリクス	説明
SystemErrors	指定された期間中にHTTP 500ステータスコードを生成する Amazon Keyspaces CDC へのリクエスト。HTTP 500 ステータスコードは通常、内部サービスエラーを示します。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • Sum • SampleCount
UserErrors	指定された期間中にHTTP 400ステータスコードを生成する Amazon Keyspaces CDC へのリクエスト。 UserErrors は、現在 AWS リージョン および現在の Amazon Keyspaces CDC のHTTP 400エラーの集計を表します AWS アカウント。 単位: Count ディメンション: Keyspace, TableName, Operation 有効な統計: • Sum • SampleCount

Amazon Keyspaces 変更データキャプチャ (CDC) メトリクスのディメンション

Amazon Keyspaces のメトリクスは、アカウント、テーブル名、オペレーションなどの値別に分類されます。CloudWatch コンソールを使用して、以下の表に示すいずれかのディメンションに従って Amazon Keyspaces データを取得できます。

ディメンション	説明
StreamLabel	このディメンションTableName は、 KeyspaceName と が指定されている場合、データを特定の変更データキャプチャ (CDC) ストリームに制限します。
Keyspace	このディメンションによりデータが特定のキースペースに制限されます。この値は、現在のリージョンおよび現在の AWS アカウントの任意のキースペースになる場合があります。
TableName	このディメンションは、特定のテーブルにデータを制限します。この値は、現在のリージョンおよび現在の AWS アカウントアの任意のテーブル名になる場合があります。そのテーブル名がアカウント内で一意でない場合は、Keyspace も指定する必要があります。
Operation	このディメンションは、データを Amazon Keyspaces Streams API オペレーション、GetRecords 、GetShardIterator および GetStream のいずれかに制限しますListStreams 。

CloudWatch での Amazon Keyspaces メトリクスの表示

Amazon Keyspaces の操作時に、以下のメトリクスとディメンションが Amazon CloudWatch に送信されます。メトリクス値はすべて、1 分ごとに集計されて報告されます。以下の手順を使用して、Amazon Keyspaces のメトリクスを表示することができます。

CloudWatch コンソールを使用してメトリクスを表示するには

メトリクスはまずサービスの名前空間ごとにグループ化され、次に各名前空間内のさまざまなディメンションの組み合わせごとにグループ化されます。

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 必要に応じて AWS リージョンを変更します。ナビゲーションバーで、 AWS リソースがあるリージョンを選択します。詳細については、AWS サービスエンドポイントを参照してください。
3. ナビゲーションペインで [Metrics] (メトリクス) を選択します。
4. [All metrics] (すべてのメトリクス) タブで、AWS/Cassandra. を選択します。

CLI AWS を使用してメトリクスを表示するには

- コマンドプロンプトで、次のコマンドを使用します。

```
aws cloudwatch list-metrics --namespace "AWS/Cassandra"
```

Amazon Keyspaces をモニタリングするための CloudWatch アラームの作成

Amazon Keyspaces 向けの Amazon CloudWatch のアラームを作成して、アラームの状態が変わったときに Amazon Simple Notification Service (Amazon SNS) メッセージが送信されるようにすることができます。指定した期間中、1 つのアラームが 1 つのメトリクスを監視します。このアラームは、複数の期間にわたる一定のしきい値とメトリクスの値の関係性に基づき、1 つ以上のアクションを実行します。アクションは、Amazon SNS トピックまたは Application Auto Scaling ポリシーに送信される通知です。

Application Auto Scaling でプロビジョニングモードで Amazon Keyspaces を使用すると、サービスがユーザーに代わって CloudWatch アラームのペアを 2 つ作成します。各ペアは、プロビジョンドスループット設定と消費スループット設定の上下の境界を示します。CloudWatch アラームは、テーブルの実際の使用率が一定期間ターゲット使用率を逸脱したときにトリガーされます。アプリケーションの Auto Scaling によって作成される CloudWatch アラームの詳細については、「the section called “Amazon Keyspaces オートスケーリングの仕組み”」を参照してください。

アラームは持続している状態変化に対してのみアクションを呼び出します。CloudWatch アラームは、特定の状態にあるという理由だけではアクションを呼び出しません。状態が変わって、変わった状態が指定期間にわたって維持される必要があります。

CloudWatch アラームの作成の詳細については、『Amazon CloudWatch ユーザーガイド』の「Using Amazon CloudWatch alarms (Amazon CloudWatch アラームの使用)」を参照してください。

を使用した Amazon Keyspaces API コールのログ記録 AWS CloudTrail

Amazon Keyspaces は AWS CloudTrail、Amazon Keyspaces のユーザー、ロール、または のサービス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 では Amazon Keyspaces のデータ定義言語 (DDL) API コールとデータ操作言語 (DML) API コールがイベントとしてキャプチャされます。キャプチャされたコールには、Amazon Keyspaces コンソー

ルからのコールと、Amazon Keyspaces API オペレーションへのプログラムによるコールが含まれます。

証跡を作成すれば、Amazon Keyspaces のイベントなど、Amazon Simple Storage Service (Amazon S3) バケットへの CloudTrail イベントの継続配信を有効にすることができます。

証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、サポートされる最新のイベントを表示できます。CloudTrail で収集された情報を使用して、Amazon Keyspaces に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

CloudTrail の詳細については、『AWS CloudTrail ユーザーガイド』を参照してください。

トピック

- CloudTrail での Amazon Keyspaces ログファイルエントリの設定
- CloudTrail での Amazon Keyspaces データ定義言語 (DDL) 情報
- CloudTrail での Amazon Keyspaces データ操作言語 (DML) 情報
- Amazon Keyspaces ログファイルエントリの理解

CloudTrail での Amazon Keyspaces ログファイルエントリの設定

CloudTrail に記録される各 Amazon Keyspaces API アクションには、CQL クエリ言語で表現されるリクエストパラメータが含まれます。詳細については、「CQL 言語リファレンス」を参照してください。

最近のイベントは、 AWS アカウントで表示、検索、ダウンロードできます。詳細については、「CloudTrail Event 履歴でのイベントの表示」を参照してください。

Amazon Keyspaces 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、証跡はすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、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。

- 証跡作成の概要
- 「CloudTrail がサポートされているサービスと統合」
- 「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。アイデンティティ情報は、以下を判別するのに役立ちます。

- リクエストがルートまたは AWS Identity and Access Management (IAM) ユーザー認証情報を使用して行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- 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 エレメント」を参照してください。

CloudTrail での Amazon Keyspaces データ定義言語 (DDL) 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 Keyspaces で DDL 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 イベントとして自動的に記録されます。次の表は、Amazon Keyspaces について記録される DDL ステートメントを示しています。

CloudTrail **eventName**	Statement	CQL アクション	AWS SDK アクション
CreateKeyspace	DDL	`CREATE KEYSPACE`	`CreateKeyspace`
AlterKeyspace	DDL	`ALTER KEYSPACE`	`UpdateKeyspace`
DropKeyspace	DDL	`DROP KEYSPACE`	`DeleteKeyspace`
CreateTable	DDL	`CREATE TABLE`	`CreateTable`
DropTable	DDL	`DROP TABLE`	`DeleteTable`
AlterTable	DDL	`ALTER TABLE`	`UpdateTable` , `TagResource` , `UntagResource`

CloudTrail **eventName**	Statement	CQL アクション	AWS SDK アクション
CreateUdt	DDL	`CREATE TYPE`	`CreateType`
DropUdt	DDL	`DROP TYPE`	`DeleteType`
GetStream	DDL	使用可能な CQL アクションがない	`GetStream`
ListStreams	DDL	使用可能な CQL アクションがない	`ListStreams`

CloudTrail での Amazon Keyspaces データ操作言語 (DML) 情報

CloudTrail で Amazon Keyspaces DML ステートメントのログ記録を有効にするには、まず CloudTrail でデータプレーン API アクティビティのログ記録を有効にする必要があります。新規または既存の証跡で Amazon Keyspaces DML イベントのログ記録を開始するには、CloudTrail コンソールを使用してデータイベントタイプの Cassandra テーブルのアクティビティを口グに記録するか、 CLI または CloudTrail API AWS オペレーション`AWS::Cassandra::Table`を使用して`resources.type`値を に設定します。詳細については、「データイベントの口グ記録」を参照してください。

データイベントのアラームを作成する方法の詳細と例については、「Amazon Keyspaces の DML 監査を使用する (Apache Cassandra の場合)」という AWS データベースブログの次の投稿を参照してください。

次の表は、`Cassandra table` に対して CloudTrail によって記録されるデータイベントを示しています。

CloudTrail **eventName**	Statement	CQL アクション	AWS SDK アクション
Select	DML	`SELECT`	`GetKeyspace` , `GetTable`, `GetType`, `ListKeyspaces` , `ListTables` , `ListTypes`

CloudTrail **eventName**	Statement	CQL アクション	AWS SDK アクション
			, ListTagsF orResource
Insert	DML	INSERT	使用可能な AWS SDK アクションがない
更新	DML	UPDATE	使用可能な AWS SDK アクションがない
削除	DML	DELETE	使用可能な AWS SDK アクションがない
GetShardI terator	DML	使用可能な CQL アクションがない	GetShardI terator
GetRecords	DML	使用可能な CQL アクションがない	GetRecords

Amazon Keyspaces ログファイルエントリの理解

CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは任意ソースからの単一リクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどの情報を含みます。CloudTrail ログファイルは、パブリック API 呼び出しの順序付けられたスタックトレースではないため、特定の順序では表示されません。

以下は、CreateKeyspace、DropKeyspace、CreateTable、DropTable の各アクションを示した CloudTrail ログエントリの例です。

```
{
  "Records": [
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
        "arn": "arn:aws:sts::111122223333:assumed-role/users/alice",
        "accountId": "111122223333",
```

```
        "sessionContext": {
          "sessionIssuer": {
            "type": "Role",
            "principalId": "AKIAIOSFODNN7EXAMPLE",
            "arn": "arn:aws:iam::111122223333:role/Admin",
            "accountId": "111122223333",
            "userName": "Admin"
          },
          "webIdFederationData": {},
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2020-01-15T18:47:56Z"
          }
        }
      },
      "eventTime": "2020-01-15T18:53:04Z",
      "eventSource": "cassandra.amazonaws.com",
      "eventName": "CreateKeyspace",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "10.24.34.01",
      "userAgent": "Cassandra Client/ProtocolV4",
      "requestParameters": {
        "rawQuery": "\n\tCREATE KEYSPACE \"mykeyspace\"\n\tWITH\n\t\tREPLICATION =
{'class': 'SingleRegionStrategy'}\n\t\t",
        "keyspaceName": "mykeyspace"
      },
      "responseElements": null,
      "requestID": "bfa3e75d-bf4d-4fc0-be5e-89d15850eb41",
      "eventID": "d25beae8-f611-4229-877a-921557a07bb9",
      "readOnly": false,
      "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Keyspace",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/"
        }
      ],
      "eventType": "AwsApiCall",
      "apiVersion": "3.4.4",
      "recipientAccountId": "111122223333",
      "managementEvent": true,
      "eventCategory": "Management",
      "tlsDetails": {
          "tlsVersion": "TLSv1.2",
```

```
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    },
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
        "arn": "arn:aws:sts::111122223333:assumed-role/users/alice",
        "accountId": "111122223333",
        "sessionContext": {
          "sessionIssuer": {
            "type": "Role",
            "principalId": "AKIAIOSFODNN7EXAMPLE",
            "arn": "arn:aws:iam::111122223333:role/Admin",
            "accountId": "111122223333",
            "userName": "Admin"
          },
          "webIdFederationData": {},
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2020-01-15T18:47:56Z"
          }
        }
      },
      "eventTime": "2020-01-15T19:28:39Z",
      "eventSource": "cassandra.amazonaws.com",
      "eventName": "DropKeyspace",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "10.24.34.01",
      "userAgent": "Cassandra Client/ProtocolV4",
      "requestParameters": {
        "rawQuery": "DROP KEYSPACE \"mykeyspace\"",
        "keyspaceName": "mykeyspace"
      },
      "responseElements": null,
      "requestID": "66f3d86a-56ae-4c29-b46f-abcd489ed86b",
      "eventID": "e5aebeac-e1dd-41e3-a515-84fe6aaabd7b",
      "readOnly": false,
      "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Keyspace",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/"
```

```
        }
      ],
      "eventType": "AwsApiCall",
      "apiVersion": "3.4.4",
      "recipientAccountId": "111122223333",
      "managementEvent": true,
      "eventCategory": "Management",
      "tlsDetails": {
          "tlsVersion": "TLSv1.2",
          "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
          "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    },
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
        "arn": "arn:aws:sts::111122223333:assumed-role/users/alice",
        "accountId": "111122223333",
        "sessionContext": {
          "sessionIssuer": {
            "type": "Role",
            "principalId": "AKIAIOSFODNN7EXAMPLE",
            "arn": "arn:aws:iam::111122223333:role/Admin",
            "accountId": "111122223333",
            "userName": "Admin"
          },
          "webIdFederationData": {},
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2020-01-15T18:47:56Z"
          }
        }
      },
      "eventTime": "2020-01-15T18:55:24Z",
      "eventSource": "cassandra.amazonaws.com",
      "eventName": "CreateTable",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "10.24.34.01",
      "userAgent": "Cassandra Client/ProtocolV4",
      "requestParameters": {
        "rawQuery": "\n\tCREATE TABLE \"mykeyspace\".\"mytable\"(\n\t\t\"ID\" int,
\n\t\t\"username\" text,\n\t\t\"email\" text,\n\t\t\"post_type\" text,\n\t\tPRIMARY
 KEY((\"ID\", \"username\", \"email\")))",
```

```
        "keyspaceName": "mykeyspace",
        "tableName": "mytable"
      },
      "responseElements": null,
      "requestID": "5f845963-70ea-4988-8a7a-2e66d061aacb",
      "eventID": "fe0dbd2b-7b34-4675-a30c-740f9d8d73f9",
      "readOnly": false,
      "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Table",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable"
        }
      ],
      "eventType": "AwsApiCall",
      "apiVersion": "3.4.4",
      "recipientAccountId": "111122223333",
      "managementEvent": true,
      "eventCategory": "Management",
      "tlsDetails": {
          "tlsVersion": "TLSv1.2",
          "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
          "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    },
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
        "arn": "arn:aws:sts::111122223333:assumed-role/users/alice",
        "accountId": "111122223333",
        "sessionContext": {
          "sessionIssuer": {
            "type": "Role",
            "principalId": "AKIAIOSFODNN7EXAMPLE",
            "arn": "arn:aws:iam::111122223333:role/Admin",
            "accountId": "111122223333",
            "userName": "Admin"
          },
          "webIdFederationData": {},
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2020-01-15T18:47:56Z"
```

```
          }
        }
      },
      "eventTime": "2020-01-15T19:27:59Z",
      "eventSource": "cassandra.amazonaws.com",
      "eventName": "DropTable",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "10.24.34.01",
      "userAgent": "Cassandra Client/ProtocolV4",
      "requestParameters": {
        "rawQuery": "DROP TABLE \"mykeyspace\".\"mytable\"",
        "keyspaceName": "mykeyspace",
        "tableName": "mytable"
      },
      "responseElements": null,
      "requestID": "025501b0-3582-437e-9d18-8939e9ef262f",
      "eventID": "1a5cbedc-4e38-4889-8475-3eab98de0ffd",
      "readOnly": false,
      "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Table",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable"
        }
      ],
      "eventType": "AwsApiCall",
      "apiVersion": "3.4.4",
      "recipientAccountId": "111122223333",
      "managementEvent": true,
      "eventCategory": "Management",
      "tlsDetails": {
          "tlsVersion": "TLSv1.2",
          "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
          "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    }
  ]
}
```

次のログファイルは、SELECT ステートメントの例を示しています。

```
{
    "eventVersion": "1.09",
```

```
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/alice",
        "accountId": "111122223333",
        "userName": "alice"
    },
    "eventTime": "2023-11-17T10:38:04Z",
    "eventSource": "cassandra.amazonaws.com",
    "eventName": "Select",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "10.24.34.01",
    "userAgent": "Cassandra Client/ProtocolV4",
    "requestParameters": {
          "keyspaceName": "my_keyspace",
          "tableName": "my_table",
          "conditions": [
              "pk = **(Redacted)",
              "ck < 3**(Redacted)0",
              "region = 't**(Redacted)t'"
          ],
          "select": [
              "pk",
              "ck",
              "region"
          ],
          "allowFiltering": true
    },
    "responseElements": null,
    "requestID": "6d83bbf0-a3d0-4d49-b1d9-e31779a28628",
    "eventID": "e00552d3-34e9-4092-931a-912c4e08ba17",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::Cassandra::Table",
            "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/my_keyspace/
table/my_table"
        }
    ],
    "eventType": "AwsApiCall",
    "apiVersion": "3.4.4",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
```

```
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    }
}
```

次のログファイルは、INSERT ステートメントの例を示しています。

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/alice",
        "accountId": "111122223333",
        "userName": "alice"
    },
    "eventTime": "2023-12-01T22:11:43Z",
    "eventSource": "cassandra.amazonaws.com",
    "eventName": "Insert",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "10.24.34.01",
    "userAgent": "Cassandra Client/ProtocolV4",
    "requestParameters": {
          "keyspaceName": "my_keyspace",
          "tableName": "my_table",
          "primaryKeys": {
              "pk": "**(Redacted)",
              "ck": "1**(Redacted)8"
          },
          "columnNames": [
              "pk",
              "ck",
              "region"
          ],
          "updateParameters": {
              "TTL": "2**(Redacted)0"
          }
      }
    },
    "responseElements": null,
```

```
    "requestID": "edf8af47-2f87-4432-864d-a960ac35e471",
    "eventID": "81b56a1c-9bdd-4c92-bb8e-92776b5a3bf1",
    "readOnly": false,
    "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Table",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/my_keyspace/table/
my_table"
        }
    ],
    "eventType": "AwsApiCall",
    "apiVersion": "3.4.4",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    }
}
```

次のログファイルは、UPDATE ステートメントの例を示しています。

```
{
    "eventVersion": "1.09",
    "userIdentity": {
          "type": "IAMUser",
          "principalId": "AKIAIOSFODNN7EXAMPLE",
          "arn": "arn:aws:iam::111122223333:user/alice",
          "accountId": "111122223333",
          "userName": "alice"
    },
    "eventTime": "2023-12-01T22:11:43Z",
    "eventSource": "cassandra.amazonaws.com",
    "eventName": "Update",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "10.24.34.01",
    "userAgent": "Cassandra Client/ProtocolV4",
    "requestParameters": {
          "keyspaceName": "my_keyspace",
          "tableName": "my_table",
```

```
            "primaryKeys": {
                "pk": "'t**(Redacted)t'",
                "ck": "'s**(Redacted)g'"
            },
            "assignmentColumnNames": [
               "nonkey"
            ],
            "conditions": [
                "nonkey < 1**(Redacted)7"
            ]
        },
        "responseElements": null,
        "requestID": "edf8af47-2f87-4432-864d-a960ac35e471",
        "eventID": "81b56a1c-9bdd-4c92-bb8e-92776b5a3bf1",
        "readOnly": false,
        "resources": [
            {
              "accountId": "111122223333",
              "type": "AWS::Cassandra::Table",
              "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/my_keyspace/table/
my_table"
            }
        ],
        "eventType": "AwsApiCall",
        "apiVersion": "3.4.4",
        "managementEvent": false,
        "recipientAccountId": "111122223333",
        "eventCategory": "Data",
        "tlsDetails": {
              "tlsVersion": "TLSv1.3",
              "cipherSuite": "TLS_AES_128_GCM_SHA256",
              "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
        }
}
```

次のログファイルは、DELETE ステートメントの例を示しています。

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/alice",
```

```
        "accountId": "111122223333",
        "userName": "alice",
    },
    "eventTime": "2023-10-23T13:59:05Z",
    "eventSource": "cassandra.amazonaws.com",
    "eventName": "Delete",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "10.24.34.01",
    "userAgent": "Cassandra Client/ProtocolV4",
    "requestParameters": {
        "keyspaceName": "my_keyspace",
        "tableName": "my_table",
        "primaryKeys": {
            "pk": "**(Redacted)",
            "ck": "**(Redacted)"
        },
        "conditions": [],
        "deleteColumnNames": [
            "m",
            "s"
        ],
        "updateParameters": {}
    },
    "responseElements": null,
    "requestID": "3d45e63b-c0c8-48e2-bc64-31afc5b4f49d",
    "eventID": "499da055-c642-4762-8775-d91757f06512",
    "readOnly": false,
    "resources": [
      {
        "accountId": "111122223333",
        "type": "AWS::Cassandra::Table",
        "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/my_keyspace/table/
my_table"
      }
    ],
    "eventType": "AwsApiCall",
    "apiVersion": "3.4.4",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
```

```
    }
}
```

Amazon Keyspaces (Apache Cassandra 向け) のセキュリティ

で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 とお客様の間の責任共有です。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説明しています。

- クラウドのセキュリティ – AWS は、 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責任を担います AWS クラウド。 は、お客様が安全に使用できるサービス AWS も提供します。セキュリティの有効性は、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審査機関によって定期的にテストおよび検証されています。Amazon Keyspaces 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 Services in scope by compliance program」(適用範囲内のコンプライアンスプログラム別サービス)」を参照してください。
- クラウドのセキュリティ – 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。

このドキュメントでは、Amazon Keyspaces を使用する際に責任共有モデルを適用する方法が説明されています。以下のトピックでは、セキュリティおよびコンプライアンス上の目的を達成するように Amazon Keyspaces を設定する方法について説明します。また、Amazon Keyspaces 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。

トピック

- Amazon Keyspaces におけるデータ保護
- AWS Identity and Access Management Amazon Keyspaces 用
- Amazon Keyspaces (Apache Cassandra 向け) のコンプライアンス検証
- Amazon Keyspaces の耐障害性と災害対策
- Amazon Keyspaces のインフラストラクチャセキュリティ
- Amazon Keyspaces の設定と脆弱性の分析
- Amazon Keyspaces のセキュリティベストプラクティス

Amazon Keyspaces におけるデータ保護

責任 AWS 共有モデル、Amazon Keyspaces (Apache Cassandra 向け) 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、データプライバシーに関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された AWS 責任共有モデルおよび GDPR のブログ記事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 Identity Center 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 が必須で、TLS 1.3 をお勧めします。
- 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、その中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 Keyspaces AWS CLIまたは他の AWS のサービス を操作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

- Amazon Keyspaces の保管データ暗号化
- Amazon Keyspaces での転送時の暗号化
- Amazon Keyspaces におけるインターネットワークトラフィックプライバシー

Amazon Keyspaces の保管データ暗号化

Amazon Keyspaces (Apache Cassandra 向け) の保管データ暗号化では、AWS Key Management Service (AWS KMS) に保存されている暗号化キーを使用して保管中のすべてのデータを暗号化することで、セキュリティを強化します。この機能は、機密データの保護における負担と複雑な作業を減らすのに役立ちます。保管されているデータを暗号化することで、データ保護に関する厳格なコンプライアンスと規制要件を満たしたセキュリティ重視のアプリケーションを構築できます。

Amazon Keyspaces の保管データ暗号化では、256 ビット高度暗号化基準 (AES-256) を使用してデータを暗号化します。この機能は、ストレージへの不正アクセスからデータを保護するのに役立ちます。

Amazon Keyspaces は、テーブルとストリーム内のデータを透過的に暗号化および復号します。Amazon Keyspaces では、データ暗号化にエンベロープ暗号化とキー階層が使用されます。ルート暗号化キーの保存と管理のために AWS KMS に統合されます。暗号化キー階層の詳細については、「the section called “動作”」を参照してください。エンベロープ暗号化などの 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 KMS 「 管理サービスの概念」を参照してください。

新しいテーブルを作成するとき、以下のいずれかの AWS KMS キー (KMS キー) を選択できます。

- AWS 所有のキー – これはデフォルトの暗号化タイプです。キーは Amazon Keyspaces により所有されます (追加料金なし)。
- カスタマーマネージドキー – このキーはアカウントに保存され、ユーザーによって作成、所有、管理されます。カスタマーマネージドキーを完全に制御できます (AWS KMS 料金が適用されます)。

Amazon Keyspaces は、基になるテーブルと同じキーを使用して変更データキャプチャ (CDC) ストリームを自動的に暗号化します。CDC の詳細については、「」を参照してくださいthe section called “CDC ストリームの使用”。

AWS 所有のキー とカスタマーマネージドキーはいつでも切り替えることができます。新しいテーブルを作成するときや、コンソールの使用またはプログラムによる CQL ステートメントの使用により、既存のテーブルの KMS キーを変更するときに、カスタマーマネージドポリシーを指定することができます。この方法の詳細は、「保管データ暗号化: カスタマーマネージドキーを使用して Amazon Keyspaces のテーブルを暗号化する方法」を参照してください。

のデフォルトオプションを使用した保管時の暗号化 AWS 所有のキー は、追加料金なしで提供されます。ただし、カスタマーマネージドキーには AWS KMS の料金がかかります。料金の詳細については、「AWS KMS の料金」を参照してください。

保管時の Amazon Keyspaces 暗号化は AWS リージョン、 AWS 中国 (北京) および AWS 中国 (寧夏) リージョンを含むすべての で使用できます。詳細については、「保管データ暗号化: Amazon Keyspaces での動作」を参照してください。

トピック

- 保管データ暗号化: Amazon Keyspaces での動作
- 保管データ暗号化: カスタマーマネージドキーを使用して Amazon Keyspaces のテーブルを暗号化する方法

保管データ暗号化: Amazon Keyspaces での動作

Amazon Keyspaces (Apache Cassandra 向け) の保管データ暗号化では、256 ビット高度暗号化基準 (AES-256) を使用してデータを暗号化します。この機能は、ストレージへの不正アクセスからデータを保護するのに役立ちます。デフォルトでは、Amazon Keyspaces テーブルのすべての顧客データが保管時に暗号化され、サーバー側の暗号化が透過的であるため、アプリケーションに変更を加える必要はありません。

保管時の暗号化は、テーブルの暗号化に使用される暗号化キーを管理するために AWS Key Management Service （ AWS KMS) と統合されます。新規テーブルの作成時や、既存テーブルの更新時に、以下の AWS KMS キーオプションのいずれかを選択できます。

- AWS 所有のキー – これはデフォルトの暗号化タイプです。キーは Amazon Keyspaces により所有されます (追加料金なし)。
- カスタマーマネージドキー – このキーはアカウントに保存され、ユーザーによって作成、所有、管理されます。カスタマーマネージドキーを完全に制御できます (AWS KMS 料金が適用されます)。

AWS KMS キー (KMS キー)

保管時の暗号化は、すべての Amazon Keyspaces データを AWS KMS キーで保護します。Amazon Keyspaces ではデフォルトで AWS 所有のキー が使用されます。これは、Amazon Keyspaces サービスアカウントで作成して管理するマルチテナント暗号化キーです。

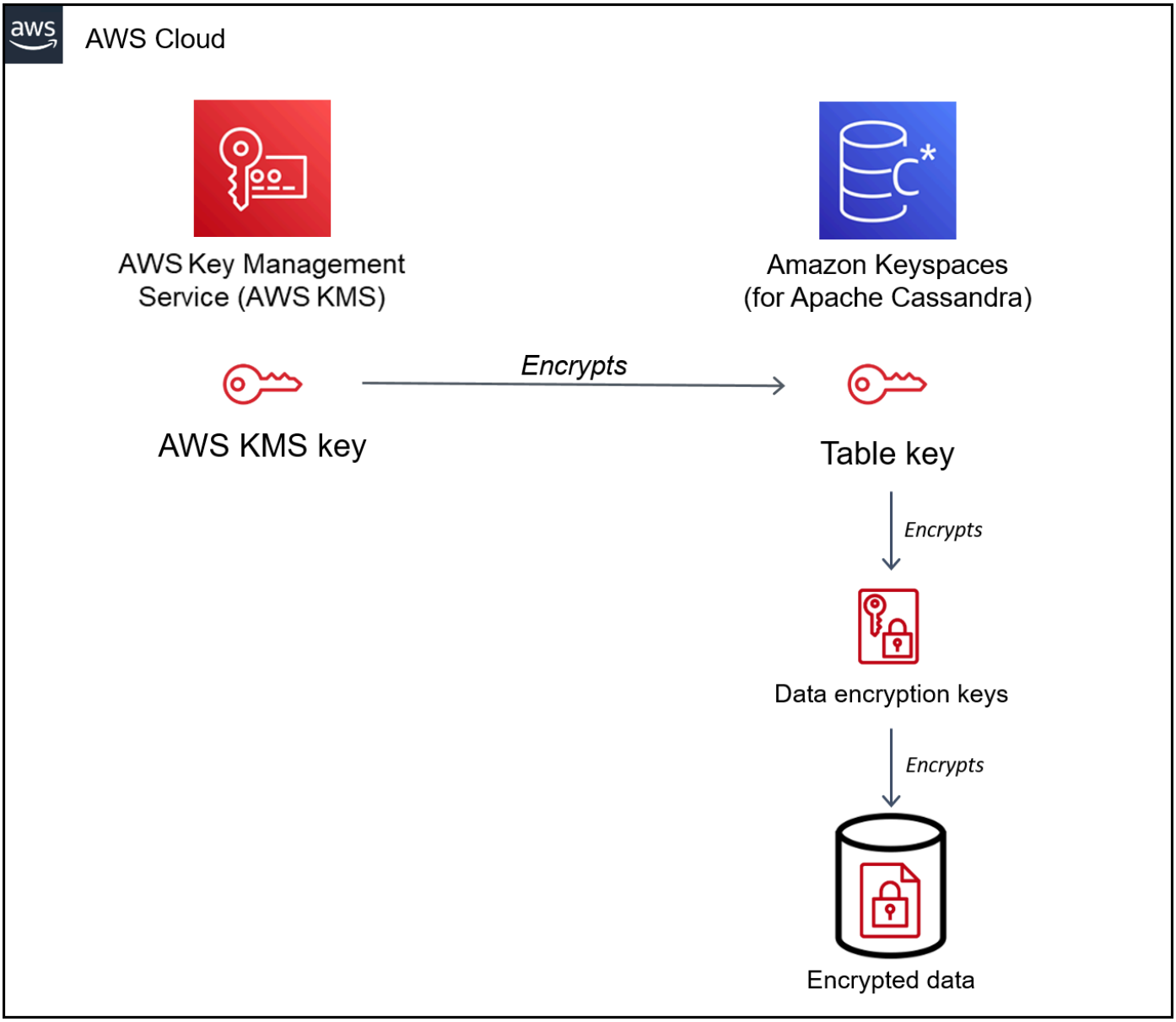
ただし、Amazon Keyspaces テーブルの暗号化には AWS アカウントのカスタマーマネージドキーを使用できます。キースペース内のテーブルごとに異なる KMS キーを選択することができます。テーブルに対して選択した KMS キーは、すべてのメタデータと復元可能なバックアップの暗号化にも使用されます。

テーブルを作成または更新するときは、テーブル用の KMS キーを選択します。テーブルの KMS キーは、Amazon Keyspaces コンソールまたは ALTER TABLE ステートメントを使用していつでも変更できます。KMS キーの切り替えプロセスはシームレスであり、ダウンタイムが必要になることもなく、サービスの低下を招くこともありません。

キー階層

Amazon Keyspaces では、キー階層を使用してデータを暗号化します。このキー階層において KMS キーはルートキーです。これは Amazon Keyspaces テーブル暗号化キーの暗号化と復号化に使用されます。テーブル暗号化キーは、読み取り操作と書き込み操作の実行時にデータの暗号化と復号化のために Amazon Keyspaces で内部的に使用される暗号化キーを暗号化するために使用されます。

暗号化キー階層を使用すると、データを再暗号化する必要もなく、アプリケーションや継続的なデータ操作に影響を与えることもなく、KMS キーに変更を加えることができます。



テーブルキー

Amazon Keyspaces のテーブルキーは、キー暗号化用のキーとして使用されます。Amazon Keyspaces では、テーブルキーを使用して、テーブルやログファイルや復元可能なバックアップに保存されたデータを暗号化するために使用される内部データ暗号化キーを保護します。Amazon Keyspaces により、テーブル内の基本構造ごとに一意のデータ暗号化キーが生成されます。ただし、複数のテーブル行が同じデータ暗号化キーで保護されていることがあります。

KMS キーをカスタマーマネージドキーに初めて設定すると、 はデータキー AWS KMS を生成します。 AWS KMS データキーは、Amazon Keyspaces のテーブルキーを参照します。

暗号化されたテーブルにアクセスすると、Amazon Keyspaces は AWS KMS KMS キーを使用してテーブルキーを復号するリクエストを に送信します。その後、プレーンテキストテーブルキーを使用して Amazon Keyspaces データ暗号化キーが復号化され、プレーンテキストデータ暗号化キーを使用してテーブルデータが復号化されます。

Amazon Keyspaces は、テーブルキーとデータ暗号化キーを の外部で使用および保存します AWS KMS。これによって、Advanced Encryption Standard (AES) 暗号化および 256 ビット暗号化キーのすべてのキーが保護されます。続いて、暗号化されたキーを暗号化されたデータと一緒に保存します。これらのキーおよびデータは、必要なときにテーブルデータの暗号化に使用できます。

テーブルキーのキャッシュ

Amazon Keyspaces オペレーション AWS KMS ごとに が呼び出されないように、Amazon Keyspaces は各接続のプレーンテキストテーブルキーをメモリにキャッシュします。Amazon Keyspaces は、5 分間非アクティブ状態が続いた後にキャッシュされたテーブルキーのリクエストを取得すると、テーブルキーを復号 AWS KMS するための新しいリクエストを に送信します。この呼び出しは、テーブルキーを復号する最後のリクエスト以降に AWS KMS または AWS Identity and Access Management (IAM) で KMS キーのアクセスポリシーに加えられた変更をキャプチャします。

エンベロープ暗号化

テーブルのカスタマーマネージドキーを変更すると、Amazon Keyspaces によって新しいテーブルキーが生成されます。次に、新しいテーブルキーを使用してデータ暗号化キーの再暗号化が行われます。また、新しいテーブルキーを使用して過去のテーブルキーが暗号化され、それを復元可能なバックアップの保護に使用します。このプロセスはエンベロープ暗号化と呼ばれます。これにより、カスタマーマネージドキーをローテーションしても、復元可能なバックアップにアクセスできるようになります。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「Envelope Encryption」(エンベロープ暗号化) を参照してください。

トピック

- AWS 所有キー
- カスタマーマネージドキー
- 保管データ暗号化の使用に関する注意事項

AWS 所有キー

AWS 所有のキー は に保存されません AWS アカウント。これらは、multiple. AWS services で使用するために が AWS 所有および管理する KMS キーのコレクションの一部です AWS アカウント。サービスは、 AWS 所有のキー を使用してデータを保護します。

表示、管理、使用 AWS 所有のキー、またはそれらの使用を監査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作業やプログラムを操作したり変更したりする必要はありません。

の使用に対して月額料金や使用料金は請求されず AWS 所有のキー、アカウントの AWS KMS クォータにもカウントされません。

カスタマーマネージドキー

カスタマーマネージドキーは、 AWS アカウント ユーザーが作成、所有、管理する のキーです。ユーザーは、この KMS キーに関する完全なコントロール権を持ちます。

カスタマーマネージドキーを使用して次の機能を取得します。

- カスタマーマネージドキーの作成と管理を行います。これには、キーポリシー、IAM ポリシー、カスタマーマネージドキーへのアクセスを制御する権限の設定および維持が含まれます。カスタマーマネージドキーの有効化と無効化、自動キーローテーションの有効化と無効化、使用しなくなったカスタマーマネージドキーの削除のスケジューリングを行うことができます。管理するカスタマーマネージドキーのタグとエイリアスを作成できます。
- インポートされたキーマテリアルを持つカスタマーマネージドキー、またはユーザーが所有して管理するカスタムキーストアで、カスタマーマネージドキーを使用できます。
- AWS CloudTrail と Amazon CloudWatch Logs を使用して、Amazon Keyspaces が AWS KMS ユーザーに代わって に送信するリクエストを追跡できます。詳細については、「the section called “ステップ 6: でモニタリングを設定する AWS CloudTrail”」を参照してください。

カスタマーマネージドキーには API コールごとに料金が発生し、これらの AWS KMS KMS キーにはクォータが適用されます。詳細については、「AWS KMS resource or request quotas」(KMS リソースクォータまたはリクエストクォータ) を参照してください。

テーブルのルート暗号化キーとしてカスタマーマネージドポリシーを指定すると、復元可能なバックアップは、バックアップの作成時にテーブルに対して指定されている暗号化キーと同じものを使用して暗号化されます。テーブルの KMS キーをローテーションすると、キーエンベロープにより、最新の KMS キーが復元可能なすべてのバックアップにアクセスできるようになります。

Amazon Keyspaces には、テーブルデータへのアクセスが可能になるカスタマーマネージドキーへのアクセス権が必要です。暗号化キーの状態が無効に設定されているか、削除がスケジュールされている場合、Amazon Keyspaces ではデータの暗号化と復号化を実行できません。そのため、テーブルで読み取り/書き込みオペレーションを実行できません。暗号化キーにアクセスできないことが検出されると、Amazon Keyspaces からアラートメール通知が送信されます。

その場合、7 日以内に暗号化キーへのアクセスを復元しないと、Amazon Keyspaces によりテーブルが自動的に削除されます。ただし、Amazon Keyspaces では、テーブルが削除される前にテーブルデータの復元可能なバックアップが作成されます。その復元可能なバックアップは Amazon Keyspaces で 35 日間維持されます。35 日を過ぎるとそのテーブルデータは復元できなくなります。復元可能なバックアップの料金はかかりませんが、標準的な復元料金はかかります。

この復元可能なバックアップを使用すれば、データを新しいテーブルに復元できます。復元を開始するには、そのテーブルに対して最後に使用したカスタマーマネージドキーを有効にし、Amazon Keyspaces からのアクセスを確立する必要があります。

> **Note**
>
> 作成プロセスが完了する前の時点で、アクセスできない、または削除がスケジュールされているカスタマーマネージドキーを使用して、暗号化されたテーブルを作成すると、エラーが発生します。テーブルの作成オペレーションが失敗し、電子メール通知が送信されます。

保管データ暗号化の使用に関する注意事項

Amazon Keyspaces で保管データの暗号化を使用する場合は、以下の点を考慮してください。

- サーバー側の保管データ暗号化は、すべての Amazon Keyspaces テーブルデータで有効になり、無効にできません。テーブル全体の保管データが暗号化されます。特定の列または行を選択して暗号化することはできません。
- Amazon Keyspaces のデフォルトでは、単一サービスのデフォルトキー (AWS 所有のキー) を使用して、すべてのテーブルで暗号化が行われます。このキーは、存在しなければ自動的に作成されます。サービスデフォルトキーは無効にできません。
- 保管時の暗号化では、永続的ストレージメディアの静的 (保管時) データのみが暗号化されます。転送中のデータあるいは使用中のデータのデータ安全性に対する懸念がある場合には、追加の対策を実行する必要があります。

- 転送中のデータ: Amazon Keyspaces 内のすべてのデータが転送中に暗号化されます。デフォルトでは、Amazon Keyspaces との通信が、Secure Sockets Layer (SSL)/Transport Layer Security (TLS) 暗号化を使用して保護されます。
- 使用中のデータ: データを保護したうえで Amazon Keyspaces に送信するには、クライアント側暗号化を使用します。
- カスタマーマネージドキー: テーブル内の保管中のデータは、常にカスタマーマネージドキーを使用して暗号化されます。ただし、複数の行のアトミック更新を実行するオペレーションは、処理 AWS 所有のキー 中に を使用してデータを一時的に暗号化します。これには、range delete オペレーションと、静的データと非静的データに同時にアクセスするオペレーションが含まれます。

- 1 つのカスタマーマネージドキーにより、最大 50,000 の権限が可能です。カスタマーマネージドキーに関連付けられている各 Amazon Keyspaces テーブルでは、2 つの権限が消費されます。テーブルが削除されると、1 つの権限が解放されます。2番目の権限はテーブルの自動スナップショットの作成に使用されます。これは、Amazon Keyspaces によるカスタマーマネージドキーへのアクセス権が意図せず失われた場合に、データが失われないように保護するためです。この権限は、テーブルが削除されてから 42 日後にリリースされます。

保管データ暗号化: カスタマーマネージドキーを使用して Amazon Keyspaces のテーブルを暗号化する方法

コンソールまたは CQL ステートメントを使用して、新しいテーブル AWS KMS key の を指定し、Amazon Keyspaces の既存のテーブルの暗号化キーを更新できます。次のトピックでは、新規および既存のテーブルにカスタマーマネージドキーを実装する方法について説明します。

トピック

- 前提条件: を使用してカスタマーマネージドキーを作成し AWS KMS 、Amazon Keyspaces にアクセス許可を付与する
- ステップ 3: 新規テーブルのカスタマーマネージドキーを指定する
- ステップ 4: 既存のテーブルの暗号化キーを更新する
- ステップ 5: ログで Amazon Keyspaces 暗号化コンテキストを使用する
- ステップ 6: でモニタリングを設定する AWS CloudTrail

前提条件: を使用してカスタマーマネージドキーを作成し AWS KMS 、Amazon Keyspaces にアクセス許可を付与する

カスタマーマネージドキーで Amazon Keyspaces テーブルを保護する前に、まず AWS Key Management Service （ AWS KMS) でキーを作成し、Amazon Keyspaces にそのキーの使用を許可する必要があります。

ステップ 1: AWS KMSを使用してカスタマーマネージドキーを作成する

Amazon Keyspaces テーブルの保護に使用するカスタマーマネージドキーを作成するには、「 コンソールまたは AWS API を使用して対称暗号化 KMS キーを作成する」の手順に従います。

ステップ 2: カスタマーマネージドキーの使用を認可する

カスタマーマネージドキーを選択して Amazon Keyspaces テーブルを保護できるようにするには、あらかじめ、そのカスタマーマネージドキーに関するポリシーにより、そのキーの使用許可を Amazon Keyspaces に付与する必要があります。カスタマーマネージドキーに関するポリシーと権限については、ユーザーが完全に制御します。これらのアクセス権限は、キーポリシー、IAM ポリシー、または 許可で指定できます。

Amazon Keyspaces では、デフォルトの AWS 所有のキー を使用して AWS アカウント内で Amazon Keyspaces テーブルを保護する場合、追加の認可は必要ありません。

以下のトピックでは、Amazon Keyspaces テーブルでカスタマーマネージドキーを使用できるようにする IAM ポリシーと権限を使用して、必要な許可を設定する方法を示します。

トピック

- カスタマーマネージドキーのキーポリシー
- キーポリシーの例
- Amazon Keyspaces の認可への権限の使用

カスタマーマネージドキーのキーポリシー

カスタマーマネージドキーを選択して Amazon Keyspaces テーブルを保護する場合、Amazon Keyspaces には、選択を行うプリンシパルの代わって KMS キーを使用する許可が付与されます。そのプリンシパル (ユーザーまたはロール) は、Amazon Keyspaces で必要となるカスタマーマネージドキーに対する許可を取得しておく必要があります。

Amazon Keyspaces には、少なくとも、カスタマーマネージドキーに対する次の許可が必要です。

- kms:Encrypt
- kms:Decrypt
- kms:ReEncrypt* (for kms:ReEncryptFrom および kms:ReEncryptTo 向け)
- kms:GenerateDataKey* (for kms:GenerateDataKey および kms:GenerateDataKeyWithoutPlaintext 向け)
- kms:DescribeKey
- kms:CreateGrant

キーポリシーの例

例えば、次のキーポリシーの例では、必要なアクセス許可のみを提供します。このポリシーには、以下の影響があります。

- Amazon Keyspaces に対して、暗号化オペレーションでのそのカスタマーマネージドキーの使用と、権限の作成を許可します。ただしこれは、Amazon Keyspaces の使用許可を持つアカウント内のプリンシパルに代わって Amazon Keyspaces により処理が行われている場合に限られます。ポリシーステートメントで指定されたプリンシパルが Amazon Keyspaces の使用許可を持っていない場合、Amazon Keyspaces サービスからのコールであっても、コールは失敗します。
- kms:ViaService 条件キーは、ポリシーステートメントにリストされているプリンシパルの代わりに Amazon Keyspaces からリクエストが送信された場合にのみアクセス許可を受け入れます。これらのプリンシパルは、これらのオペレーションを直接呼び出すことはできません。`kms:ViaService` の値である `cassandra.*.amazonaws.com` は、リージョンの位置にアスタリスク (*) が付いていることに注意してください。Amazon Keyspaces では、特定の から独立したアクセス許可が必要です AWS リージョン。
- カスタマーマネージドキー管理者 (`db-team` ロールを引き受けることができるユーザー) に、カスタマーマネージドキーへの読み取り専用アクセス権と、権限 (テーブルを保護するために Amazon Keyspaces で必要となる権限を含む) を取り消す許可を与えます。
- Amazon Keyspaces に、カスタマーマネージドキーへの読み取り専用アクセス権を付与します。この場合、Amazon Keyspaces によりこれらのオペレーションが直接呼び出されます。アカウントプリンシパルに代わって処理する必要はありません。

サンプルキーポリシーを使用する前に、サンプルプリンシパルを の実際のプリンシパルに置き換えます AWS アカウント。

```
{
```

```
  "Id": "key-policy-cassandra",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access through Amazon Keyspaces for all principals in the account
 that are authorized to use Amazon Keyspaces",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "cassandra.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the customer managed key and revoke
 grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon Keyspaces の認可への権限の使用

Amazon Keyspaces は、キーポリシーに加え、権限を使ってカスタマーマネージドキーに対しアクセス許可を設定できます。アカウントにあるカスタマーマネージドキーへの権限を表示するには、ListGrants オペレーションを使用します。Amazon Keyspaces では、AWS 所有のキー を使用してテーブルを保護する場合に、権限や追加のアクセス許可は必要ありません。

Amazon Keyspaces は、バックグラウンドシステムメンテナンスと継続的なデータ保護タスクを実行するときに、権限のアクセス許可を使用します。また、テーブルキーの生成にグラントを使用します。

各グラントは、テーブルに固有です。アカウントに、同じカスタマーマネージドキーを使って暗号化された複数のテーブルがある場合、テーブルごとに、各タイプの権限があります。許可は、テーブル名と AWS アカウント ID を含む Amazon Keyspaces 暗号化コンテキストによって制約されます。この権限には、権限が不要になった場合に権限の使用停止を行うための許可が含まれています。

権限を作成するには、暗号化されたテーブルを作成したユーザーに代わって `CreateGrant` を呼び出すアクセス許可が Amazon Keyspaces に必要です。

キーポリシーは、アカウントがカスタマーマネージドキーの権限を取り消すことも許可できます。ただし、アクティブな暗号化テーブルで権限トを取り消すと、Amazon Keyspaces はテーブルを保護して維持することはできません。

ステップ 3: 新規テーブルのカスタマーマネージドキーを指定する

以下の手順に従って、Amazon Keyspaces コンソールまたは CQL を使用して新しいテーブルでカスタマーマネージドキーを指定します。

カスタマーマネージドキーを使用して暗号化されたテーブルを作成する (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで [Tables] (テーブル) を選択して、[Create table (テーブルの作成)] を選択します。
3. [Table details (テーブルの詳細)] セクションの [Create table (テーブルの作成)] ページで、キースペースを選択し、新しいテーブルに名前を付けます。
4. [Schema (スキーマ)] セクションで、テーブルのスキーマを作成します。
5. [Table settings (テーブルの設定)] セクションで、[Customize settings (設定のカスタマイズ)] を選択します。

6. [Encryption settings (暗号化設定)] に進みます。

   このステップでは、テーブルの暗号化設定を選択します。

   「 を選択」の「保管時の暗号化」セクションで、「別の KMS キーを選択 (詳細) 」オプションを選択し、検索フィールドで「」を選択する AWS KMS key か、「Amazon リソースネーム (ARN) を入力します。 AWS KMS key

   > **Note**
   >
   > 選択したキーにアクセスできない場合、または必要なアクセス許可がない場合は、「 AWS Key Management Service デベロッパーガイド」の「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。

7. [Create (作成)] を選択して暗号化テーブルを作成します。

保管データ暗号化 (CQL) にカスタマーマネージドキーが使用される新しいテーブルを作成する

保管データ暗号化にカスタマーマネージドキーが使用される新しいテーブルを作成するには、次の例に示すように、CREATE TABLE ステートメントを使用します。キー ARN を、Amazon Keyspaces に付与されたアクセス許可がある有効キーの ARN に置き換えてください。

```
CREATE TABLE my_keyspace.my_table(id bigint, name text, place text STATIC, PRIMARY
 KEY(id, name)) WITH CUSTOM_PROPERTIES = {
        'encryption_specification':{
                'encryption_type': 'CUSTOMER_MANAGED_KMS_KEY',
                'kms_key_identifier':'arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111'
            }
    };
```

Invalid Request Exception を受け取った場合、カスタマーマネージドキーが有効であり Amazon Keyspaces に必要な許可が付与されていることを確認する必要があります。キーが正しく設定されていることを確認するには、 AWS Key Management Service デベロッパーガイドの「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。

ステップ 4: 既存のテーブルの暗号化キーを更新する

Amazon Keyspaces コンソールまたは CQL を使用して、 AWS 所有のキー とカスタマーマネージド KMS キーの間で既存のテーブルの暗号化キーをいつでも変更できます。

新しいカスタマーマネージドキーで既存のテーブルを更新する (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. ナビゲーションペインで、[Tables (テーブル)] を選択します。
3. 更新するテーブルを選択し、次に [Additional settings (追加設定)] タブを選択します。
4. [Encryption at rest (保管データ暗号化)] セクションで、[Manage Encryption (暗号化の管理)] を選択してテーブルの暗号化設定を編集します。

   で AWS KMS key、別の KMS キーを選択する (詳細) オプションを選択し、検索フィールドで を選択する AWS KMS key か、Amazon リソースネーム (ARN) を入力します。

   > **Note**
   >
   > 選択したキーが有効でない場合は、「 AWS Key Management Service デベロッパーガイド」の「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。

   または、カスタマーマネージドキーで暗号化された AWS 所有のキー テーブルの を選択することもできます。
5. [Save changes (変更を保存)] を選択して、テーブルに加えた変更内容を保存します。

既存のテーブルに使用される暗号化キーを更新する

既存のテーブルの暗号化キーを変更するには、ALTER TABLE ステートメントを使用して、保管データ暗号化に使用するカスタマーマネージドキーを指定します。キー ARN を、Amazon Keyspaces に付与されたアクセス許可がある有効キーの ARN に置き換えてください。

```
ALTER TABLE my_keyspace.my_table WITH CUSTOM_PROPERTIES = {
              'encryption_specification':{
                     'encryption_type': 'CUSTOMER_MANAGED_KMS_KEY',
                     'kms_key_identifier':'arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111'
                  }
         };
```

Invalid Request Exception を受け取った場合、カスタマーマネージドキーが有効であり Amazon Keyspaces に必要な許可が付与されていることを確認する必要があります。キーが正し

く設定されていることを確認するには、「 AWS Key Management Service デベロッパーガイド」の「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。

を使用して暗号化キーをデフォルトの保管時の暗号化オプションに戻すには AWS 所有のキー、次の例に示すように ALTER TABLEステートメントを使用できます。

```
ALTER TABLE my_keyspace.my_table WITH CUSTOM_PROPERTIES = {
                'encryption_specification':{
                      'encryption_type' : 'AWS_OWNED_KMS_KEY'
                    }
        };
```

ステップ 5: ログで Amazon Keyspaces 暗号化コンテキストを使用する

暗号化コンテキスト は、一連のキー値のペアおよび任意非シークレットデータを含みます。データを暗号化するリクエストに暗号化コンテキストを含めると、 は暗号化コンテキストを暗号化されたデータに AWS KMS 暗号化バインドします。データを復号するには、同じ暗号化コンテキストに渡す必要があります。

Amazon Keyspaces は、すべての暗号化オペレーションで同じ AWS KMS 暗号化コンテキストを使用します。カスタマーマネージドキーを使用して Amazon Keyspaces テーブルを保護する場合は、暗号化コンテキストを使用して、監査の記録やログ内でカスタマーマネージドキーの使用を特定することができます。これは、AWS CloudTrail のログや Amazon CloudWatch Logs などのログにもプレーンテキストで表示されます。

へのリクエストでは AWS KMS、Amazon Keyspaces は 3 つのキーと値のペアを持つ暗号化コンテキストを使用します。

```
"encryptionContextSubset": {
    "aws:cassandra:keyspaceName": "my_keyspace",
    "aws:cassandra:tableName": "mytable"
    "aws:cassandra:subscriberId": "111122223333"
}
```

- Keyspace (キー空間) — 1 つ目のキーバリューペアは、Amazon Keyspaces により暗号化されるテーブルが含まれているキー空間を識別します。キーは、aws:cassandra:keyspaceName です。この値は、このキー空間の名前です。

  ```
  "aws:cassandra:keyspaceName": "<keyspace-name>"
  ```

例:

```
"aws:cassandra:keyspaceName": "my_keyspace"
```

- Table (テーブル) - 2 つ目のキーバリューペアは、Amazon Keyspaces により暗号化されるテーブルを識別します。キーは、aws:cassandra:tableName です。この値は、テーブルの名前です。

```
"aws:cassandra:tableName": "<table-name>"
```

例:

```
"aws:cassandra:tableName": "my_table"
```

- アカウント - 3 番目のキーバリューペアは、 AWS アカウントを識別します。キーは、aws:cassandra:subscriberId です。値は、アカウント ID です。

```
"aws:cassandra:subscriberId": "<account-id>"
```

例:

```
"aws:cassandra:subscriberId": "111122223333"
```

ステップ 6: でモニタリングを設定する AWS CloudTrail

カスタマーマネージドキーを使用して Amazon Keyspaces テーブルを保護する場合は、 AWS CloudTrail ログを使用して、Amazon Keyspaces が AWS KMS ユーザーに代わって に送信するリクエストを追跡できます。

このセクションでは、GenerateDataKey、DescribeKey、Decrypt、および CreateGrant の各リクエストについて説明します。さらに、Amazon Keyspaces では、テーブルを削除するときに RetireGrant オペレーションを使用して権限を削除します。

> **Note**
>
> Amazon Keyspaces を使用する場合、一部のオペレーションでは invokedByフィールドがの CloudTrail イベントが発生することがありますdynamodb.amazonaws.com。これは想定

され、Amazon Keyspaces が Amazon DynamoDB と統合してサービスを提供するため発生します。

GenerateDataKey

Amazon Keyspaces では、保管中のデータを暗号化するために一意のテーブルキーが作成します。テーブルの KMS キー AWS KMS を指定する GenerateDataKey リクエストを に送信します。

GenerateDataKey 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは Amazon Keyspaces サービスアカウントです。このパラメータには、カスタマーマネージドキーの Amazon リソースネーム (ARN)、256 ビットキーを必要とするキー指定子、および、キー空間とテーブルと AWS アカウントを識別する暗号化コンテキストが含まれます。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-04-16T04:56:05Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keySpec": "AES_256",
        "encryptionContext": {
            "aws:cassandra:keyspaceName": "my_keyspace",
            "aws:cassandra:tableName": "my_table",
            "aws:cassandra:subscriberId": "123SAMPLE012"
        },
        "keyId": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
    },
    "responseElements": null,
    "requestID": "5e8e9cb5-9194-4334-aacc-9dd7d50fe246",
    "eventID": "49fccab9-2448-4b97-a89d-7d5c39318d6f",
    "readOnly": true,
    "resources": [
```

```
        {
            "accountId": "123SAMPLE012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123SAMPLE012",
    "sharedEventID": "84fbaaf0-9641-4e32-9147-57d2cb08792e"
}
```

DescribeKey

Amazon Keyspaces では、DescribeKey オペレーションを使用して、選択した KMS キーがアカウントとリージョンに存在するかどうかが判断されます。

DescribeKey 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは Amazon Keyspaces サービスアカウントです。このパラメータには、カスタマーマネージドキーの ARN と、256 ビットキーを要求するキー識別子が含まれます。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAZ3FNIIVIZZ6H7CFQG",
        "arn": "arn:aws:iam::123SAMPLE012:user/admin",
        "accountId": "123SAMPLE012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "admin",
        "sessionContext": {
            "sessionIssuer": {},
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-04-16T04:55:42Z"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-04-16T04:55:58Z",
```

```
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
    },
    "responseElements": null,
    "requestID": "c25a8105-050b-4f52-8358-6e872fb03a6c",
    "eventID": "0d96420e-707e-41b9-9118-56585a669658",
    "readOnly": true,
    "resources": [
        {
            "accountId": "123SAMPLE012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123SAMPLE012"
}
```

Decrypt

Amazon Keyspaces のテーブルにアクセスすると、階層内でそのテーブルキーの下のキーの復号化を可能にするために、Amazon Keyspaces によりテーブルキーが復号化される必要があります。次に、テーブル内のデータを復号化します。テーブルキーを復号するために、Amazon Keyspaces はテーブルの KMS キー AWS KMS を指定する Decrypt リクエストを に送信します。

Decrypt 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは、テーブルにアクセスする のプリンシパル AWS アカウント です。パラメータには、暗号化されたテーブルキー (暗号文 BLOB として) と、テーブルを識別する暗号化コンテキストが含まれ、AWS アカウント. は暗号文からカスタマーマネージドキーの ID AWS KMS を取得します。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
```

```
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-04-16T05:29:44Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "encryptionContext": {
            "aws:cassandra:keyspaceName": "my_keyspace",
            "aws:cassandra:tableName": "my_table",
            "aws:cassandra:subscriberId": "123SAMPLE012"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "50e80373-83c9-4034-8226-5439e1c9b259",
    "eventID": "8db9788f-04a5-4ae2-90c9-15c79c411b6b",
    "readOnly": true,
    "resources": [
        {
            "accountId": "123SAMPLE012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123SAMPLE012",
    "sharedEventID": "7ed99e2d-910a-4708-a4e3-0180d8dbb68e"
}
```

CreateGrant

カスタマーマネージドキーを使用して Amazon Keyspaces テーブルを保護する場合、Amazon Keyspaces では、権限を使用して、サービスによるデータの継続的保護とメンテナンスおよび耐久タスクの実行を許可します。これらの権限は、AWS 所有のキー では不要です。

Amazon Keyspaces により作成される権限はテーブルごとに固有となります。CreateGrant リクエストのプリンシパルは、テーブルを作成したユーザーです。

CreateGrant 演算を記録するイベントは、次のようなサンプルイベントになります。このパラメータには、そのテーブルのカスタマーマネージドキーの ARN、被付与 (グランティー) プリンシパルと使用停止プリンシパル (Amazon Keyspaces サービス)、およびこの権限の対象となるオペレーションが含まれます。また、指定された暗号化コンテキストを使用するすべての暗号化オペレーションを必要とする制約も含まれています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAZ3FNIIVIZZ6H7CFQG",
        "arn": "arn:aws:iam::arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111:user/admin",
        "accountId": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "admin",
        "sessionContext": {
            "sessionIssuer": {},
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-04-16T04:55:42Z"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-04-16T05:11:10Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "a7d328af-215e-4661-9a69-88c858909f20",
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "RetireGrant"
```

```
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:cassandra:keyspaceName": "my_keyspace",
                "aws:cassandra:tableName": "my_table",
                "aws:cassandra:subscriberId": "123SAMPLE012"
            }
        },
        "retiringPrincipal": "cassandratest.us-east-1.amazonaws.com",
        "granteePrincipal": "cassandratest.us-east-1.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "18e4235f1b07f289762a31a1886cb5efd225f069280d4f76cd83b9b9b5501013"
    },
    "requestID": "b379a767-1f9b-48c3-b731-fb23e865e7f7",
    "eventID": "29ee1fd4-28f2-416f-a419-551910d20291",
    "readOnly": false,
    "resources": [
        {
            "accountId": "123SAMPLE012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123SAMPLE012"
}
```

Amazon Keyspaces での転送時の暗号化

Amazon Keyspaces では、Transport Layer Security (TLS) を使用した安全な接続しか許容されません。転送時の暗号化では、Amazon Keyspaces との間で送受信するときにデータを暗号化することによって、データ保護のレイヤーを追加します。組織のポリシー、業界や政府の規制、またはコンプライアンス要件によって、ネットワークを介したデータ転送時にアプリケーションのデータセキュリティを高めるために転送時の暗号化の使用が求められることがあります。

TLS を使用した Amazon Keyspaces への cqlsh 接続を暗号化する方法については、「the section called “TLS の cqlsh 接続を手動で設定する方法”」を参照してください。TLS 暗号化をクライアントドライバーとともに使用する方法については、「the section called “Cassandra クライアントドライバーの使用”」を参照してください。

Amazon Keyspaces におけるインターネットワークトラフィックプライバシー

このトピックでは、Amazon Keyspaces (Apache Cassandra 用) がオンプレミスアプリケーションから Amazon Keyspaces への接続、および Amazon Keyspaces と同じ 内の他の AWS リソース間の接続を保護する方法について説明します AWS リージョン。

サービスとオンプレミスのクライアントおよびアプリケーションとの間のトラフィック

プライベートネットワークと の間には 2 つの接続オプションがあります AWS。

- AWS Site-to-Site VPN 接続。詳細については、「 AWS Site-to-Site VPNユーザーガイド」の「What is AWS Site-to-Site VPN ?」(とは ?) を参照してください。
- AWS Direct Connect 接続。詳細については、「 AWS Direct Connectユーザーガイド」の「What is AWS Direct Connect ?」(とは ?) を参照してください。

マネージドサービスである Amazon Keyspaces (Apache Cassandra 向け) 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 AWS 環境を設計するには、「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 Keyspaces にアクセスします。クライアントは以下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。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。またはAWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Amazon Keyspaces では、2 つのクライアントリクエスト認証方法がサポートされています。1 つ目の方法では、サービス固有の認証情報を使用します。これは、特定の IAM ユーザーに対して生成されたパスワードベースの認証情報です。IAM コンソール、、または AWS API を使用して AWS CLIパスワードを作成および管理できます。詳細については、「Using IAM with Amazon Keyspaces (Amazon Keyspaces での IAM の使用)」 を参照してください。

2 つ目の方法は、Cassandra 用のオープンソース DataStax Java ドライバーに対して認証プラグインを使用します。このプラグインでは、IAM ユーザー、ロール、およびフェデレーテッドアイデンティティにより、AWS 署名バージョン 4 署名プロセス (SigV4) を使用して、Amazon Keyspaces (Apache Cassandra 向け) API リクエストに認証情報を追加することができます。詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

同じリージョン内の AWS リソース間のトラフィック

インターフェイス VPC エンドポイントでは、Amazon VPC で実行されている仮想プライベートクラウド (VPC) と Amazon Keyspaces 間のプライベート通信ができます。インターフェイス VPC エンドポイントは AWS PrivateLink、VPCs と AWS サービス間のプライベート通信を可能にする AWS サービスです。 は、ネットワークトラフィックが Amazon ネットワークを離れないように、VPC 内のプライベート IPs で Elastic Network Interface を使用してこれ AWS PrivateLink を有効にします。インターフェイス VPC エンドポイントには、インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続は必要ありません。詳細については、「Amazon Virtual Private CloudAmazon (仮想プライベートクラウド)」と「Interface VPC endpoints (AWS PrivateLink) (インターフェイス VPC エンドポイント ())」を参照してください。エンドポイントポリシーの例については、「the section called “Amazon Keyspaces 用インターフェイス VPC エンドポイントの使用”」を参照してください。

AWS Identity and Access Management Amazon Keyspaces 用

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、認証 (サインイン) され、かつ Amazon Keyspaces リソースを使用する認可 を受ける (許可がある) ことができるユーザーを管理します。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

- 対象者
- 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 Keyspaces で IAM が機能する仕組み
- Amazon Keyspaces のアイデンティティベースポリシーの例
- AWS Amazon Keyspaces の マネージドポリシー
- Amazon Keyspaces のアイデンティティとアクセスに関するトラブルシューティング
- Amazon Keyspaces のサービスリンクロールの使用

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon Keyspaces で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Amazon Keyspaces サービスを使用する場合は、管理者から必要な権限と認証情報が与えられます。多くの Amazon Keyspaces 機能を使用して作業を行う場合は、追加の権限が必要になる場合があります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役に立ちます。Amazon Keyspaces の機能にアクセスできない場合は、「Amazon Keyspaces のアイデンティティとアクセスに関するトラブルシューティング」を参照してください。

サービス管理者 – 社内の Amazon Keyspaces リソースを担当している場合は、通常、Amazon Keyspaces への完全アクセス権を持っています。サービスのユーザーがどの Amazon Keyspaces 機能やリソースにアクセスできるかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で Amazon Keyspaces と IAM を併用する方法の詳細については、「Amazon Keyspaces で IAM が機能する仕組み」を参照してください。

IAM 管理者 – IAM 管理者は、Amazon Keyspaces へのアクセスを管理するポリシーの詳しい作成方法を理解しておくことが推奨されます。IAM で使用できる Amazon Keyspaces アイデンティティベースのポリシーの例を表示するには、「Amazon Keyspaces のアイデンティティベースポリシーの例」を参照してください。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして 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によって認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook 認証情報は、フェデレーション ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して にアクセスすると、間接的 AWS にロールを引き受けることになります。

ユーザーの種類に応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、 AWS サインイン ユーザーガイドの「 へのサインイン方法 AWS アカウント」を参照してください。

AWS プログラムで にアクセスする場合、 はソフトウェア開発キット (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストを暗号化して署名します。AWS ツールを使用しない場合は、自分でリクエストに署名する必要があります。リクエストに自分で署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエストに対するAWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。たとえば、 では、アカウントのセキュリティを高めるために多要素認証 (MFA) を使用する AWS ことをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「IAM のAWS 多要素認証」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、アカウント内のすべての およびリソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。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。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「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。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内の ID です。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーション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「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。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 AWS アカウント を持つ 内の ID です。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受けるには AWS Management Console、ユーザーから IAM ロール (コンソール) に切り替えることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「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 ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールについては、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity

Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。

- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、(プロキシとしてロールを使用する代わりに) リソースに直接ポリシーをアタッチできます。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。
- クロスサービスアクセス — 一部の は他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用してアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を 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 と組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。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。
  - サービスロール - 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「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。
  - サービスにリンクされたロール – 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、サービスリンク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。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。EC2 インスタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」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 AWS 、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ときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) 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。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。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構造と内容の詳細については、「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 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。

デフォルトでは、ユーザーやロールに権限はありません。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。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

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ポリシーは、IAM ユーザーグループ、ユーザーのグループ、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。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたは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。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制御できます。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は、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。

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。

アクセスコントロールリスト (ACL)

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 S3、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的でない追加のポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エ

ンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。Principal フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。

- サービスコントロールポリシー (SCPs) – SCPsは、 の組織または組織単位 (OU) の最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所有する複数の AWS アカウント をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー (SCP)」を参照してください。
- リソースコントロールポリシー (RCP) – RCP は、所有する各リソースにアタッチされた IAM ポリシーを更新することなく、アカウント内のリソースに利用可能な最大数のアクセス許可を設定するために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースのアクセス許可を制限し、組織に属しているかどうかにかかわらず AWS アカウントのルートユーザー、 を含む ID の有効なアクセス許可に影響を与える可能性があります。RCP をサポートする のリストを含む Organizations と RCP の詳細については、AWS Organizations RCPs「リソースコントロールポリシー (RCPs」を参照してください。 AWS のサービス
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。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権限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 Keyspaces で IAM が機能する仕組み

IAM を使用して Amazon Keyspaces へのアクセスを管理する前に、Amazon Keyspaces 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 Keyspaces およびその他の AWS のサービスが IAM 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。

トピック

- Amazon Keyspaces のアイデンティティベースポリシー
- Amazon Keyspaces のリソースベースポリシー
- Amazon Keyspaces タグに基づいた認可
- Amazon Keyspaces の IAM ロール

Amazon Keyspaces のアイデンティティベースポリシー

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、またアクションを許可または拒否する条件を指定できます。Amazon Keyspaces は、特定のアクション、リソース、および条件キーをサポートしています。JSON ポリシーで使用するすべての要素については、『IAM ユーザーガイド』の「IAM JSON policy elements reference (IAM JSON ポリシーエレメントのリファレンス)」を参照してください。

IAM アクセス権限ポリシーに使用できる Amazon Keyspaces サービス固有のリソースとアクション、および条件コンテキストキーを確認するには、『Service Authorization Reference (サービス認可リファレンス)』の「Actions, Resources, and Condition Keys for Amazon Keyspaces (for Apache Cassandra) (Amazon Keyspaces (Apache Cassandra 向け)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。

アクション

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。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは依存アクションと呼ばれます。

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。

Amazon Keyspaces のポリシーアクションは、アクションの前にプレフィックス `cassandra:` を使用します。例えば、Amazon Keyspaces CREATE CQL ステートメントを使用して Amazon Keyspaces のキースペースを作成するための権限を付与するには、ポリシーに `cassandra:Create` アクションを含めます。ポリシーステートメントには`Action` または `NotAction` 要素を含める必要があります。Amazon Keyspaces は、このサービスで実行できるタスクを記述する独自のアクションのセットを定義します。

単一のステートメントに複数のアクションを指定するには次のようにコンマで区切ります。

```
"Action": [
      "cassandra:CREATE",
      "cassandra:MODIFY"
          ]
```

Amazon Keyspaces アクションのリストを確認するには、『Service Authorization Reference (サービス認可リファレンス)』の「Actions Defined by Amazon Keyspaces (for Apache Cassandra) (Amazon Keyspaces (Apache Cassandra 向け) で定義されるアクション)」を参照してください。

リソース

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。ステートメントには`Resource` または `NotResource` 要素を含める必要があります。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。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限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": "*"
```

Amazon Keyspaces では、IAM アクセス許可の `Resource`要素でキースペース、テーブル、ストリームを使用できます。

> **Note**
>
> Amazon Keyspaces のユーザーキースペースとテーブルにアクセスするには、IAM ポリシーにシステムテーブルに対するcassandra:Selectアクセス許可が含まれている必要があります。
>
> ```
> arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/system*
> ```
>
> これは、次のシナリオに適用されます。
>
> - AWS マネジメントコンソールへのアクセス
> - 、、GetKeyspace、 などの SDK リソースオペレーション GetTable ListKeyspaces ListTables
> - 標準 Apache Cassandra クライアントドライバー接続。ドライバーは接続の初期化中にシステムテーブルを自動的に読み取るため
>
> システムテーブルは読み取り専用であり、変更できません。

Amazon Keyspaces のキースペースリソースには次の ARN があります。

```
arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/${keyspaceName}/
```

Amazon Keyspaces のテーブルリソースには次の ARN があります。

```
arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/${keyspaceName}/table/
${tableName}
```

Amazon Keyspaces ストリームリソースには、次の ARN があります。

```
arn:${Partition}:cassandra:{Region}:${Account}:/keyspace/${keyspaceName}/table/
${tableName}/stream/${streamLabel}
```

ARN の形式の詳細については、「Amazon リソースネーム (ARNsと AWS サービス名前空間」を参照してください。

例えば、ステートメントで mykeyspace キースペースを指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/"
```

特定のアカウントに属するすべてのキースペースを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:cassandra:aws-region:111122223333:/keyspace/*"
```

リソースを作成するためのアクションなど、Amazon Keyspaces アクションには特定のリソースで実行できないものがあります。このような場合はワイルドカード *を使用する必要があります。

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

たとえば、 mytableの IAM プリンシパルにアクセスSELECT許可を付与するにはmykeyspace、プリンシパルに mytableと の両方を読み取るアクセス許可が必要ですkeyspace/system*。複数リソースを単一ステートメントで指定するには、ARN をカンマで区切ります。

```
"Resource": "arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
```

Amazon Keyspaces リソースのタイプとその ARN のリストを確認するには、「Service Authorization Reference」(サービス認可リファレンス) の「Resources Defined by Amazon Keyspaces (for Apache Cassandra)」(Amazon Keyspaces (Apache Cassandra 向け) で定義されるリソース) を参照してください。どのアクションで各リソースの ARN を指定できるかについては、「Actions Defined by Amazon Keyspaces (for Apache Cassandra)」(Amazon Keyspaces (Apache Cassandra 向け) で定義されるアクション) を参照してください。

条件キー

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 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。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。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS では AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与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「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートしています。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。

Amazon Keyspaces では独自の条件キーが定義されており、また一部のグローバル条件キーの使用がサポートされています。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。

すべての Amazon Keyspaces アクションでは、aws:RequestTag/${TagKey}、aws:ResourceTag/${TagKey} および aws:TagKeys 条件キーがサポートされます。詳細については、「the section called “ タグに基いた Amazon Keyspaces リソースアクセス”」を参照してください。

Amazon Keyspaces の条件キーのリストを確認するには、『Service Authorization Reference (サービス認可リファレンス)』の「Condition Keys for Amazon Keyspaces (for Apache Cassandra)」(Amazon Keyspaces (Apache Cassandra 向け) の条件キー) を参照してください。どのアクションおよびリソースと条件キーを使用できるかについては、「Actions Defined by Amazon Keyspaces (for Apache Cassandra) (Amazon Keyspaces (Apache Cassandra 向け)」で定義されるアクション) を参照してください。

例

Amazon Keyspaces のアイデンティティベースポリシーの例は、「Amazon Keyspaces のアイデンティティベースポリシーの例」でご確認ください。

Amazon Keyspaces のリソースベースポリシー

Amazon Keyspaces 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。詳細なリソースベースポリシーのページの例については、「https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html」を参照してください。

Amazon Keyspaces タグに基づいた認可

タグを使用して Amazon Keyspaces リソースへのアクセスを管理することができます。タグに基づいてリソースアクセスを管理するには、`cassandra: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。Amazon Keyspaces リソースのタグ付けの詳細については、「the section called “タグの操作”」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースのポリシーの例を表示するには、「 タグに基いた Amazon Keyspaces リソースアクセス」を参照してください。

Amazon Keyspaces の IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のエンティティです。

Amazon Keyspaces での一時認証情報の使用

一時的な認証情報を使用して、フェデレーションでサインイン、IAM ロールを引き受ける、またはクロスアカウントロールを引き受けることができます。一時的なセキュリティ認証情報を取得するには、AssumeRole や GetFederationToken などの AWS STS API オペレーションを呼び出します。

Amazon Keyspaces は、以下の言語で Github リポジトリから入手できる AWS Signature Version 4 (SigV4) 認証プラグインでの一時的な認証情報の使用をサポートしています。

- Java: https://github.com/aws/aws-sigv4-auth-cassandra-java-driver-plugin。
- Node.js: https://github.com/aws/aws-sigv4-auth-cassandra-nodejs-driver-plugin。
- Python: https://github.com/aws/aws-sigv4-auth-cassandra-python-driver-plugin。
- Go: https://github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin。

認証プラグインを実装してプログラムによって Amazon Keyspaces にアクセスする方法の例とチュートリアルについては、「the section called “Cassandra クライアントドライバーの使用”」を参照してください。

サービスにリンクされた役割

サービスにリンクされたロールを使用すると、 AWS サービスは他の サービスのリソースにアクセスして、ユーザーに代わってアクションを実行できます。サービスリンクロールは IAM アカウント内に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

Amazon Keyspaces のサービスリンクロールの作成または管理の詳細については、「the section called “サービスにリンクされたロールの使用”」を参照してください。

サービス役割

Amazon Keyspaces ではサービスロールがサポートされていません。

Amazon Keyspaces のアイデンティティベースポリシーの例

デフォルトでは、IAM ユーザーおよびロールには Amazon Keyspaces リソースを作成または変更する権限はありません。また、コンソール、CQLSH AWS CLI、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらの権限が必要な IAM ユーザーまたはグループにそのポリシーをアタッチする必要があります。

これらの JSON ポリシードキュメント例を使用して IAM の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「JSON タブでのポリシーの作成」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 Keyspaces コンソールの使用
- 自分の権限の表示をユーザーに許可する
- Amazon Keyspaces テーブルへのアクセス
- タグに基いた Amazon Keyspaces リソースアクセス

ポリシーに関するベストプラクティス

ID 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 Keyspaces 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を実行すると、 AWS アカウントに料金が発生する可能性があります。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する – 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 AWS 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 Access Analyzer 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 Access Analyzer 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、セキュリティを強化するために MFA を有効にします。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 Keyspaces コンソールの使用

Amazon Keyspaces では、Amazon Keyspaces コンソールにアクセスするための特定の権限は必要ありません。の Amazon Keyspaces リソースの詳細を一覧表示および表示するには、少なくとも読み取り専用のアクセス許可が必要です AWS アカウント。最小限必要な許可よりも厳しく制限されたアイデンティティベースポリシーを作成すると、そのポリシーを添付したエンティティ (IAM 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。

Amazon Keyspaces コンソールにアクセスするためのエンティティには、2 つの AWS マネージドポリシーを使用できます。

- AmazonKeyspacesReadOnlyAccess_v2 – Amazon Keyspaces への読み取り専用アクセス権を付与するポリシーです。
- AmazonKeyspacesFullAccess – Amazon Keyspaces へのアクセスを許可し、すべての機能へのフルアクセスを認めるポリシーです。

Amazon Keyspaces のマネージドポリシーの詳細については、「the section called “AWS マネージドポリシー”」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 Keyspaces テーブルへのアクセス

> **Note**
>
> Amazon Keyspaces のユーザーキースペースとテーブルにアクセスするには、IAM ポリシーにシステムテーブルに対するcassandra:Selectアクセス許可が含まれている必要があります。
>
> ```
> arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/system*
> ```
>
> これは、次のシナリオに適用されます。
>
> - AWS マネジメントコンソールへのアクセス
> - 、、GetKeyspace、 などの SDK リソースオペレーション GetTable ListKeyspaces ListTables
> - 標準 Apache Cassandra クライアントドライバー接続。ドライバーは接続の初期化中にシステムテーブルを自動的に読み取るため
>
> システムテーブルは読み取り専用であり、変更できません。

次のサンプルポリシーは、Amazon Keyspaces のシステムテーブルへの読み取り専用 (SELECT) アクセス許可を付与しています。すべてのサンプルで、Amazon リソースネーム (ARN) のリージョンとアカウント ID を実際の値と置き換えてください。

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

```
         "Effect":"Allow",
         "Action":[
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

次のサンプルポリシーは、キースペース mykeyspace のユーザーテーブル mytable への読み取り専用アクセスを追加します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

以下のサンプルポリシーでは、ユーザーテーブルへの読み取り/書き込みアクセス権と、システムテーブルへの読み取りアクセス権を割り当てます。

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

```
         ],
         "Resource":[
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

次のサンプルポリシーでは、ユーザーがキースペース mykeyspace でテーブルを作成できます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Create",
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/*",
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

次のサンプルポリシーは、システムテーブルへの読み取りアクセスを割り当てますが、ユーザーテーブル へのアクセス SELECT (読み取り) と MODIFY (書き込み) を制限しますmytable。

```
{
  "Version": "2012-10-17",
  "Statement": [
   {
     "Effect": "Allow",
     "Action": [
      "cassandra:Select"
     ],
     "Resource": [
      "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
     ]
```

```
    },
    {
      "Effect": "Deny",
      "Action": [
       "cassandra:Select",
       "cassandra:Modify"
      ],
      "Resource": [
       "arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/table/mytable"
      ]
    }
  ]
}
```

タグに基いた Amazon Keyspaces リソースアクセス

アイデンティティベースポリシーの条件を使用して、タグに基づいた Amazon Keyspaces リソースへのアクセスを制御することができます。これらのポリシーは、アカウント内のキースペースとテーブルの可視性を制御します。システムテーブルのタグベースのアクセス許可は、リクエストが AWS SDK を使用して行われた場合と、Cassandra ドライバーおよび開発者ツールを介した Cassandra クエリ言語 (CQL) API コールでは動作が異なることに注意してください。

- タグベースのアクセスの使用時に AWS SDK でリクエストを行い、`List` リソースリクエストと `Get` リソースリクエストを行うには、呼び出し元にはシステムテーブルへの読み取りアクセス権限が必要です。たとえば、`GetTable` オペレーションでシステムテーブルからデータを読み取るには `Select` アクション権限が必要です。呼び出し元に特定のテーブルに対するタグベースのアクセス権限しかない場合、システムテーブルへの追加アクセスが必要な操作は失敗します。
- 確立されている Cassandra ドライバーの動作との互換性を保つため、Cassandra ドライバーと開発者ツールを介して Cassandra クエリ言語 (CQL) API コールでシステムテーブルに対して操作を実行するとき、タグベースの承認ポリシーは適用されません。

次の例では、テーブルの `Owner` にユーザーのユーザー名の値が含まれている場合に、そのテーブルを表示するための権限をユーザーに与えるポリシーを作成する方法を説明します。この例では、システムテーブルへの読み取りアクセス権限も与えられています。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ReadOnlyAccessTaggedTables",
```

```
         "Effect":"Allow",
         "Action":"cassandra:Select",
         "Resource":[
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/table/*",
            "arn:aws:cassandra:aws-region:111122223333:/keyspace/system*"
         ],
         "Condition":{
            "StringEquals":{
               "aws:ResourceTag/Owner":"${aws:username}"
            }
         }
      }
   ]
}
```

このポリシーはアカウントの IAM ユーザーにアタッチできます。richard-roe という名前のユーザーが Amazon Keyspaces テーブルを表示しようとすると、そのテーブルには Owner=richard-roe または owner=richard-roe というタグが付きます。それ以外の場合、アクセスは拒否されます。条件キー名では大文字と小文字は区別されないため、条件タグキー Owner は Owner と owner に一致します。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。

次のポリシーでは、テーブルの Owner にユーザーのユーザー名の値が含まれている場合に、タグ付きのテーブルを作成するための権限をユーザーに付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
          "Sid": "CreateTagTableUser",
          "Effect": "Allow",
          "Action": [
              "cassandra:Create",
              "cassandra:TagResource"
          ],
          "Resource": "arn:aws:cassandra:aws-region:111122223333:/keyspace/mykeyspace/
table/*",
          "Condition":{
             "StringEquals":{
                "aws:RequestTag/Owner":"${aws:username}"
             }
          }
```

```
      }
   ]
}
```

AWS Amazon Keyspaces の マネージドポリシー

AWS 管理ポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。 AWS 管理ポリシーは、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できるように、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ように設計されています。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケースに固有のカスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS 管理ポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。 AWS は、新しい が起動されるか、新しい API オペレーション AWS のサービス が既存のサービスで使用できるようになったときに、 AWS 管理ポリシーを更新する可能性が最も高くなります。

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」を参照してください。

AWS マネージドポリシー: AmazonKeyspacesReadOnlyAccess_v2

`AmazonKeyspacesReadOnlyAccess_v2` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、Amazon Keyspaces への読み取り専用アクセス権を付与するもので、プライベート VPC エンドポイント経由で接続する場合に必要な権限が含まれます。

権限の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `Amazon Keyspaces` – Amazon Keyspaces への読み取り専用アクセスを提供します。

- `Amazon Keyspaces CDC streams` – プリンシパルに Amazon Keyspaces CDC ストリームの表示を許可します。
- `Application Auto Scaling` — Application Auto Scaling の設定をプリンシパルが表示できるようにします。これは、テーブルにアタッチされているオートスケーリングポリシーをユーザーが表示できる場合に必須です。
- `CloudWatch` - CloudWatch で設定されたメトリクスデータとアラームをプリンシパルが表示できるようにします。これは、テーブルに対して設定された請求対象テーブルサイズと CloudWatch アラームをユーザーが表示できる場合に必須です。
- `AWS KMS` – プリンシパルが で設定されたキーを表示できるようにします AWS KMS。これは、ユーザーがアカウントで作成および管理している AWS KMS キーを表示して、Amazon Keyspaces に割り当てられたキーが有効な対称暗号化キーであることを確認するために必要です。
- `Amazon EC2` — VPC エンドポイントを介して Amazon Keyspaces に接続するプリンシパルで、Amazon EC2 インスタンスの VPC にエンドポイントとネットワークインターフェイスの情報をクエリできます。Amazon Keyspaces が接続負荷分散に使用される `system.peers` テーブルで利用可能なインターフェイス VPC エンドポイントを検索して保存できるようにするには、Amazon EC2 インスタンスへのこの読み取り専用アクセスが必要です。

JSON 形式のポリシーを確認するには、「AmazonKeyspacesReadOnlyAccess_v2」を参照してください。

AWS マネージドポリシー: AmazonKeyspacesReadOnlyAccess

`AmazonKeyspacesReadOnlyAccess` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、Amazon Keyspaces への読み取り専用アクセス権を付与するものです。

権限の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `Amazon Keyspaces` – Amazon Keyspaces への読み取り専用アクセスを提供します。
- `Amazon Keyspaces CDC streams` – プリンシパルに Amazon Keyspaces CDC ストリームの表示を許可します。

- `Application Auto Scaling` — Application Auto Scaling の設定をプリンシパルが表示できるようにします。これは、テーブルにアタッチされているオートスケーリングポリシーをユーザーが表示できる場合に必須です。
- `CloudWatch` - CloudWatch で設定されたメトリクスデータとアラームをプリンシパルが表示できるようにします。これは、テーブルに対して設定された請求対象テーブルサイズと CloudWatch アラームをユーザーが表示できる場合に必須です。
- `AWS KMS` – プリンシパルが で設定されたキーを表示できるようにします AWS KMS。これは、ユーザーがアカウントで作成および管理している AWS KMS キーを表示して、Amazon Keyspaces に割り当てられたキーが有効な対称暗号化キーであることを確認するために必要です。

ポリシーを `JSON` フォーマットで確認するには、「AmazonKeyspacesReadOnlyAccess」を参照してください。

AWS マネージドポリシー: AmazonKeyspacesFullAccess

`AmazonKeyspacesFullAccess` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、Amazon Keyspaces への無制限の管理者アクセスを許可する管理者用権限を付与するものです。

権限の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `Amazon Keyspaces` — プリンシパルが任意の Amazon Keyspaces リソースにアクセスしてすべてのアクションを実行できるようにします。
- `Application Auto Scaling` — プリンシパルが Amazon Keyspaces テーブルのオートスケーリングポリシーの作成、表示、削除を実行できるようにします。これは、管理者が Amazon Keyspaces テーブルのオートスケーリングポリシーを管理できる場合に必須です。
- `CloudWatch` — プリンシパルが請求可能なテーブルサイズを表示できるほか、Amazon Keyspaces 自動スケーリングポリシーの CloudWatch アラームの作成、表示、削除を実行できるようにします。これは、管理者が請求対象テーブルサイズを表示し、CloudWatch ダッシュボードを作成できる場合に必須です。

- IAM — 以下の機能が有効なとき、Amazon Keyspaces は IAM でサービスにリンクされたロールを自動的に作成できます。
  - Amazon Keyspaces CDC streams – 管理者がテーブルのストリームを有効にすると、Amazon Keyspaces はサービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesCDC を作成し、ユーザーに代わって CloudWatch メトリクスをアカウントに発行します。
  - Application Auto Scaling – 管理者がテーブルの Application Auto Scaling を有効にすると、Amazon Keyspaces はサービスにリンクされたロール AWSServiceRoleForApplicationAutoScaling_CassandraTable を作成し、ユーザーに代わって自動スケーリングアクションを実行します。
  - Amazon Keyspaces multi-Region replication – 管理者が新しいマルチリージョンキースペースを作成するか、既存の単一リージョンキースペース AWS リージョン に新しい を追加すると、Amazon Keyspaces はサービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesReplication を作成し、ユーザーに代わって選択したリージョンにテーブル、データ、メタデータのレプリケーションを実行します。
- AWS KMS — AWS KMSで設定されたキーをプリンシパルが表示できるようにします。これは、ユーザーがアカウントで作成および管理している AWS KMS キーを表示して、Amazon Keyspaces に割り当てられたキーが有効な対称暗号化キーであることを確認するために必要です。
- Amazon EC2 — VPC エンドポイントを介して Amazon Keyspaces に接続するプリンシパルで、Amazon EC2 インスタンスの VPC にエンドポイントとネットワークインターフェイスの情報をクエリできます。Amazon Keyspaces が接続負荷分散に使用される system.peers テーブルで利用可能なインターフェイス VPC エンドポイントを検索して保存するために、Amazon EC2 インスタンスへのこの読み取り専用アクセス権限が必要です。

JSON 形式のポリシーを確認するには、「AmazonKeyspacesFullAccess」を参照してください。

AWS マネージドポリシー: KeyspacesCDCServiceRolePolicy

IAM エンティティに KeyspacesCDCServiceRolePolicy をアタッチすることはできません。このポリシーは、Amazon Keyspaces がユーザーに代わってアクションを実行できるようにするサービスにリンクされたロールにアタッチされます。詳細については、「the section called “CDC ストリーム”」を参照してください。

このポリシーは、ユーザーに代わって Amazon Keyspaces CDC ストリームメトリクスデータを CloudWatch に発行AWSServiceRoleForAmazonKeyspacesCDCするために必要なアクセス許可をサービスにリンクされたロールに付与します。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- CloudWatch – service-linked-roleの AWSServiceRoleForAmazonKeyspacesCDC が、ユーザーに代わって Amazon Keyspaces CDC ストリームから CloudWatch アカウントの にメトリクスデータをパブリッシュ"cloudwatch:namespace": "AWS/Cassandra"できるようにします。

JSON 形式のポリシーを確認するには、KeyspacesCDCServiceRolePolicy」を参照してください。

AWS マネージドポリシーに対する Amazon Keyspaces の更新

このサービスがこれらの変更の追跡を開始してからの Amazon Keyspaces の AWS マネージドポリシーの更新に関する詳細を表示します。このページへの変更に関する自動アラートについては、ドキュメント履歴 ページの RSS フィードを購読してください。

変更	説明	日付
KeyspacesCDCServiceRolePolicy – 新しいポリシー	Amazon Keyspaces は、ユーザーに代わって Amazon Keyspaces CDC ストリームメトリクスデータを CloudWatch に発行AWSServiceRoleForAmazonKeyspacesCDC するために必要なアクセス許可をサービスにリンクKeyspacesCDCServiceRolePolicy されたロールに付与する	2025 年 7 月 2 日

変更	説明	日付
	新しい管理ポリシーを追加しました。詳細については、「the section called “CDC ストリーム”」を参照してください。	
AmazonKeyspacesReadOnlyAccess_v2 – 既存のポリシーの更新	Amazon Keyspaces に新しいアクセス許可が追加され、IAM プリンシパルが Amazon Keyspaces CDC ストリームを表示できるようになりました。詳細については、「the section called “CDC ストリームを表示する”」を参照してください。	2025 年 7 月 2 日
AmazonKeyspacesReadOnlyAccess — 既存のポリシーに対する更新	Amazon Keyspaces に新しいアクセス許可が追加され、IAM プリンシパルが Amazon Keyspaces CDC ストリームを表示できるようになりました。詳細については、「the section called “CDC ストリームを表示する”」を参照してください。	2025 年 7 月 2 日

変更	説明	日付
AmazonKeyspacesFullAccess — 既存のポリシーに対する更新	Amazon Keyspaces は、サービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesCDC の KeyspacesCDCServiceRolePolicy マネージドポリシーを作成し、管理者がテーブルのストリームを有効にするときに必要なアクセス許可を追加しました。 Amazon Keyspaces は、サービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesCDC を使用して、ユーザーに代わって CloudWatch メトリクスをアカウントに発行します。詳細については、「the section called “CDC ストリーム”」を参照してください。	2025 年 7 月 2 日

変更	説明	日付
AmazonKeyspacesFullAccess — 既存のポリシーに対する更新	Amazon Keyspaces は、サービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesReplication KeyspacesReplicationServiceRolePolicy の を更新し、管理者が単一またはマルチリージョンのキースペース AWS リージョン に新しい を追加するときに必要なアクセス許可を追加しました。 Amazon Keyspaces は、サービスにリンクされたロールAWSServiceRoleForAmazonKeyspacesReplication を使用して、ユーザーに代わってテーブル、その設定、データをレプリケートします。詳細については、「the section called “マルチリージョンレプリケーション”」を参照してください。	2024 年 11 月 19 日

変更	説明	日付
AmazonKeyspacesFullAccess — 既存のポリシーに対する更新	Amazon Keyspaces は、管理者が単一またはマルチリージョンのキースペースに新しいリージョンを追加したときに、Amazon Keyspaces がサービスにリンクされたロールを作成できるようにする新しいアクセス許可を追加しました。 Amazon Keyspaces は、サービスにリンクされたロールを使用して、ユーザーに代わってデータレプリケーションタスクを実行します。詳細については、「the section called “マルチリージョンレプリケーション”」を参照してください。	2023 年 10 月 3 日

変更	説明	日付
AmazonKeyspacesReadOnlyAccess_v2 – 新しいポリシー	Amazon Keyspaces は、インターフェイス VPC エンドポイントを介して Amazon Keyspaces に接続し、Amazon EC2 インスタンスにアクセスしてネットワーク情報を検索するクライアントに読み取り専用アクセス許可を追加する新しいポリシーを作成しました。 Amazon Keyspaces は、接続負荷を分散するために、使用可能なインターフェイス VPC エンドポイントを `system.peers` テーブルに保存します。詳細については、「the section called “インターフェイス VPC エンドポイントの使用”」を参照してください。	2023 年 9 月 12 日

変更	説明	日付
AmazonKeyspacesFullAccess — 既存のポリシーに対する更新	Amazon Keyspaces で、管理者がマルチリージョンキー空間を作成する際に、Amazon Keyspaces でサービスにリンクされたロールを作成できる新しい権限を追加しました。 Amazon Keyspaces は、サービスにリンクされたロール`AWSServiceRoleForAmazonKeyspacesReplication` を使用して、ユーザーに代わってデータレプリケーションタスクを実行します。詳細については、「the section called “マルチリージョンレプリケーション”」を参照してください。	2023 年 6 月 5 日
AmazonKeyspacesReadOnlyAccess — 既存のポリシーに対する更新	Amazon Keyspaces で、ユーザーが CloudWatch を使用してテーブルの請求対象サイズを表示できる新しい権限を追加しました。 Amazon Keyspaces は Amazon CloudWatch と統合されているため、請求対象となるテーブルのサイズをモニタリングできます。詳細については、「the section called “Amazon Keyspaces のメトリクス”」を参照してください。	2022 年 7 月 7 日

変更	説明	日付
AmazonKeyspacesFullAccess — 既存のポリシーに対する更新	Amazon Keyspaces で、ユーザーが CloudWatch でテーブルの請求対象サイズを表示できる新しい権限を追加しました。 Amazon Keyspaces は Amazon CloudWatch と統合されているため、請求対象となるテーブルのサイズをモニタリングできます。詳細については、「the section called “Amazon Keyspaces のメトリクス”」を参照してください。	2022 年 7 月 7 日
AmazonKeyspacesReadOnlyAccess — 既存のポリシーに対する更新	Amazon Keyspaces は、ユーザーが Amazon Keyspaces の保管時の暗号化用に設定された AWS KMS キーを表示できるようにする新しいアクセス許可を追加しました。 Amazon Keyspaces 保管時の暗号化は と統合 AWS KMS され、保管時のデータの暗号化に使用される暗号化キーを保護および管理します。Amazon Keyspaces 用に設定された AWS KMS キーを表示するには、読み取り専用アクセス許可が追加されました。	2021 年 6 月 1 日

変更	説明	日付
AmazonKeyspacesFullAccess — 既存のポリシーに対する更新	Amazon Keyspaces は、ユーザーが Amazon Keyspaces の保管時の暗号化用に設定された AWS KMS キーを表示できるようにする新しいアクセス許可を追加しました。 Amazon Keyspaces 保管時の暗号化は と統合 AWS KMS され、保管時のデータの暗号化に使用される暗号化キーを保護および管理します。Amazon Keyspaces 用に設定された AWS KMS キーを表示するには、読み取り専用アクセス許可が追加されました。	2021 年 6 月 1 日
Amazon Keyspaces で変更の追跡が開始されました。	Amazon Keyspaces は、 AWS 管理ポリシーの変更の追跡を開始しました。	2021 年 6 月 1 日

Amazon Keyspaces のアイデンティティとアクセスに関するトラブルシューティング

次の情報は、Amazon Keyspaces と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に使用できます。

トピック

- Amazon Keyspaces でアクションを実行するための認可を受けていません。
- IAM ユーザーまたはロールを変更しましたが、その変更がすぐには反映されませんでした
- Amazon Keyspaces ポイントインタイムリカバリ (PITR) を使用してテーブルを復元できません。
- iam: PassRole を実行する権限がない
- 管理者として Amazon Keyspaces へのアクセスを他のユーザーに許可したい

- 自分の 以外のユーザーに Amazon Keyspaces リソース AWS アカウント へのアクセスを許可したい

Amazon Keyspaces でアクションを実行するための認可を受けていません。

でアクションを実行する権限がないと AWS Management Console 通知された場合は、管理者に連絡してサポートを依頼する必要があります。担当の管理者はお客様のユーザー名とパスワードを発行した人です。

以下のエラーの例は、mateojackson IAM ユーザーがコンソールを使用して####の詳細を表示したときに、そのテーブルの cassandra:*Select* 権限を持っていない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 cassandra:Select on resource: mytable
```

この場合、Mateo は、cassandra:*Select* アクションを使用して *mytable* リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。

IAM ユーザーまたはロールを変更しましたが、その変更がすぐには反映されませんでした

IAM ポリシーの変更が、Amazon Keyspaces への接続が既存で確立されているアプリケーションに反映されるまで、最長で 10 分かかる場合があります。アプリケーションで新しい接続が確立された場合は、IAM ポリシーの変更がすぐに反映されます。既存の IAM ユーザーまたはロールに変更を加えても、その変更がすぐに反映されない場合は、10 分ほど待つか、Amazon Keyspaces への接続を切って接続し直してください。

Amazon Keyspaces ポイントインタイムリカバリ (PITR) を使用してテーブルを復元できません。

ポイントインタイムリカバリ (PITR) を使用して Amazon Keyspaces テーブルの復元を試みた際に、復元プロセスが開始しても正常に完了しない場合は、復元プロセスに必要な権限の一部が設定されていない可能性があります。管理者に問い合わせて、Amazon Keyspaces でテーブルを復元できるようにポリシーを更新してもらう必要があります。

Amazon Keyspaces では、ユーザー権限に加えて、復元プロセス中にプリンシパルに代わってアクションを実行するための権限が必要になる場合があります。これは、テーブルが顧客管理キーで暗号化されている場合や、着信トラフィックを制限する IAM ポリシーを使用している場合です。

例えば、IAM ポリシーで条件キーを使用してソーストラフィックを特定のエンドポイントまたは IP 範囲に制限している場合、復元オペレーションは失敗します。プリンシパルの代わりに Amazon Keyspaces によってテーブルの復元オペレーションが実行されるようにするには、IAM ポリシーに `aws:ViaAWSService` グローバル条件キーを追加する必要があります。

テーブルを復元するための権限の詳細については、「the section called “復元用の IAM アクセス許可の設定”」を参照してください。

iam: PassRole を実行する権限がない

`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 Keyspaces 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、既存のロールをそのサービスに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権限が必要です。

以下の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用して Amazon Keyspaces でアクションを実行しようする場合に発生します。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権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
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。

管理者として Amazon Keyspaces へのアクセスを他のユーザーに許可したい

他のユーザーに Amazon Keyspaces へのアクセスを認めるには、アクセス権を必要とするユーザーやアプリケーションにアクセス許可を付与する必要があります。 AWS IAM Identity Center を使用してユーザーとアプリケーションを管理する場合は、アクセス許可セットをユーザーやグループに割り当て、それぞれのアクセスレベルを定義します。アクセス許可セットは、ユーザーまたはアプリケーションに関連付けられている IAM ロールに自動的に IAM ポリシーを作成して割り当てます。詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。

IAM アイデンティティセンターを使用していない場合は、アクセスを必要としているユーザーまたはアプリケーションの IAM エンティティ (ユーザーまたはロール) を作成する必要があります。次に、Amazon Keyspaces の適切な権限を付与するポリシーを、そのエンティティにアタッチする必要があります。アクセス許可が付与されたら、ユーザーまたはアプリケーション開発者に認証情報を提供します。これらの認証情報を使用して AWSにアクセスします。IAM ユーザー、グループ、ポリシー、アクセス許可の作成の詳細については、「IAM ユーザーガイド」の「IAM アイデンティティ」と「IAM のポリシーとアクセス許可」を参照してください。

自分の 以外のユーザーに Amazon Keyspaces 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 Keyspaces がこれらの機能をサポートしているかどうかを確認するには、「Amazon Keyspaces で IAM が機能する仕組み」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 Keyspaces のサービスリンクロールの使用

Amazon Keyspaces (Apache Cassandra 用) 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。サービスリンクロールは、Amazon Keyspaces 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 Keyspaces 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サポートする他のサービスの詳細については、AWS「IAM と連携するサービス」を参照し、「サービスにリンクされたロール」列で「はい」があるサービスを探します。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。

トピック

- Amazon Keyspaces アプリケーションの自動スケーリングにロールを使用
- Amazon Keyspaces のマルチリージョンレプリケーションでのロールの使用
- Amazon Keyspaces CDC ストリームのロールの使用

Amazon Keyspaces アプリケーションの自動スケーリングにロールを使用

Amazon Keyspaces (Apache Cassandra 用) 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。サービスリンクロールは、Amazon Keyspaces 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 Keyspaces 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。

必要な許可を手動で追加する必要がないため、サービスリンクロールは Amazon Keyspaces のセットアップを容易にします。サービスリンクロールの許可は Amazon Keyspaces が定義し、特に定義されない限り、Amazon Keyspaces のみがそのロールを引き受けることができます。定義されるアクセス許可には、信頼ポリシーと許可ポリシーが含まれており、その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。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これは、リソースにアクセスするための許可を不用意に削除できないため、Amazon Keyspaces リソースを保護できます。

Amazon Keyspaces のサービスリンクロール権限

Amazon Keyspaces では、AWSServiceRoleForApplicationAutoScaling_CassandraTable という名前のサービスリンクロールを使用して、ユーザーに代わってアプリケーションオートスケーリングにより Amazon Keyspaces と Amazon CloudWatch が呼び出されるようにします。

AWSServiceRoleForApplicationAutoScaling_CassandraTable サービスリンクロールは、以下のサービスを信頼してロールを引き受けます。

- `cassandra.application-autoscaling.amazonaws.com`

このロール権限ポリシーは、アプリケーションオートスケーリングにより、指定された Amazon Keyspaces リソースで以下のアクションが実行されるようにします。

- アクション: `cassandra:Select`。対象リソース: `arn:*:cassandra:*:*:/keyspace/system/table/*`
- アクション: リソース `arn:*:cassandra:*:*:/keyspace/system_schema/table/*` での `cassandra:Select`
- アクション: リソース `arn:*:cassandra:*:*:/keyspace/system_schema_mcs/table/*` での `cassandra:Select`
- アクション: リソース `arn:*:cassandra:*:*:"*"` での `cassandra:Alter`

Amazon Keyspaces 向けのサービスリンクロールの作成

Amazon Keyspaces のオートスケーリング用のサービスリンクロールについては、手動で作成する必要はありません。、CQL AWS Management Console、、 AWS CLIまたは AWS API を使用してテーブルで Amazon Keyspaces 自動スケーリングを有効にすると、Application Auto Scaling によってサービスにリンクされたロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。テーブルで Amazon Keyspaces の自動スケーリングを有効にすると、Application Auto Scaling が再びサービスリンクロールを自動作成します。

> **Important**
>
> このサービスリンク役割はこの役割でサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。詳細については、「新しいロールが AWS アカウント」を参照してください。

Amazon Keyspaces のサービスリンクロールの編集

Amazon Keyspaces では、サービスリンクロール AWSServiceRoleForApplicationAutoScaling_CassandraTable を編集できません。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 Keyspaces のサービスリンクロールの削除

サービスリンクロールを必要とする機能やサービスが不要になった場合は、ロールを削除することをお勧めします。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またはメンテナンスされ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手動で削除する AWS リージョン には、まずアカウント内のすべてのテーブルで自動スケーリングを無効にする必要があります。Amazon Keyspaces テーブルのオートスケーリングを無効にするには、「the section called “テーブルの Amazon Keyspaces 自動スケーリングを無効にする”」を参照してください。

> **Note**
>
> リソースの変更を試みた時点で、Amazon Keyspaces のオートスケーリングでロールが使用されていると、登録解除が失敗する可能性があります。その場合は、数分待ってからオペレーションを再試行してください。

IAM を使用してサービスリンクロールを手動で削除するには

IAM コンソール、 AWS CLI、または AWS API を使用して、AWSServiceRoleForApplicationAutoScaling_CassandraTable サービスにリンクされたロールを削除します。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削除」を参照してください。

> **Note**
>
> Amazon Keyspaces のオートスケーリングで使用されたサービスリンクロールを削除するには、まずアカウント内のすべてのテーブルでオートスケーリングを無効にする必要があります。

Amazon Keyspaces サービスにリンクされたロールのサポートされているリージョン

Amazon Keyspaces は、サービスが利用可能な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をサポートします。詳細については、「Service endpoints for Amazon Keyspaces」(Amazon Keyspaces のサービスエンドポイント) を参照してください。

Amazon Keyspaces のマルチリージョンレプリケーションでのロールの使用

Amazon Keyspaces (Apache Cassandra 用) 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。サービスリンクロールは、Amazon Keyspaces 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 Keyspaces 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。

必要な許可を手動で追加する必要がないため、サービスリンクロールは Amazon Keyspaces のセットアップを容易にします。サービスリンクロールの許可は Amazon Keyspaces が定義し、特に定義されない限り、Amazon Keyspaces のみがそのロールを引き受けることができます。定義されるアクセス許可には、信頼ポリシーと許可ポリシーが含まれており、その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。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これは、リソースにアクセスするための許可を不用意に削除できないため、Amazon Keyspaces リソースを保護できます。

Amazon Keyspaces のサービスリンクロール権限

Amazon Keyspaces は、AWSServiceRoleForAmazonKeyspacesReplication 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 Keyspaces がユーザーに代わってキースペース AWS リージョン に新しい を追加し、テーブルとそのすべてのデータと設定を新しいリージョンにレプリケートできるようにします。このロールでは、Amazon Keyspaces がユーザーに代わってすべてのリージョンのテーブルへの書き込みをレプリケートすることもできます。

AWSServiceRoleForAmazonKeyspacesReplication サービスにリンクされたロールは、以下のサービスを信頼してロールを引き受けます。

- `replication.cassandra.amazonaws.com`

KeyspacesReplicationServiceRolePolicy という名前のロールのアクセス許可ポリシーは、以下のアクションを実行することを Amazon Keyspaces に許可します。

- アクション: `cassandra:Select`
- アクション: `cassandra:SelectMultiRegionResource`
- アクション: `cassandra:Modify`
- アクション: `cassandra:ModifyMultiRegionResource`

- アクション: `cassandra:AlterMultiRegionResource`
- アクション: `application-autoscaling:RegisterScalableTarget` – Amazon Keyspaces は、Auto Scaling が有効になっているプロビジョニングモードで単一のリージョンテーブルにレプリカを追加するときに、アプリケーションの Auto Scaling アクセス許可を使用します。
- アクション: `application-autoscaling:DeregisterScalableTarget`
- アクション: `application-autoscaling:DescribeScalableTargets`
- アクション: `application-autoscaling:PutScalingPolicy`
- アクション: `application-autoscaling:DescribeScalingPolicies`
- アクション: `cassandra:Alter`
- アクション: `cloudwatch:DeleteAlarms`
- アクション: `cloudwatch:DescribeAlarms`
- アクション: `cloudwatch:PutMetricAlarm`

Amazon Keyspaces esのサービスにリンクされたロール
AWSServiceRoleForAmazonKeyspacesReplication は、ポリシーで指定された Amazon リソースネーム (ARN)「arn: *」に対するアクセス権「アクション:」を提供しますが、Amazon Keyspaces はアカウントの ARN を提供します。

サービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesReplication を作成するアクセス許可は、 `AmazonKeyspacesFullAccess`マネージドポリシーに含まれています。詳細については、「the section called “AmazonKeyspacesFullAccess”」を参照してください。

ユーザー、グループ、またはロールにサービスリンクロールの作成、編集、または削除を許可するには、アクセス許可を設定する必要があります。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役割のアクセス許可」を参照してください。

Amazon Keyspaces 向けのサービスリンクロールの作成

サービスにリンクされたロールは手動で作成できません。 AWS Management Console、、 AWS CLI または AWS API でマルチリージョンキースペースを作成すると、Amazon Keyspaces によってサービスにリンクされたロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。マルチリージョンキースペース を作成すると、Amazon Keyspaces 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 Keyspaces のサービスリンクロールの編集

Amazon Keyspaces では、AWSServiceRoleForAmazonKeyspacesReplication というサービスにリンクされたロールを編集できません。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 Keyspaces のサービスリンクロールの削除

サービスリンクロールを必要とする機能やサービスが不要になった場合は、ロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、サービスにリンクされたロールを手動で削除する AWS リージョンには、まずアカウント内のすべてのマルチリージョンキースペースを削除する必要があります。

サービスリンク役割のクリーンアップ

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、最初に、そのロールで使用されているマルチリージョンキースペースとテーブルをすべて削除する必要があります。

> **Note**
>
> リソースを削除しようとしているときに Amazon Keyspaces サービスがロールを使用している場合は、削除が失敗する可能性があります。失敗した場合は数分待ってから操作を再試行してください。

AWSServiceRoleForAmazonKeyspacesReplication (コンソール) が使用している Amazon Keyspaces リソースを削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home で Amazon Keyspaces コンソールを開きます。
2. 左側のパネルから [Keyspaces] を選択します。
3. リストからマルチリージョンキースペースをすべて選択します。
4. [削除] を選択して削除を確認し、[キースペースを削除] を選択します。

次のいずれかの方法で、マルチリージョンキースペースをプログラムで削除することもできます。

- Cassandra クエリ言語 (CQL) ??? ステートメント

- CLI の delete-keyspace AWS オペレーション。
- Amazon Keyspaces API の DeleteKeyspace 操作。

サービスリンク役割の手動による削除

IAM コンソール、 AWS CLI、または AWS API を使用して、AWSServiceRoleForAmazonKeyspacesReplication サービスにリンクされたロールを削除します。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 Keyspaces サービスにリンクされたロールのサポートされているリージョン

Amazon Keyspaces 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をサポートしているわけではありません。AWSServiceRoleForAmazonKeyspacesReplication ロールは、以下のリージョンで使用できます。

リージョン名	リージョン識別子	Amazon Keyspaces でのサポート
米国東部 (バージニア北部)	us-east-1	はい
米国東部 (オハイオ)	us-east-2	はい
米国西部 (北カリフォルニア)	us-west-1	はい
米国西部 (オレゴン)	us-west-2	はい
アジアパシフィック (ムンバイ)	ap-south-1	はい
アジアパシフィック (大阪)	ap-northeast-3	はい
アジアパシフィック (ソウル)	ap-northeast-2	はい
アジアパシフィック (シンガポール)	ap-southeast-1	はい
アジアパシフィック (シドニー)	ap-southeast-2	はい
アジアパシフィック (東京)	ap-northeast-1	はい

リージョン名	リージョン識別子	Amazon Keyspaces でのサポート
カナダ (中部)	ca-central-1	はい
欧州 (フランクフルト)	eu-central-1	はい
欧州 (アイルランド)	eu-west-1	はい
欧州 (ロンドン)	eu-west-2	はい
欧州 (パリ)	eu-west-3	はい
アフリカ (ケープタウン)	af-south-1	はい
南米 (サンパウロ)	sa-east-1	はい
AWS GovCloud (米国東部)	us-gov-east-1	いいえ
AWS GovCloud (米国西部)	us-gov-west-1	いいえ

Amazon Keyspaces CDC ストリームのロールの使用

Amazon Keyspaces (Apache Cassandra 用) 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。サービスリンクロールは、Amazon Keyspaces 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 Keyspaces 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。

必要な許可を手動で追加する必要がないため、サービスリンクロールは Amazon Keyspaces のセットアップを容易にします。サービスリンクロールの許可は Amazon Keyspaces が定義し、特に定義されない限り、Amazon Keyspaces のみがそのロールを引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その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。

サービスにリンクされたロールは削除できません。

Amazon Keyspaces のサービスリンクロール権限

Amazon Keyspaces は、AWSServiceRoleForAmazonKeyspacesCDC 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 Keyspaces CDC ストリームがユーザーに代わって CloudWatch メトリクスをアカウントに発行できるようにします。

AWSServiceRoleForAmazonKeyspacesCDC サービスにリンクされたロールは、次のサービスを信頼してロールを引き受けます。

- `cassandra-streams.amazonaws.com`

KeyspacesCDCServiceRolePolicy という名前のロールアクセス許可ポリシーにより、Amazon Keyspaces は CloudWatch 名前空間 のリソースに対して次のアクションを実行できます`AWS/Cassandra`。

- アクション: `cloudwatch:PutMetricData`。対象リソース: `*`

  AWSServiceRoleForAmazonKeyspacesCDC には、次の条件に一致するすべてのリソースに対するアクセス許可 Action: cloudwatch:PutMetricData が用意されています。
  `"cloudwatch:namespace": "AWS/Cassandra"`

KeyspacesCDCServiceRolePolicy の詳細については、「」を参照してくださいthe section called “KeyspacesCDCServiceRolePolicy”。

サービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesCDC を自動的に作成するテーブルの CDC ストリームを有効にするには、IAM プリンシパルに次のアクセス許可が必要です。

```
{
    "Sid": "KeyspacesCDCServiceLinkedRole",
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/cassandra-streams.amazonaws.com/
AWSServiceRoleForAmazonKeyspacesCDC",
    "Condition": {
    "StringLike": {
        "iam:AWSServiceName": "cassandra-streams.amazonaws.com"
    }
}
```

サービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesCDC を作成するアクセス許可は、 AmazonKeyspacesFullAccessマネージドポリシーに含まれています。詳細については、「the section called “AmazonKeyspacesFullAccess”」を参照してください。

Amazon Keyspaces 向けのサービスリンクロールの作成

Amazon Keyspaces CDC ストリームのサービスにリンクされたロールを手動で作成する必要はありません。、CQL AWS Management Console、、 AWS CLIまたは AWS API を使用してテーブルで Amazon Keyspaces CDC ストリームを有効にすると、Amazon Keyspaces によってサービスにリンクされたロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。テーブルの Amazon Keyspaces CDC ストリームを有効にすると、Amazon Keyspaces 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 Keyspaces のサービスリンクロールの編集

Amazon Keyspaces では、AWSServiceRoleForAmazonKeyspacesCDC 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 Keyspaces サービスにリンクされたロールのサポートされているリージョン

Amazon Keyspaces は、サービスが利用可能な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をサポートします。詳細については、「AWS リージョンとエンドポイント」を参照してください。

Amazon Keyspaces (Apache Cassandra 向け) のコンプライアンス検証

サードパーティーの監査者は、複数の コンプライアンスプログラムの一環として、Amazon Keyspaces (Apache Cassandra 向け) のセキュリティと AWS コンプライアンスを評価します。具体的には次のとおりです。

- ISO/IEC 27001:2013、27017:2015、27018:2019、および ISO/IEC 9001:2015。詳細については、「AWS ISO and CSA STAR certifications and services (ISO および CSA STAR に関する認定およびサービス)」 を参照してください。
- System and Organization Controls (SOC)

- Payment Card Industry (PCI)
- Federal Risk and Authorization Management Program (FedRAMP) High
- Health Insurance Portability and Accountability Act (HIPAA)

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、AWS のサービス「コンプライアンスプログラムによる対象範囲内」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「 Compliance ProgramsAssurance」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供します。

- セキュリティのコンプライアンスとガバナンス – これらのソリューション実装ガイドでは、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の機能をデプロイする手順を示します。
- HIPAA 対応サービスのリファレンス – HIPAA 対応サービスの一覧が提供されています。すべての AWS のサービス が HIPAA の対象となるわけではありません。
- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドは、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールを保護し、そのガイダンスに AWS のサービス マッピングするためのベストプラクティスをまとめたものです。
- 「 デベロッパーガイド」の「ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が内部プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、Security Hub のコントロールリファレンスを参照してください。

- Amazon GuardDuty – 不審なアクティビティや悪意のあるアクティビティがないか環境をモニタリングすることで AWS アカウント、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

Amazon Keyspaces の耐障害性と災害対策

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、冗長性の高い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 Keyspaces は、耐久性と高可用性 AWS リージョン を実現するために、同じ 内の複数の AWS アベイラビリティーゾーンでデータを 3 回自動的にレプリケートします。

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 Keyspaces 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いくつかの機能が用意されています。

マルチリージョンレプリケーション

Amazon Keyspaces は、データまたはアプリケーションを地理的距離を超えてレプリケートする必要がある場合に、マルチリージョンレプリケーションを提供します。Amazon Keyspaces テーブルは、選択したさまざまな にレプリケート AWS リージョン できます。詳細については、「the section called “マルチリージョンレプリケーション”」を参照してください。

ポイントインタイムリカバリ (PITR)

PITR を使用すると、テーブルデータを継続的にバックアップできるため、Amazon Keyspaces テーブルの偶発的な書き込みや削除などのオペレーションを防止できます。詳細については、

「Point-in-Time Recovery for Amazon Keyspaces (Amazon Keyspaces 向けポイントインタイムリカバリ)」を参照してください。

Amazon Keyspaces 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 Keyspaces (Apache Cassandra 向け) 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 AWS 環境を設計するには、「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 Keyspaces にアクセスします。クライアントは以下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。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。またはAWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Amazon Keyspaces では、2 つのクライアントリクエスト認証方法がサポートされています。1 つ目の方法では、サービス固有の認証情報を使用します。これは、特定の IAM ユーザーに対して生成されたパスワードベースの認証情報です。IAM コンソール、、または AWS API を使用して AWS CLI、パスワードを作成および管理できます。詳細については、「Using IAM with Amazon Keyspaces (Amazon Keyspaces での IAM の使用)」 を参照してください。

2 つ目の方法は、Cassandra 用のオープンソース DataStax Java ドライバーに対して認証プラグインを使用します。このプラグインでは、IAM ユーザー、ロール、およびフェデレーテッドアイデンティティにより、AWS 署名バージョン 4 署名プロセス (SigV4) を使用して、Amazon Keyspaces (Apache Cassandra 向け) API リクエストに認証情報を追加することができます。詳細については、「the section called “ AWS 認証用の IAM 認証情報を作成する”」を参照してください。

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が、Amazon Keyspaces はリソースベースのアクセスポリシーをサポートしており、ソース IP アドレスに基づ

く制限を含めることができます。Amazon Keyspaces 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 VPCs からのアクセスを制御することもできます。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 Keyspaces リソースへの AWS ネットワークアクセスが分離されます。

インターフェイス VPC エンドポイントを使用して、Amazon VPC と Amazon Keyspaces 間のトラフィックが Amazon ネットワークから離れないようにすることができます。インターフェイス VPC エンドポイントは AWS PrivateLink、Amazon VPC AWS 内のプライベート IPs を持つ Elastic Network Interface を使用して AWS サービス間のプライベート通信を可能にするテクノロジーを利用しています。詳細については、「the section called “インターフェイス VPC エンドポイントの使用”」を参照してください。

インターフェイス VPC エンドポイントと Amazon Keyspaces の使用

インターフェイス VPC エンドポイントでは、Amazon VPC で実行されている仮想プライベートクラウド (VPC) と Amazon Keyspaces 間のプライベート通信ができます。インターフェイス VPC エンドポイントは AWS PrivateLink、VPC と AWS AWS サービス間のVPCs。

AWS PrivateLink は、ネットワークトラフィックが Amazon ネットワークを離れないように、VPC 内のプライベート IP アドレスを持つ Elastic Network Interface を使用してこれを可能にします。インターフェイス VPC エンドポイントは、インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を必要としません。詳細については、「Amazon Virtual Private CloudAmazon」(仮想プライベートクラウド) と「Interface VPC endpoints (AWS PrivateLink)」(インターフェイス VPC エンドポイント ()) を参照してください。

トピック

- Amazon Keyspaces 用インターフェイス VPC エンドポイントの使用
- インターフェイス VPC エンドポイント情報を含む system.peers テーブルエントリの入力
- Amazon Keyspaces のインターフェイス VPC エンドポイントへのアクセスの制御
- 可用性
- VPC エンドポイントポリシーと Amazon Keyspaces ポイントインタイムリカバリ (PITR)
- よくあるエラーおよび警告

Amazon Keyspaces 用インターフェイス VPC エンドポイントの使用

Amazon Keyspaces と Amazon VPC リソース間のトラフィックがインターフェイス VPC エンドポイントを経由して流れるように、インターフェイス VPC エンドポイントを作成することができま

す。開始するには、インターフェイスエンドポイントの作成手順を実行します。次に、前のステップで作成したエンドポイントに関連付けられたセキュリティグループを編集し、ポート 9142 のインバウンドルールを設定します。詳細については、「Adding, removing, and updating rules」(ルールの追加、削除、および更新) を参照してください。

VPC エンドポイントを介して Amazon Keyspaces への接続を設定するステップバイステップのチュートリアルについては、「the section called “VPC エンドポイントによる接続”」を参照してください。VPC AWS アカウント 内の異なる のアプリケーションから分離された Amazon Keyspaces リソースのクロスアカウントアクセスを設定する方法については、「」を参照してくださいthe section called “クロスアカウントアクセスの設定”。

インターフェイス VPC エンドポイント情報を含む **system.peers** テーブルエントリの入力

Apache Cassandra ドライバーにより、`system.peers` テーブルを使用してクラスターに関するノード情報のクエリが行われます。Cassandra ドライバーでは、ノード情報を使用して接続のロードバランスと再試行オペレーションが行われます。Amazon Keyspaces では、パブリックエンドポイント経由のクライアント接続のために、9 つのエントリが `system.peers` テーブルに自動で入力されます。

同様の機能を備えたインターフェイス VPC エンドポイントを経由するクライアント接続を実現するために、Amazon Keyspaces では、VPC エンドポイントを利用できる各アベイラビリティーゾーンのエントリが含まれているアカウント内で、`system.peers` テーブルへの入力が行われます。Amazon Keyspaces では、使用可能な VPC エンドポイントを探し出して `system.peers` テーブルに保存する場合、Amazon Keyspaces への接続に使用する IAM エンティティ対して、エンドポイントとネットワークインターフェイスの情報について VPC をクエリするためのアクセス許可を付与する必要があります。

> **Important**
>
> `system.peers` テーブルに使用可能なインターフェイス VPC エンドポイントを入力することで、ロードバランシングが改善され、読み取り/書き込みスループットが向上します。全てのクライアントがインターフェイス VPC エンドポイントを使用して Amazon Keyspaces にアクセスすることが推奨され、Apache Spark に必要です。

Amazon Keyspaces への接続に使用する IAM エンティティに、必要なインターフェイス VPC エンドポイント情報を検索するための許可を付与するには、既存の IAM ロールまたはユーザーポリシーを更新するか、または、次の例に示すように新しい IAM ポリシーを作成します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListVPCEndpoints",
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcEndpoints"
         ],
         "Resource":"*"
      }
   ]
}
```

> **Note**
>
> 管理ポリシー AmazonKeyspacesReadOnlyAccess_v2 および AmazonKeyspacesFullAccess には、Amazon Keyspaces が Amazon EC2 インスタンスにアクセスして、使用可能なインターフェイス VPC エンドポイントに関する情報を読み取れるようにするために必要なアクセス権限が含まれています。

ポリシーが正しく設定されていることを確認するには、system.peers テーブルをクエリしてネットワーク情報を表示します。system.peers テーブルが空である場合は、ポリシーが正常に設定されなかったこと、または DescribeNetworkInterfaces と DescribeVPCEndpoints API アクションのリクエストレートクォータを超過していることを示している可能性があります。DescribeVPCEndpoints は Describe* カテゴリーになって不変アクションとみなされます。DescribeNetworkInterfaces はフィルター処理とページ分割が施されていない不変アクションのサブセットになり、別のクォータが適用されます。詳細については、「Amazon EC2 API Reference」(Amazon EC2 API リファレンス) の「Request token bucket sizes and refill rates」(リクエストトークンのバケットサイズと補充レート) を参照してください。

空のテーブルが表示された場合は、数分待ってから、リクエストレートクォータ問題の除外を再試行してください。VPC エンドポイントが正しく設定されていることを確認するには、「the section called “VPC エンドポイント接続エラー”」を参照してください。クエリによりテーブルから結果が返された場合は、ポリシーが正しく設定されています。

Amazon Keyspaces のインターフェイス VPC エンドポイントへのアクセスの制御

VPC エンドポイントポリシーによって、次の 2 つの方法でリソースへのアクセスを制御できます。

- IAM ポリシー – 特定の VPC エンドポイントを経由する Amazon Keyspaces へのアクセスを許可されているリクエスト、ユーザー、またはグループを管理できます。これを実行するには、IAM ユーザー、グループ、またはロールにアタッチされているポリシー内で条件キーを使用します。
- VPC ポリシー — ポリシーをアタッチすることで、Amazon Keyspaces リソースへのアクセス権が付与されている VPC エンドポイントを制御できます。特定のキースペースまたはテーブルへのアクセスを制限して、特定の VPC エンドポイントを通過するトラフィックのみを許可するには、リソースアクセスを制限する既存のIAMポリシーを編集し、その VPC エンドポイントを追加します。

以下は、Amazon Keyspaces リソースにアクセスするためのエンドポイントポリシーです。

- IAM ポリシー例: トラフィックが指定 VPCエンドポイントからのものでない限り特定の Amazon Keyspaces テーブルへのすべてのアクセスを制限する - このサンプルポリシーは IAM ユーザー、ロール、またはグループにアタッチできます。これにより、受信トラフィックが指定の VPC エンドポイントから発信されない限り、指定の Amazon Keyspaces テーブルへのアクセスが制限されます。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "UserOrRolePolicyToDenyAccess",
         "Action": "cassandra:*",
         "Effect": "Deny",
         "Resource": [
                         "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
mykeyspace/table/mytable",
                         "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
           ],
         "Condition": { "StringNotEquals" : { "aws:sourceVpce": "vpce-abc123" } }
      }
   ]
}
```

> **Note**
>
> 特定のテーブルへのアクセスを制限するには、システムテーブルへのアクセスも含める必要があります。システムテーブルは読み取り専用です。

- VPC ポリシーの例: 読み取り専用アクセス - このサンプルポリシーは VPC エンドポイントにアタッチできます。詳細については、「Controlling access to Amazon VPC resources」(Amazon VPC のリソースに対するアクセスの制御) を参照してください。これにより、このポリシーがアタッチされている VPC エンドポイントを経由する Amazon Keyspaces リソースへの読み取り専用アクセスに、アクションが制限されます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "cassandra:Select"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- VPC ポリシーの例: 特定の Amazon Keyspaces テーブルへのアクセスを制限する - このサンプルポリシーは VPC エンドポイントにアタッチできます。これにより、このポリシーがアタッチされている VPC エンドポイントを経由する特定のデータストリームへのアクセスが制限されます。

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Sid": "RestrictAccessToTable",
            "Principal": "*",
            "Action": "cassandra:*",
            "Effect": "Allow",
            "Resource": [
```

```
                        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
mykeyspace/table/mytable",
                        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
            ]
        }
    ]
}
```

> **Note**
>
> 特定のテーブルへのアクセスを制限するには、システムテーブルへのアクセスも含める必要があります。システムテーブルは読み取り専用です。

可用性

Amazon Keyspaces は、 AWS リージョン サービスが利用可能なすべての でインターフェイス VPC エンドポイントの使用をサポートしています。詳細については、「???」を参照してください。

VPC エンドポイントポリシーと Amazon Keyspaces ポイントインタイムリカバリ (PITR)

IAM ポリシーを条件キーとともに使用して受信トラフィックを制限している場合は、テーブルの復元オペレーションが失敗することがあります。例えば、aws:SourceVpce 条件キーを使用してソーストラフィックを特定のエンドポイントに制限している場合、テーブルの復元オペレーションは失敗します。プリンシパルの代わりに Amazon Keyspaces によって復元オペレーションが実行されるようにするには、IAM ポリシーに aws:ViaAWSService 条件キーを追加する必要があります。aws:ViaAWSService 条件キーは、任意の AWS サービスがプリンシパルの認証情報を使用してリクエストを行うときにアクセスを許可します。詳細については、「IAM ユーザーガイド」の「IAM JSON policy elements: Condition key」(IAM JSON ポリシー要素: 条件キー) を参照してください。以下のポリシーはこの例です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"CassandraAccessForVPCE",
         "Effect":"Allow",
         "Action":"cassandra:*",
         "Resource":"*",
```

```
         "Condition":{
            "Bool":{
               "aws:ViaAWSService":"false"
            },
            "StringEquals":{
               "aws:SourceVpce":[
                  "vpce-12345678901234567"
               ]
            }
         }
      },
      {
         "Sid":"CassandraAccessForAwsService",
         "Effect":"Allow",
         "Action":"cassandra:*",
         "Resource":"*",
         "Condition":{
            "Bool":{
               "aws:ViaAWSService":"true"
            }
         }
      }
   ]
}
```

よくあるエラーおよび警告

Amazon Virtual Private Cloud を使用していて Amazon Keyspaces に接続すると、次の警告が表示される場合があります。

```
Control node cassandra.us-east-1.amazonaws.com/1.111.111.111:9142 has an entry
 for itself in system.peers: this entry will be ignored. This is likely due to a
 misconfiguration;
please verify your rpc_address configuration in cassandra.yaml on all nodes in your
 cluster.
```

この警告は、接続している Amazon VPC エンドポイントを含め、Amazon Keyspaces が表示権限を持つすべての Amazon VPC エンドポイントのエントリが `system.peers` テーブルに含まれているために発生します。この警告を無視しても問題ありません。

その他のエラーについては、「the section called “VPC エンドポイント接続エラー”」を参照してください。

インターフェイス VPC エンドポイントでの Amazon Keyspaces CDC ストリームの使用

インターフェイス VPC エンドポイントでは、Amazon VPC で実行されている仮想プライベートクラウド (VPC) と Amazon Keyspaces 間のプライベート通信ができます。インターフェイス VPC エンドポイントは AWS PrivateLink、VPC と AWS AWS サービス間のVPCs。

AWS PrivateLink は、ネットワークトラフィックが Amazon ネットワークを離れないように、VPC 内のプライベート IP アドレスを持つ Elastic Network Interface を使用してこれを可能にします。インターフェイス VPC エンドポイントは、インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を必要としません。詳細については、「Amazon Virtual Private CloudAmazon」(仮想プライベートクラウド) と「Interface VPC endpoints (AWS PrivateLink)」(インターフェイス VPC エンドポイント ()) を参照してください。

トピック

- Amazon Keyspaces CDC ストリームのインターフェイス VPC エンドポイントの使用
- Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイント
- Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイントを作成する
- Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイントを更新する
- Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイントを使用してストリームを一覧表示する
- Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイントのポリシーを作成する

Amazon Keyspaces CDC ストリームのインターフェイス VPC エンドポイントの使用

インターフェイス VPC エンドポイントを使用すると、Amazon Keyspaces CDC ストリームと Amazon VPC リソース間のトラフィックがインターフェイス VPC エンドポイントを経由し始めることができます。VPC エンドポイントポリシーを使用して、CDC ストリームへのアクセスを制限できます。

Amazon Keyspaces CDC ストリームの詳細については、「」を参照してくださいthe section called “CDC ストリームの使用”。

Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイント

インターフェイスエンドポイントを作成すると、Amazon Keyspaces CDC ストリームはストリームにリージョンとゾーンの 2 種類のエンドポイント固有の DNS 名を生成します。

リージョン別

リージョン DNS 名には、次の情報が含まれます。

- 一意の Amazon VPC エンドポイント ID
- サービス識別子
- AWS リージョン「��
- `vpce.amazonaws.com` サフィックス

ID が の Amazon VPC エンドポイントの場合`vpce-1a2b3c4d`、生成された DNS 名は次の例のようになります。 `vpce-1a2b3c4d-5e6f.cassandra-streams.us-east-1.vpce.amazonaws.com`

ゾーン別

ゾーン DNS 名には、リージョン DNS 名の情報に加えて、アベイラビリティーゾーンが含まれます。ID の Amazon VPC エンドポイントに対して生成された DNS 名`vpce-1a2b3c4d`は、次の例のようになります。 にアベイラビリティーゾーンが含まれる AWS リージョン ようになりました。 `vpce-1a2b3c4d-5e6f-us-east-1a.cassandra-streams.us-east-1.vpce.amazonaws.com`

このオプションは、アーキテクチャがアベイラビリティーゾーンを分離する場合に使用できます。例えば、障害を隔離し、リージョン間のデータ転送コストを削減するために使用できます。

> **Note**
>
> 最適な信頼性を実現するには、最低 3 つのアベイラビリティーゾーンにサービスをデプロイすることをお勧めします。

Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイントを作成する

AWS CLI または AWS SDK を使用して、Amazon Keyspaces CDC Streams インターフェイスエンドポイントを介して Amazon Keyspaces CDC Streams API オペレーションにアクセスできます。使

用可能なすべての API オペレーションの完全なリストについては、「Amazon Keyspaces Streams API リファレンス」を参照してください。

VPC エンドポイントの作成方法の詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントの作成」を参照してください。

VPC エンドポイントを作成するには、次の例の 構文を使用できます。

```
aws ec2 create-vpc-endpoint \
  --region us-east-1 \
  --service-name api.aws.us-east-1.cassandra-streams \
  --vpc-id client-vpc-id \
  --subnet-ids client-subnet-id \
  --vpc-endpoint-type Interface \
  --security-group-ids client-sg-id
```

Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイントを更新する

VPC エンドポイントを更新するには、次の例の 構文を使用できます。

```
aws ec2 modify-vpc-endpoint \
  --region us-east-1 \
  --vpc-endpoint-id client-vpc-id \
  --policy-document policy-document \ #example optional parameter
  --add-security-group-ids security-group-ids \ #example optional parameter
```

Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイントを使用してストリームを一覧表示する

VPC エンドポイントを使用しているストリームを一覧表示するには、次の例の 構文を使用できます。VPC エンドポイント ID のリージョンと DNS 名を自分の情報に置き換えてください。

```
aws keyspacesstreams \
  --endpoint https://vpce-1a2b3c4d-5e6f.cassandra-streams.us-east-1.vpce.amazonaws.com
 \
  --region us-east-1 \
  list-streams
```

Amazon Keyspaces CDC ストリームインターフェイス VPC エンドポイントのポリシーを作成する

Amazon Keyspaces CDC ストリームへのアクセスを制御するエンドポイントポリシーを Amazon 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 AWS Identity and Access Management (IAM) 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

特定の Amazon Keyspaces CDC ストリームへのアクセスを制限して Amazon VPC 内の特定の AWS サービスのみを許可するには、次の例を使用できます。

次のストリームポリシーは、 アクションcassandra:GetStreamと、アカウント /keyspace/mykeyspace/table/mytable/に属するリソースに2025-02-20T11:22:33.444アタッチされた指定されたストリームcassandra:GetRecordsの任意の IAM プリンシパルへのアクセスを許可します123456788901。このエンドポイントポリシーを使用するには、リージョン、アカウント ID、リソースをストリームラベルに置き換えてください。

```
{
"Version": "2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-stream-only",
      "Principal": "*",
      "Action": [
        "cassandra:GetStream",
        "cassandra:GetRecords"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:cassandra:us-east-1:123456788901:/keyspace/mykeyspace/
table/mytable/stream/2025-02-20T11:22:33.444"]
    }
  ]
}
```

> **Note**
>
> Amazon Keyspaces は、CDC ストリームのゲートウェイエンドポイントをサポートしていません。

Amazon Keyspaces の設定と脆弱性の分析

AWS は、ゲストオペレーティングシステム (OS) やデータベースのパッチ適用、ファイアウォール設定、ディザスタリカバリなどの基本的なセキュリティタスクを処理します。これらの手順は適切な第三者によって確認され、証明されています。詳細については、以下のリソース を参照してください。

- 責任共有モデル、、
- アマゾン ウェブ サービス: セキュリティプロセスの概要 (ホワイトペーパー)

Amazon Keyspaces のセキュリティベストプラクティス

Amazon Keyspaces (Apache Cassandra 向け) には、独自のセキュリティポリシーを策定および実装する際に考慮すべきさまざまなセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションを説明するものではありません。これらのベストプラクティスはお客様の環境に適切ではないか、十分ではない場合があるため、これらは指示ではなく、有用な考慮事項と見なしてください。

トピック

- Amazon Keyspaces の予防的セキュリティベストプラクティス
- Amazon Keyspaces の発見的セキュリティベストプラクティス

Amazon Keyspaces の予防的セキュリティベストプラクティス

セキュリティに関する以下のベストプラクティスは、Amazon Keyspaces でのセキュリティインシデントの予測と予防に役立ちます。

保管データ暗号化を使用する

Amazon Keyspaces では、テーブルに保管されているすべてのユーザーデータに対して、AWS Key Management Service (AWS KMS) に保存されている暗号化キーを使用して保管データ暗号化

が行われます。この機能は、基になるストレージへの不正アクセスからデータを保護することによって、データ保護の追加レイヤーを提供します。

デフォルトでは、Amazon Keyspaces はすべてのテーブルを暗号化 AWS 所有のキー するために を使用します。このキーは、存在しなければ自動的に作成されます。サービスデフォルトキーは無効にできません。

代わりに、カスタマーマネージドキーを保管データ暗号化に使用できます。詳細については、「Amazon Keyspaces Encryption at Rest」(Amazon Keyspaces の保管データ暗号化) を参照してください。

IAM ロールを使用して Amazon Keyspaces へのアクセスを認証する

ユーザー、アプリケーション、およびその他の AWS サービスが Amazon Keyspaces にアクセスするには、 AWS API リクエストに有効な AWS 認証情報を含める必要があります。 AWS 認証情報をアプリケーションまたは EC2 インスタンスに直接保存しないでください。自動更新されない長期認証情報条件のため、漏洩すると業務に深刻な悪影響が及ぶ可能性があります。IAM ロールでは、 AWS サービスおよびリソースにアクセスするために使用できる一時的なアクセスキーを有効にすることができます。

詳細については、「IAM ロール」を参照してください。

IAM ポリシーを使用して Amazon Keyspaces ベースの認可を行う

許可を付与する場合、許可を取得するユーザー、取得する許可の対象となる Amazon Keyspaces、およびそれらのリソースに対して許可される特定のアクションを決定します。最小特権の実装は、セキュリティリスクはもちろん、エラーや悪意ある行動によってもたらされる可能性のある影響を減らす上での鍵となります。

IAM アイデンティティ (ユーザー、グループ、ロール) にアクセス権限ポリシーをアタッチし、Amazon Keyspaces リソースでオペレーションを実行する許可を付与します。

これを行うには、次を使用します。

- AWS 管理 (事前定義) ポリシー
- カスタマー管理ポリシー

詳細に設定されたアクセスコントロールのための IAM ポリシー条件を使用する

Amazon Keyspaces でアクセス許可を付与するときは、アクセス権限ポリシーを有効にする方法を決める条件を指定できます。最小特権の実装は、セキュリティリスクはもちろん、エラーや悪意ある行動によってもたらされる可能性のある影響を減らす上での鍵となります。

IAM ポリシーを使用して、アクセス許可を付与するときに条件を指定できます。例えば、次のオペレーションを実行できます。

- 特定のキースペースまたはテーブルに対する読み取り専用アクセスをユーザーに許可するために、アクセス許可を付与します。
- ユーザーのアイデンティティに基づいて、特定のテーブルへのユーザー書き込みアクセスを許可するために、アクセス許可を付与します。

詳細については、「Identity-Based Policy Examples (アイデンティティベースのポリシーの例)」を参照してください。

クライアント側の暗号化を考慮する

機密データや機密データを Amazon Keyspaces に保存する場合は、データを可能な限りオリジンの近くで暗号化して、ライフサイクル全体にわたってデータを保護することを検討してください。伝送中および保管時の機密データを暗号化することで、サードパーティーがお客様のプレーンテキストデータを使用することはできません。

Amazon Keyspaces の発見的セキュリティベストプラクティス

セキュリティに関する以下のベストプラクティスは、潜在的なセキュリティ上の弱点とインシデントの検出に役立つため、発見的とみなされています。

を使用して AWS Key Management Service （ AWS KMS) AWS KMS キーの使用状況をモニタリング AWS CloudTrail する

保管時の暗号化にカスタマーマネージド AWS KMS キーを使用している場合、このキーの使用が にログインされます AWS CloudTrail。CloudTrail は、アカウントで実行されたアクションをレコードすることで、ユーザーのアクティビティを可視化します。CloudTrail は、リクエストを行ったユーザー、使用されたサービス、実行されたアクション、アクションのパラメータ、AWS サービスによって返されたレスポンス要素など、各アクションに関する重要な情報を記録します。この情報は、 AWS リソースに加えられた変更を追跡し、運用上の問題をトラブルシューティングするのに役立ちます。CloudTrail を使用すると、社内ポリシーや規制スタンダードへのコンプライアンスが容易になります。

CloudTrail を使用して、キーの使用状況を監査できます。CloudTrail は、アカウントの AWS API コールおよび関連イベントの履歴を含むログファイルを作成します。これらのログファイルには、統合 AWS サービスを通じて行われたものに加えて、コンソール、 AWS SDKs、コマンドラインツールを使用して行われたすべての AWS KMS API リクエストが含まれます。これらのログファイルを使用して、 AWS KMS キーが使用された日時、リクエストされたオペレーション、

リクエスタの ID、リクエスト元の IP アドレスなどに関する情報を取得できます。詳細については、「AWS CloudTrailを使用した AWS Key Management Service API 呼び出しのログ記録」と「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail を使用して、Amazon Keyspaces データ定義言語 (DDL) オペレーションをモニタリングする

CloudTrail は、アカウントで実行されたアクションをレコードすることで、ユーザーのアクティビティを可視化します。CloudTrail は、リクエストを行ったユーザー、使用されたサービス、実行されたアクション、アクションのパラメータ、 AWS のサービスから返されたレスポンス要素など、各アクションに関する重要な情報をレコードします。この情報は、 AWS リソースに加えられた変更の追跡、およびオペレーション問題のトラブルシューティングに役立ちます。CloudTrail を使用すると、社内ポリシーや規制スタンダードへのコンプライアンスが容易になります。

Amazon Keyspaces DDL オペレーションはすべて CloudTrail のログに自動的に記録されます。DDL オペレーションでは、Amazon Keyspaces のキースペースとテーブルの作成と管理を行います。

Amazon Keyspaces 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。詳細については、「Logging Amazon Keyspaces operations by using AWS CloudTrail」(を使用した Amazon Keyspaces オペレーションの口グ記録) を参照してください。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、「AWS CloudTrail ユーザーガイド」の「Viewing events with CloudTrail event history」(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 Keyspaces 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 Simple Storage Service (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、証跡がすべての AWS リージョンに適用されます。証跡では、 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がログに記録され、指定した S3 バケットにログファイルが配信されます。さらに、CloudTrail ログで収集されたイベントデータをより詳細に分析し、それに基づいて行動するように、他の AWS サービスを設定できます。

識別とオートメーションのために Amazon Keyspaces リソースにタグを付ける

タグ形式で AWS リソースにメタデータを割り当てることができます。各タグは、カスタマー定義のキーと値 (オプション) で構成されるシンプルなラベルです。タグを使用すると、リソースの管理、検索、フィルターが容易になります。

タグ付けを行うと、グループ化されたコントロールを実装できます。タグには固有のタイプはありませんが、 用途、所有者、環境などの基準でリソースを分類できます。次に例をいくつか示します。

- アクセス — タグに基づいて Amazon Keyspaces リソースへのアクセスを制御するために使用されます。詳細については、「the section called “Amazon Keyspaces タグに基づいた認可”」を参照してください。
- セキュリティ — データ保護設定などの要件を決定するために使用されます。
- 機密性 — リソースでサポートされるデータ機密性レベルの識別子。
- 環境 — 開発、テスト、本番稼働用インフラストラクチャを区別するために使用されます。

詳細については、「AWS tagging strategies (タグ付け戦略)」と「Adding tags and labels to resources (リソースへのタグとラベルの追加)」を参照してください。

Amazon Keyspaces (Apache Cassandra 向け) の CQL 言語リファレンス

Amazon Keyspaces エンドポイントに接続した後は、Cassandra クエリ言語 (CQL) を使用してデータベースを操作します。CQL は多くの点で構造化クエリ言語 (SQL) に似ています。

- CQL 要素 – Amazon Keyspaces でサポートされている CQL の基本要素 (識別子、定数、用語、データ型など) を取り上げます。文字列型、数値型、コレクション型などの概念について説明します。
- データ定義言語 (DDL) – DDL ステートメントは、Amazon Keyspaces のキースペースやテーブルなどのデータ構造を管理するために使用されます。このセクションでは、キースペースとテーブルを作成、変更、削除するステートメントや、ポイントインタイムバックアップからテーブルを復元するステートメントを取り上げます。
- データ操作言語 (DML) – DML ステートメントは、テーブル内のデータを管理するために使用されます。このセクションでは、データの選択、挿入、更新、削除に関するステートメントを取り上げます。また、`IN` 演算子の使用、結果の順序付け、ページ分割など、高度なクエリ機能についても説明します。
- 組み込み関数 – Amazon Keyspaces は、さまざまな組み込みスカラー関数をサポートしています。これらの関数を CQL ステートメントで使用できます。このセクションでは、これらの関数について、使用例を含め、概要を紹介します。

このトピックを通じて、Amazon Keyspaces で CQL を効果的に使用するための具体的な構文、例、ベストプラクティスを紹介します。

トピック

- Amazon Keyspaces における Cassandra クエリ言語 (CQL) 要素
- Amazon Keyspaces の DDL ステートメント (データ定義言語)
- Amazon Keyspaces の DML ステートメント (データ操作言語)
- Amazon Keyspaces の組み込み関数

Amazon Keyspaces における Cassandra クエリ言語 (CQL) 要素

識別子、定数、用語、データ型など、Amazon Keyspaces でサポートされている Cassandra クエリ言語 (CQL) 要素について説明します。

トピック

- 識別子
- 定数
- 用語
- データ型
- Amazon Keyspaces データ型の JSON エンコード

識別子

識別子 (または名前) は、テーブル、列、およびその他のオブジェクトの識別に使用されます。識別子は、引用符で囲んだものも、囲んでいないものも有効です。以下が適用されます。

```
identifier          ::=  unquoted_identifier | quoted_identifier
unquoted_identifier ::=  re('[a-zA-Z][a-zA-Z0-9_]*')
quoted_identifier   ::=  '"' (any character where " can appear if doubled)+ '"'
```

定数

以下の定数が定義されます。

```
constant ::=  string | integer | float | boolean | uuid | blob | NULL
string   ::=  '\'' (any character where ' can appear if doubled)+ '\''
              '$$' (any character other than '$$') '$$'
integer  ::=  re('-?[0-9]+')
float    ::=  re('-?[0-9]+(\.[0-9]*)?([eE][+-]?[0-9+])?') | NAN | INFINITY
boolean  ::=  TRUE | FALSE
uuid     ::=  hex{8}-hex{4}-hex{4}-hex{4}-hex{12}
hex      ::=  re("[0-9a-fA-F]")
blob     ::=  '0' ('x' | 'X') hex+
```

用語

用語は、サポートされている値の種類を表します。用語は以下により定義されます。

```
term                ::=  constant | literal | function_call | arithmetic_operation |
 type_hint | bind_marker
literal             ::=  collection_literal | tuple_literal
function_call       ::=  identifier '(' [ term (',' term)* ] ')'
```

```
arithmetic_operation ::=  '-' term | term ('+' | '-' | '*' | '/' | '%') term
```

データ型

Amazon Keyspaces では、次のデータ型がサポートされています。

文字列型

データ型	説明
ascii	ASCII 文字列を表します。
text	UTF-8 でエンコードされた文字列を表します。
varchar	UTF-8 でエンコードされた文字列を表します (varchar は text のエイリアス)。

数値型

データ型	説明
bigint	64 ビットの符号付き長整数を表します。
counter	64 ビットの符号付き整数カウンタを表します。詳細については、「the section called “カウンタ”」を参照してください。
decimal	可変精度の 10 進数を表します。
double	64 ビット の IEEE 754 浮動小数点を表します。
float	32 ビット の IEEE 754 浮動小数点を表します。
int	32 ビットの符号付き整数を表します。
varint	任意精度の整数を表します。

カウンタ

counter 列に 64 ビットの符号付き整数が含まれます。カウンタ値は、the section called “UPDATE” ステートメントを用いて増加または減少されます。直接設定することはできません。これにより、カウントの追跡に役立つ counter 列が作成されます。例えば、カウンタを使用して、ログファイル内のエントリ数や、ソーシャルネットワークで投稿が閲覧された回数を追跡することができます。counter 列には以下の制限があります。

- counter 型の列をテーブルの primary key の一部にすることはできません。
- counter 型の列が 1 つ以上含まれているテーブルでは、そのテーブルのすべての列は counter 型の列でなければなりません。

カウンタの更新が失敗した場合 (タイムアウトや Amazon Keyspaces との接続の喪失など)、カウンタ値の更新の有無がクライアントに認識されません。更新が再試行された場合、カウンタ値への更新が 2 回目に適用されることがあります。

BLOB 型

データ型	説明
blob	任意のバイトを表します。

ブール型

データ型	説明
boolean	true または false を表します。

時間関連の型

データ型	説明
date	形式の文字列<yyyy>-<mm>-<dd> 。

データ型	説明
timestamp	エポック (1970 年 1 月 1 日 00:00:00 GMT) からの日付と時刻をミリ秒単位で表す 64 ビットの符号付き整数。
timeuuid	バージョン 1 UUID を表します。

コレクション型

データ型	説明
list	順序が設定された一連のリテラル要素を表します。
map	順序が設定されていない一連のキーバリューペアを表します。
set	順序を設定した一連のリテラル要素 (1 つ以上) を表します。

コレクション型でコレクション列を宣言し、その後に山括弧で囲んだ別のデータ型 (たとえば、TEXT または INT) を使用します。次の例のようにTEXT の SET で列を作成することも、TEXT および INT のキーと値のペアの MAP を作成することもできます。

```
SET <TEXT>
MAP <TEXT, INT>
```

非フリーズコレクションでは、個々のコレクション要素を更新できます。クライアント側のタイムスタンプと有効期限 (TTL) 設定は、個々の要素ごとに保存されます。

FROZEN コレクション型にキーワードを使用すると、コレクションの値は 1 つの不変の値にシリアル化され、Amazon Keyspaces ではそれらを BLOB のように扱われます。これはフリーズコレクションです。INSERT または UPDATE ステートメントはフリーズされたコレクション全体を上書きします。フリーズコレクション内の個々の要素は更新できません。

クライアント側のタイムスタンプと有効期限 (TTL) の設定は、個々の要素ではなく、フリーズされたコレクション全体に適用されます。Frozen コレクション列は PRIMARY KEY のテーブルの一部にすることができます。

フリーズコレクションはネストできます。たとえば、次の例のように、MAP が FROZEN キーワードを使用している場合、SET 内の MAP を定義できます。

```
SET <FROZEN> <MAP <TEXT, INT>>>
```

Amazon Keyspaces では、デフォルトで最大 8 レベルのフリーズコレクションをネストできます。詳細については、「the section called “Amazon Keyspaces サービスクォータ”」を参照してください。Apache Cassandra との機能の違いの詳細については、「the section called “FROZEN collections”」を参照してください。CQL 構文の詳細については、「the section called “CREATE TABLE”」と「the section called “ALTER TABLE”」を参照してください。

タプル型

tuple データ型は、リテラル要素の有境グループを表します。user defined type の代わりにタプルを使用できます。タプルには FROZEN キーワードを使用する必要はありません。これは、タプルは常にフリーズされており、要素を個別に更新することはできないからです。

その他の型

データ型	説明
inet	IPv4 形式または IPv6 形式の IP アドレスを表す文字列。

静的

クラスタリング列を含む Amazon Keyspaces テーブルでは、STATIC キーワードを使用して、任意の型の静的列を作成できます。

以下のステートメントは、この例です。

```
my_column INT STATIC
```

静的列の操作の詳細については、「the section called “静的列のキャパシティ消費量の推定”」を参照してください。

ユーザー定義タイプ (UDTs)

Amazon Keyspaces はユーザー定義タイプ (UDTsをサポートしています。コレクションやその他の既存の UDT など、任意の有効な Amazon Keyspaces データ型を使用して UDTs を作成できます。キースペースに UDTs を作成し、それを使用してキースペース内の任意のテーブルの列を定義できます。

構文の詳細については、「the section called “型”」を参照してください。UDTs「」を参照してくださいthe section called “ユーザー定義タイプ (UDTs)”。

キースペースごとにサポートされている UDTs の数、サポートされているネストレベル、および UDTs「」を参照してくださいthe section called “Amazon Keyspaces のユーザー定義タイプ (UDTs クォータとデフォルト値”。

Amazon Keyspaces データ型の JSON エンコード

Amazon Keyspaces の JSON データ型マッピングは Apache Cassandra のものと同じです。次の表では、`INSERT JSON` ステートメントで Amazon Keyspaces に許容されるデータ型と、Amazon Keyspaces により `SELECT JSON` ステートメントともにデータが返される場合に使用されるデータ型について説明します。

`float`、`int`、`UUID`、`date` などの単一フィールドデータ型の場合は、データを `string` として挿入することもできます。`tuple`、`map`、`list` などの複合的なデータ型とデータ収集については、JSON またはエンコードされた `JSON string` としてデータを挿入することもできます。

JSON データ型	**`INSERT JSON`** ステートメントで許容されたデータ型	**`SELECT JSON`** ステートメントで返却されたデータ型	メモ
`ascii`	`string`	`string`	JSON 文字エスケープ `\u` を使用します。
`bigint`	`integer, string`	`integer`	文字列は有効な 64 ビット整数でなければなりません。
`blob`	`string`	`string`	文字列の最初は `0x` で、すぐ後に偶数の 16 進数が続きます。

JSON データ型	**INSERT JSON** ステートメントで許容されたデータ型	**SELECT JSON** ステートメントで返却されたデータ型	メモ
boolean	boolean, string	boolean	文字列は true または false のいずれかでなければなりません。
date	string	string	YYYY-MM-DD 形式、タイムゾーン UTC の日付。
decimal	integer, float, string	float	クライアント側のデコーダーで、32 ビットまたは 64 ビットの IEEE-754 浮動小数点精度を超えることがあります。
double	integer, float, string	float	文字列は有効な整数または浮動小数点でなければなりません。
float	integer, float, string	float	文字列は有効な整数または浮動小数点でなければなりません。
inet	string	string	IPv4 または IPv6 アドレス。
int	integer, string	integer	文字列は有効な 32 ビット整数でなければなりません。

JSON データ型	**INSERT JSON** ステートメントで許容されたデータ型	**SELECT JSON** ステートメントで返却されたデータ型	メモ
list	list, string	list	ネイティブ JSON リスト表現を使用します。
map	map, string	map	ネイティブ JSON マップ表現を使用します。
smallint	integer, string	integer	文字列は有効な 16 ビット整数でなければなりません。
set	list, string	list	ネイティブ JSON リスト表現を使用します。
text	string	string	JSON 文字エスケープ \u を使用します。
time	string	string	HH-MM-SS[.fffffffff] 形式の時刻。
timestamp	integer, string	string	タイムスタンプ。文字列定数を使用するとタイムスタンプを日付として保存できます。YYYY-MM-DD HH:MM:SS.SSS 形式の日付スタンプが返されます。

JSON データ型	**INSERT JSON** ステートメントで許容されたデータ型	**SELECT JSON** ステートメントで返却されたデータ型	メモ
timeuuid	string	string	1 UUID 型。UUID 形式については「constants」を参照してください。
tinyint	integer, string	integer	文字列は有効な 8 ビット整数でなければなりません。
tuple	list, string	list	ネイティブ JSON リスト表現を使用します。
UDT	map, string	map	フィールド名をキーとするネイティブ JSON マップ表現を使用します。
uuid	string	string	UUID 形式については「constants」を参照してください。
varchar	string	string	JSON 文字エスケープ \u を使用します。
varint	integer, string	integer	可変長。クライアント側のデコーダーで 32 ビットまたは 64 ビットの整数をオーバーフローする可能性があります。

Amazon Keyspaces の DDL ステートメント (データ定義言語)

データ定義言語 (DDL) は、キー空間やテーブルなどの Amazon Keyspaces (Apache Cassandra 向け) のデータ構造を管理するために使用する一連の Cassandra クエリ言語 (CQL) ステートメントです。DDL を使用して、これらのデータ構造の作成、作成後の変更、および、使用しなくなったときの削除を行います。Amazon Keyspaces では DDL オペレーションが非同期的に実行されます。非同期操作が完了したことを確認する方法の詳細については、「the section called “キースペースとテーブルの非同期的な作成および削除”」を参照してください。

次の DDL ステートメントがサポートされています。

- CREATE KEYSPACE
- ALTER KEYSPACE
- DROP KEYSPACE
- 使用
- CREATE TABLE
- ALTER TABLE
- RESTORE TABLE
- DROP TABLE
- タイプの作成
- ドロップタイプ

トピック

- Keyspaces
- テーブル
- ユーザー定義タイプ (UDTs)

Keyspaces

キー空間は、1 つ以上のアプリケーションに関係している関連テーブルをグループ化するものです。リレーショナルデータベース管理システム (RDBMS) に関しては、キー空間が、データベース、テーブルスペース、または類似の構造とほぼ同じです。

> **Note**
>
> Apache Cassandra では、キー空間により、複数のストレージノードにおけるデータのレプリケーション方法が決まります。ただし、Amazon Keyspaces はフルマネージドサービスであり、ストレージレイヤーの詳細情報がユーザーに代わって管理されます。このため、Amazon Keyspaces の Keyspaces は論理構造のみであり、基礎となる物理ストレージとは関係ありません。

Amazon Keyspaces キー空間のクォータ制限と制約については、「クォータ」を参照してください。

キースペースのステートメント

- CREATE KEYSPACE
- ALTER KEYSPACE
- DROP KEYSPACE
- 使用

CREATE KEYSPACE

`CREATE KEYSPACE` ステートメントを使用して、新しいキー空間を作成します。

[Syntax (構文)]

```
create_keyspace_statement ::=
    CREATE KEYSPACE [ IF NOT EXISTS ] keyspace_name
    WITH options
```

Where:

- *keyspace_name* は作成するキー空間の名前です。
- options は以下のうちの 1 つ以上です。
  - `REPLICATION` — キー空間のレプリケーション戦略を示すマップ。
    - `SingleRegionStrategy` — 単一リージョンのキー空間用。(必須)
    - `NetworkTopologyStrategy` – 少なくとも 2 つ指定します AWS リージョン。各リージョンのレプリケーション係数は 3 です。(オプション)

- DURABLE_WRITES — Amazon Keyspaces への書き込みは常に耐久性があるため、このオプションは必要ありません。ただし、指定する場合は、値は true でなければなりません。
- TAGS – 作成時にリソースにアタッチされるキー値のペアタグのリスト。(オプション)

例

次のようなキー空間を作成します。

```
CREATE KEYSPACE my_keyspace
    WITH REPLICATION = {'class': 'SingleRegionStrategy'} and TAGS ={'key1':'val1',
 'key2':'val2'} ;
```

マルチリージョンキースペースを作成するには、 を指定NetworkTopologyStrategyし、少なくとも 2 つ含めます AWS リージョン。各リージョンのレプリケーション係数は 3 です。

```
CREATE KEYSPACE my_keyspace
    WITH REPLICATION = {'class':'NetworkTopologyStrategy', 'us-east-1':'3', 'ap-
southeast-1':'3','eu-west-1':'3'};
```

ALTER KEYSPACE

ALTER KEYSPACE WITH ステートメントは、次のオプションに使用できます。

- REPLICATION – このオプションを使用して、キースペースに新しい AWS リージョン レプリカを追加します。新しいリージョンを単一リージョンまたはマルチリージョンのキースペースに追加できます。
- TAGS – このオプションを使用して、キースペースにタグを追加または削除します。

[Syntax] (構文)

```
alter_keyspace_statement ::=
    ALTER KEYSPACE keyspace_name
    WITH options
```

Where:

- *keyspace_name* は変更するキー空間の名前です。
- オプションは次のいずれかです。

- ADD | DROP TAGS — キー空間で追加または削除されるキーバリューペアタグのリスト。
- REPLICATION – キースペースのレプリケーション戦略を示すマップ。
  - class– キースペースをマルチリージョンキースペースとしてNetworkTopologyStrategy定義します。
  - region– このキースペース AWS リージョン に追加の 1 つを指定します。各リージョンのレプリケーション係数は 3 です。
  - CLIENT_SIDE_TIMESTAMPS – デフォルトは ですDISABLED。ステータスは にのみ変更できますENABLED。

例

次の例に示すようにキースペースを変更してタグを追加します。

```
ALTER KEYSPACE my_keyspace ADD TAGS {'key1':'val1', 'key2':'val2'};
```

マルチリージョンキースペースに 3 番目のリージョンを追加するには、次のステートメントを使用できます。

```
ALTER KEYSPACE my_keyspace
WITH REPLICATION = {
    'class': 'NetworkTopologyStrategy',
    'us-east-1': '3',
    'us-west-2': '3',
    'us-west-1': '3'
} AND CLIENT_SIDE_TIMESTAMPS = {'status': 'ENABLED'};
```

DROP KEYSPACE

DROP KEYSPACE ステートメントを使用してキー空間(テーブルなどあらゆるコンテンツを含む)を削除します。

[Syntax (構文)]

```
drop_keyspace_statement ::=
    DROP KEYSPACE [ IF EXISTS ] keyspace_name
```

Where:

- keyspace_name は、削除 (ドロップ) されるキー空間の名前です。

例

```
DROP KEYSPACE my_keyspace;
```

使用

USE ステートメントを使用して、現在のキースペースを定義します。これにより、キースペースプレフィックスを含む完全修飾名を使用せずに、テーブルやタイプなど、特定のキースペースにバインドされたオブジェクトを参照できます。

[Syntax] (構文)

```
use_statement ::=
    USE keyspace_name
```

Where:

- keyspace_name は、使用するキースペースの名前です。

例

```
USE my_keyspace;
```

テーブル

テーブルは Amazon Keyspaces の主要なデータ構造です。テーブル内のデータは行と列で編成されます。これらの列のサブセットは、パーティションキーの指定によるパーティショニング (および最終的にはデータ配置) を決定するために使用されます。

別の列セットをクラスタリング列に定義できます。つまり、クエリ実行に述語として盛り込めるということです。

デフォルトでは、新規のテーブルはオンデマンドのスループットキャパシティがある状態で作成されます。新規のテーブルと既存のテーブルのキャパシティモードは変更できます。読み込み/書き込みのキャパシティスループットモードの詳細については、「the section called “読み取り/書き込みキャパシティモードの設定”」を参照してください。

プロビジョンドモードのテーブルでは、オプションの AUTOSCALING_SETTINGS を設定できます。Amazon Keyspaces の自動スケーリングと使用可能なオプションの詳細については、「the section called “既存のテーブルに対して自動スケーリングを設定する”」を参照してください。

Amazon Keyspaces テーブルのクォータ制限と制約については、「クォータ」を参照してください。

テーブルのステートメント

- CREATE TABLE
- ALTER TABLE
- RESTORE TABLE
- DROP TABLE

CREATE TABLE

CREATE TABLE ステートメントを使用して新しいテーブルを作成します。

[Syntax (構文)]

```
create_table_statement ::=  CREATE TABLE [ IF NOT EXISTS ] table_name
    '('
        column_definition
        ( ',' column_definition )*
        [ ',' PRIMARY KEY '(' primary_key ')' ]
    ')' [ WITH table_options ]

column_definition      ::=  column_name cql_type [ FROZEN ][ STATIC ][ PRIMARY KEY]

primary_key            ::=  partition_key [ ',' clustering_columns ]

partition_key          ::=  column_name
                              | '(' column_name ( ',' column_name )* ')'

clustering_columns     ::=  column_name ( ',' column_name )*

table_options          ::=  [table_options]
                              | CLUSTERING ORDER BY '(' clustering_order
 ')' [ AND table_options ]
                              | cdc
                              | CUSTOM_PROPERTIES
                              | AUTOSCALING_SETTINGS
                              | default_time_to_live
                              | TAGS
```

```
clustering_order        ::=  column_name (ASC | DESC) ( ',' column_name (ASC | DESC) )*
```

Where:

- *table_name* は作成するテーブルの名前です。完全修飾名には、キースペースプレフィックスが含まれます。または、keyspace ステートメントを使用して現在のUSEキースペースを設定することもできます。
- *column_definition* は以下で構成されます。
  - *column_name* – 列の名前。
  - *cql_type* — Amazon Keyspaces のデータ型 (「データ型」を参照)。
  - *FROZEN* – ユーザー定義またはタイプ collection (、、 などSETMAP) LISTのこの列をフリーズとして指定します。フリーズコレクションは 1 つの不変の値にシリアル化され、BLOBと同様に扱われます。詳細については、「the section called “コレクション型”」を参照してください。
  - *STATIC* — この列を静的として指定します。静的列には、同じパーティション内のすべての行で共有される値が保存されます。
  - *PRIMARY KEY* — この列をテーブルのプライマリキーとして指定します。
- *primary_key* は以下で構成されます。
  - partition_key
  - clustering_columns
- *partition_key*:
  - パーティションキーは、1 つの列である場合もあれば、2 つ以上の列で構成される複合値である場合もあります。プライマリキーのパーティションキー部分は必須で、これによって Amazon Keyspaces におけるデータの保存方法が決まります。
- *clustering_columns*:
  - プライマリキーのオプションのクラスタリング列部分は、各パーティションにおけるデータのクラスター処理方法とソート方法を決定するものです。
- *table_options* は以下で構成されます。
  - *CLUSTERING ORDER BY* — テーブルのデフォルトの CLUSTERING ORDER は、ASC (昇順) ソート方向のクラスタリングキーで構成されます。デフォルトのソート動作をオーバーライドするには、この値を指定します。
  - *cdc* – Amazon Keyspaces がテーブルの変更データキャプチャ (CDC) ストリームを作成するかどうかを指定するブール値。デフォルトは false です。ストリームを有効にするview typeときに を指定するには、 cdc specificationで CUSTOM_PROPERTIES を設定します。

- *CUSTOM_PROPERTIES* — Amazon Keyspaces に固有の設定のマップ。
  - capacity_mode: テーブルの読み取り/書き込みスループットキャパシティモードを指定します。オプションは throughput_mode:PAY_PER_REQUEST と throughput_mode:PROVISIONED です。プロビジョンドキャパシティモードには入力として read_capacity_units と write_capacity_units が必要です。デフォルトは throughput_mode:PAY_PER_REQUEST です。
  - cdc_specification: CDC ストリームview_typeの を指定します。オプションは以下のとおりです。
    - NEW_AND_OLD_IMAGES – 変更前と変更後の両方のバージョンの行。これがデフォルトです。
    - NEW_IMAGE – 変更後の行のバージョン。
    - OLD_IMAGE – 変更前の行のバージョン。
    - KEYS_ONLY – 変更された行のパーティションキーとクラスタリングキー。

    CDC ストリームの詳細については、「」を参照してくださいthe section called “CDC ストリームの使用”。コード例については、the section called “新しいテーブルの CDC ストリームを有効にする” を参照してください。
  - client_side_timestamps: テーブルに対してクライアント側のタイムスタンプを有効にするか無効にするかを指定します。オプションは {'status': 'enabled'} と {'status': 'disabled'} です。指定しない場合、デフォルトの status:disabled になります。クライアント側のタイムスタンプをテーブルで有効にした後は、この設定を無効にすることはできません。
  - encryption_specification: 保管データ暗号化の暗号化オプションを指定します。指定しない場合、デフォルトの encryption_type:AWS_OWNED_KMS_KEY になります。暗号化オプションのカスタマーマネージドキーでは、入力として Amazon リソースネーム (ARN) 形式の AWS KMS キーが必要です: kms_key_identifier:ARN: kms_key_identifier:ARN。
  - point_in_time_recovery: テーブルに対してポイントインタイムリストアを有効にするか無効にするかを指定します。オプションは status:enabled と status:disabled です。指定しない場合、デフォルトの status:disabled になります。
  - replica_updates: マルチリージョンテーブルの AWS リージョン固有の設定を指定します。マルチリージョンテーブルの場合、テーブルの読み取りキャパシティの設定を AWS リージョンごとに変えることができます。その場合は、次のパラメータを設定します。詳細な説明と例についてはthe section called “プロビジョンドモードのマルチリージョンテーブルの作成”を参照してください。

- region – 次の設定のテーブルレプリカ AWS リージョン の 。
  - read_capacity_units
- TTL: テーブルの 有効期限 (TTL) カスタム設定を有効にします。有効にする場合は、status:enabled を使用します。デフォルトは status:disabled です。TTL を有効にした後、そのテーブルに対してそれを無効にすることはできません。

- *AUTOSCALING_SETTINGS* には、プロビジョンドモードのテーブルに対する次のオプションの設定が含まれています。詳細な説明と例についてはthe section called “自動スケーリングを有効にして新規テーブルを作成する”を参照してください。
  - provisioned_write_capacity_autoscaling_update:
    - autoscaling_disabled – 書き込みキャパシティの自動スケーリングを有効にするには、値を false に設定します。デフォルトは true です。(オプション)
    - minimum_units – テーブルで常に処理できるようにしておくべき最低限の書き込みスループット。1 から、アカウントの秒単位のスループット上限 (デフォルトは 40,000) までの値を指定する必要があります。
    - maximum_units – テーブルで常に処理できるようにしておくべき最大限の書き込みスループット。1 から、アカウントの秒単位のスループット上限 (デフォルトは 40,000) までの値を指定する必要があります。
    - scaling_policy – Amazon Keyspaces はターゲット追跡ポリシーに対応しています。自動スケーリングのターゲットは、テーブルのプロビジョニング済みの書き込みキャパシティです。
      - target_tracking_scaling_policy_configuration – ターゲット追跡ポリシーを定義するには、目標 (ターゲット) 値を定義する必要があります。ターゲット追跡とクールダウン期間の詳細については、「Application Auto Scaling ユーザーガイド」の「ターゲット追跡スケーリングポリシー」を参照してください。
        - target_value – テーブルの目標とする使用率。Amazon Keyspaces の自動スケーリングでは、プロビジョニング済みのキャパシティに占める実際のキャパシティ消費の割合が、この値と同じか近い値で維持されます。target_value をパーセンテージとして定義します。20 ~ 90 の倍精度浮動小数点数。(必須)
        - scale_in_cooldown – スケーリングアクティビティ間のクールダウン期間 (秒数)。この期間を使って、別のスケールインアクティビティが開始する前にテーブルを安定させます。値を指定しない場合、デフォルト値は 0 です。(オプション)
        - scale_out_cooldown – スケーリングアクティビティ間のクールダウン期間 (秒数)。この期間を使って、別のスケールアウトアクティビティが開始する前にテーブルを安定させます。値を指定しない場合、デフォルト値は 0 です。(オプション)

- disable_scale_in: テーブルに対して scale-in が無効か有効かを指定するboolean。このパラメータは、デフォルトでは無効になっています。scale-in を有効にするには、boolean 値を FALSE に設定します。その場合は、テーブルのキャパシティが自動的にスケールダウンされます。(オプション)

- provisioned_read_capacity_autoscaling_update:
  - autoscaling_disabled – 読み取りキャパシティの自動スケーリングを有効にするには、値を false に設定します。デフォルトは true です。(オプション)
  - minimum_units – テーブルで常に処理できるようにしておくべき最低限のスループット。1 から、アカウントの秒単位のスループット上限 (デフォルトは 40,000) までの値を指定する必要があります。
  - maximum_units – テーブルで常に処理できるようにしておくべき最大限のスループット。1 から、アカウントの秒単位のスループット上限 (デフォルトは 40,000) までの値を指定する必要があります。
  - scaling_policy – Amazon Keyspaces はターゲット追跡ポリシーに対応しています。自動スケーリングのターゲットは、テーブルのプロビジョニング済みの読み取りキャパシティです。
    - target_tracking_scaling_policy_configuration – ターゲット追跡ポリシーを定義するには、目標 (ターゲット) 値を定義する必要があります。ターゲット追跡とクールダウン期間の詳細については、「Application Auto Scaling ユーザーガイド」の「ターゲット追跡スケーリングポリシー」を参照してください。
      - target_value – テーブルの目標とする使用率。Amazon Keyspaces の自動スケーリングでは、プロビジョニング済みのキャパシティに占める実際のキャパシティ消費の割合が、この値と同じか近い値で維持されます。target_value をパーセンテージとして定義します。20 ~ 90 の倍精度浮動小数点数。(必須)
      - scale_in_cooldown – スケーリングアクティビティ間のクールダウン期間 (秒数)。この期間を使って、別のスケールインアクティビティが開始する前にテーブルを安定させます。値を指定しない場合、デフォルト値は 0 です。(オプション)
      - scale_out_cooldown – スケーリングアクティビティ間のクールダウン期間 (秒数)。この期間を使って、別のスケールアウトアクティビティが開始する前にテーブルを安定させます。値を指定しない場合、デフォルト値は 0 です。(オプション)
      - disable_scale_in: テーブルに対して scale-in が無効か有効かを指定するboolean。このパラメータは、デフォルトでは無効になっています。scale-in を有効にするには、boolean 値を FALSE に設定します。その場合は、テーブルのキャパシティが自動的にスケールダウンされます。(オプション)

- `replica_updates`: マルチリージョンテーブルの AWS リージョン 特定の自動スケーリング設定を指定します。マルチリージョンテーブルの場合、テーブルの読み取りキャパシティの設定を AWS リージョンごとに変えることができます。その場合は、次のパラメータを設定します。詳細な説明と例についてはthe section called “マルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定の更新”を参照してください。
  - `region` – 次の設定のテーブルレプリカ AWS リージョン の 。
    - `provisioned_read_capacity_autoscaling_update`
      - `autoscaling_disabled` – テーブルの読み取りキャパシティの自動スケーリングを有効にするには、値を `false` に設定します。デフォルトは `true` です。(オプション)

        > **Note**
        >
        > マルチリージョンテーブルの自動スケーリングは、そのテーブルのすべてのレプリカに対して有効または無効にする必要があります。

      - `minimum_units` – テーブルで常に処理できるようにしておくべき最低限の読み取りスループット。1 から、アカウントの秒単位のスループット上限 (デフォルトは 40,000) までの値を指定する必要があります。
      - `maximum_units` – テーブルで常に処理できるようにしておくべき最大限の読み取りスループット。1 から、アカウントの秒単位のスループット上限 (デフォルトは 40,000) までの値を指定する必要があります。
      - `scaling_policy` – Amazon Keyspaces はターゲット追跡ポリシーに対応しています。自動スケーリングのターゲットは、テーブルのプロビジョニング済みの読み取りキャパシティです。
        - `target_tracking_scaling_policy_configuration` – ターゲット追跡ポリシーを定義するには、目標 (ターゲット) 値を定義する必要があります。ターゲット追跡とクールダウン期間の詳細については、「Application Auto Scaling ユーザーガイド」の「ターゲット追跡スケーリングポリシー」を参照してください。
          - `target_value` – テーブルの目標とする使用率。Amazon Keyspaces の自動スケーリングは、プロビジョニング済みの読み取りキャパシティに占める実際の読み取りキャパシティ消費の割合が、この値と同じか近い値で維持されます。`target_value` をパーセンテージとして定義します。20 ~ 90 の倍精度浮動小数点数。(必須)

- scale_in_cooldown – スケーリングアクティビティ間のクールダウン期間 (秒数)。この期間を使って、別のスケールインアクティビティが開始する前にテーブルを安定させます。値を指定しない場合、デフォルト値は 0 です。(オプション)
- scale_out_cooldown – スケーリングアクティビティ間のクールダウン期間 (秒数)。この期間を使って、別のスケールアウトアクティビティが開始する前にテーブルを安定させます。値を指定しない場合、デフォルト値は 0 です。(オプション)
- disable_scale_in: テーブルに対して scale-in が無効か有効かを指定するboolean。このパラメータは、デフォルトでは無効になっています。scale-in を有効にするには、boolean 値を FALSE に設定します。つまり、テーブルの読み取りキャパシティが自動的にスケールダウンされます。(オプション)

- default_time_to_live — テーブルのデフォルトの有効期限 (TTL) 設定 (秒)。
- TAGS – 作成時にリソースにアタッチされるキーバリューペアタグのリスト。
- *clustering_order* は以下で構成されます。
  - *column_name* – 列の名前。
  - *ASC | DESC* — 昇順修飾子 (ASC) または降順修飾子 (DESC) を設定します。指定しない場合、デフォルトの順序 ASC になります。

例

```
CREATE TABLE IF NOT EXISTS my_keyspace.my_table (
                                            id text,
                                            name text,
                                            region text,
                                            division text,
                                            project text,
                                            role text,
                                            pay_scale int,
                                            vacation_hrs float,
                                            manager_id text,
                                            PRIMARY KEY (id,division))
                                            WITH CUSTOM_PROPERTIES={
                                                'capacity_mode':{
                                                        'throughput_mode':
 'PROVISIONED', 'read_capacity_units': 10, 'write_capacity_units': 20
                                                    },
                                                'point_in_time_recovery':{'status':
 'enabled'},
                                                'encryption_specification':{
```

```
                                                    'encryption_type':
 'CUSTOMER_MANAGED_KMS_KEY',

 'kms_key_identifier':'arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111'
                                                        }
                                                }
                                                AND CLUSTERING ORDER BY (division ASC)
                                                AND TAGS={'key1':'val1', 'key2':'val2'}
                                                AND default_time_to_live = 3024000;
```

クラスタリング列を使用しているテーブルでは、テーブル定義で非クラスタリング列を静的として宣言できます。静的列の詳細については、「the section called “静的列のキャパシティ消費量の推定”」を参照してください。

例

```
CREATE TABLE my_keyspace.my_table (
                                                id int,
                                                name text,
                                                region text,
                                                division text,
                                                project text STATIC,
                                                PRIMARY KEY (id,division));
```

ユーザー定義タイプ (UDT) を使用する列を使用してテーブルを作成できます。例の最初のステートメントは タイプを作成し、2 番目のステートメントは タイプを使用する列を持つテーブルを作成します。

例

```
CREATE TYPE my_keyspace."udt""N@ME" (my_field int);
CREATE TABLE my_keyspace.my_table (my_col1 int pri key, my_col2 "udt""N@ME");
```

ALTER TABLE

ALTER TABLE ステートメントを使用して、新しい列の追加、タグの追加、テーブルのカスタムプロパティの変更を行います。

[Syntax] (構文)

```
alter_table_statement ::=  ALTER TABLE table_name

        [ ADD ( column_definition | column_definition_list)  ]
        [[ADD | DROP] TAGS {'key1':'val1', 'key2':'val2'}]
        [ WITH table_options [ , ... ] ] ;

column_definition      ::=  column_name cql_type
```

Where:

- *table_name* は変更するテーブルの名前です。
- *column_definition* は追加する列の名前とデータ型です。
- *column_definition_list* は括弧内に配置された列のリストでカンマで区切られています。
- *table_options* は以下で構成されます。
  - *AUTOSCALING_SETTINGS* には、プロビジョニング済みのテーブルに対するオプションの自動スケーリング設定が含まれています。構文と詳しい説明については、「the section called “CREATE TABLE”」を参照してください。例については「the section called “既存のテーブルに対して自動スケーリングを設定する”」を参照してください。
  - *cdc* – Amazon Keyspaces がテーブルの変更データキャプチャ (CDC) ストリームを作成するかどうかを指定するブール値。デフォルトは `false` です。ストリームを有効にするview_typeときに を指定するには、 cdc_specificationで CUSTOM_PROPERTIES を設定します。
  - *CUSTOM_PROPERTIES* — Amazon Keyspaces に固有の設定のマップ。
    - capacity_mode: テーブルの読み取り/書き込みスループットキャパシティモードを指定します。オプションは throughput_mode:PAY_PER_REQUEST と throughput_mode:PROVISIONED です。プロビジョンドキャパシティモードには入力として read_capacity_units と write_capacity_units が必要です。デフォルトは throughput_mode:PAY_PER_REQUEST です。
    - cdc_specification: CDC ストリームview_typeの を指定します。オプションは以下のとおりです。
      - NEW_AND_OLD_IMAGES – 変更前と変更後の両方のバージョンの行。これがデフォルトです。
      - NEW_IMAGE – 変更後の行のバージョン。
      - OLD_IMAGE – 変更前の行のバージョン。

    - KEYS_ONLY – 変更された行のパーティションキーとクラスタリングキー。

      CDC ストリームの詳細については、「」を参照してくださいthe section called “CDC ストリームの使用”。コード例については、the section called “既存のテーブルの CDC ストリームを有効にする” を参照してください。

  - client_side_timestamps: テーブルに対してクライアント側のタイムスタンプを有効にするか無効にするかを指定します。オプションは {'status': 'enabled'} と {'status': 'disabled'} です。指定しない場合、デフォルトの status:disabled になります。クライアント側のタイムスタンプをテーブルで有効にした後は、この設定を無効にすることはできません。
  - encryption_specification: 保管データ暗号化の暗号化オプションを指定します。オプションは encryption_type:AWS_OWNED_KMS_KEY と encryption_type:CUSTOMER_MANAGED_KMS_KEY です。暗号化オプションのカスタマーマネージドキーには、入力として Amazon リソースネーム (ARN) 形式の AWS KMS キーが必要です: kms_key_identifier:ARN。
  - point_in_time_recovery: テーブルに対してポイントインタイムリストアを有効にするか無効にするかを指定します。オプションは status:enabled と status:disabled です。デフォルトは status:disabled です。
  - replica_updates: マルチリージョンテーブルの特定 AWS リージョン の設定を指定します。マルチリージョンテーブルの場合、テーブルの読み取りキャパシティの設定を AWS リージョンごとに変えることができます。その場合は、次のパラメータを設定します。詳細な説明と例についてはthe section called “マルチリージョンテーブルのプロビジョンドキャパシティと自動スケーリングの設定の更新”を参照してください。
    - region – 次の設定のテーブルレプリカ AWS リージョン の 。
      - read_capacity_units
  - ttl: テーブルの 有効期限 (TTL) カスタム設定を有効にします。有効にする場合は、status:enabled を使用します。デフォルトは status:disabled です。ttl を有効にした後、そのテーブルに対してそれを無効にすることはできません。
- default_time_to_live: テーブルのデフォルトの有効期限 (TTL) 設定 (秒)。
- *TAGS* はリソースにアタッチされるキーバリューペアタグのリストです。

> **Note**
>
> ALTER TABLE では、1 つのカスタムプロパティしか変更できません。同一ステートメント内で複数の ALTER TABLE コマンドを組み合わせることはできません。

例

次のステートメントは、既存のテーブルに列を追加する方法を示しています。

```
ALTER TABLE mykeyspace.mytable ADD (ID int);
```

このステートメントは、既存のテーブルに 2 つのコレクション列を追加する方法を示します。

- ネストされたフリーズコレクションを含むフリーズコレクション列 col_frozen_list
- ネストされたフリーズコレクションを含む非フリーズコレクション列 col_map

```
ALTER TABLE my_Table ADD(col_frozen_list FROZEN<LIST<FROZEN<SET<TEXT>>>>, col_map
 MAP<INT, FROZEN<SET<INT>>>);
```

次の例は、ユーザー定義タイプ (UDT) を使用する列をテーブルに追加する方法を示しています。

```
ALTER TABLE my_keyspace.my_table ADD (my_column, my_udt;);
```

テーブルのキャパシティモードを変更し、読み取りおよび書き込みのキャパシティユニットを指定するには、次のステートメントを使用します。

```
ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 10, 'write_capacity_units':
 20}};
```

次のステートメントでは、テーブルのカスタマー管理 KMS キーが指定されます。

```
ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={
              'encryption_specification':{
                     'encryption_type': 'CUSTOMER_MANAGED_KMS_KEY',
                     'kms_key_identifier':'arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111'
```

```
                }
        };
```

テーブルのポイントインタイムリストアを有効にするには、次のステートメントを使用できます。

```
ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={'point_in_time_recovery':
 {'status': 'enabled'}};
```

テーブルのデフォルトの有効期限 (TTL) 値を秒単位で設定するには、次のステートメントを使用します。

```
ALTER TABLE my_table WITH default_time_to_live = 2592000;
```

このステートメントは、テーブルのカスタムの有効期限 (TTL) 設定を有効にします。

```
ALTER TABLE mytable WITH CUSTOM_PROPERTIES={'ttl':{'status': 'enabled'}};
```

RESTORE TABLE

RESTORE TABLE ステートメントを使用して、特定時点 (ポイントインタイム) にテーブルを復元 (リストア) します。このステートメントでは、テーブルでポイントインタイムリカバリを有効にする必要があります。詳細については、「the section called “ポイントインタイムリカバリによるバックアップと復元”」を参照してください。

[Syntax] (構文)

```
restore_table_statement ::=
    RESTORE TABLE restored_table_name FROM TABLE source_table_name
                    [ WITH table_options [ , ... ] ];
```

Where:

- *restored_table_name* は復元されたテーブルの名前です。
- *source_table_name* はソーステーブルの名前です。
- *table_options* は以下で構成されます。
  - *restore_timestamp* は ISO 8601 形式の復元ポイントタイムです。これが指定されない場合は、現在のタイムスタンプが使用されます。
  - *CUSTOM_PROPERTIES* — Amazon Keyspaces に固有の設定のマップ。

- capacity_mode: テーブルの読み取り/書き込みスループットキャパシティモードを指定します。オプションは throughput_mode:PAY_PER_REQUEST と throughput_mode:PROVISIONED です。プロビジョンドキャパシティモードには入力として read_capacity_units と write_capacity_units が必要です。デフォルトは、ソーステーブルの現在の設定です。
- encryption_specification: 保管データ暗号化の暗号化オプションを指定します。オプションは encryption_type:AWS_OWNED_KMS_KEY と encryption_type:CUSTOMER_MANAGED_KMS_KEY です。暗号化オプションのカスタマーマネージドキーでは、入力として Amazon リソースネーム (ARN) 形式の AWS KMS キー が必要です。 kms_key_identifier:ARNカスタマーマネージドキーで暗号化されたテーブルを で暗号化されたテーブルに復元するには AWS 所有のキー、Amazon Keyspaces はソーステーブルの AWS KMS キーにアクセスする必要があります。
- point_in_time_recovery: テーブルに対してポイントインタイムリストアを有効にするか無効にするかを指定します。オプションは status:enabled と status:disabled です。新しいテーブルを作成する場合とは異なり、復元されたテーブルのデフォルトのステータスは status:enabled になります。これは設定がソーステーブルから継承されるためです。復元されたテーブルの PITR を無効にするには、status:disabled を明示的に設定する必要があります。
- replica_updates: マルチリージョンテーブルの特定 AWS リージョン の設定を指定します。マルチリージョンテーブルの場合、テーブルの読み取りキャパシティの設定を AWS リージョンごとに変えることができます。その場合は、次のパラメータを設定します。
  - region – 次の設定のテーブルレプリカ AWS リージョン の 。
    - read_capacity_units
- *AUTOSCALING_SETTINGS* には、プロビジョニング済みのテーブルに対するオプションの自動スケーリング設定が含まれています。構文と詳しい説明については、「the section called “CREATE TABLE”」を参照してください。
- *TAGS* はリソースにアタッチされるキーバリューペアタグのリストです。

> **Note**
>
> 削除されたテーブルは、削除時にのみ復元できます。

例

```
RESTORE TABLE mykeyspace.mytable_restored from table mykeyspace.my_table
WITH restore_timestamp = '2020-06-30T04:05:00+0000'
AND custom_properties = {'point_in_time_recovery':{'status':'disabled'},
 'capacity_mode':{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 10,
 'write_capacity_units': 20}}
AND TAGS={'key1':'val1', 'key2':'val2'};
```

DROP TABLE

キー空間からテーブルを削除するには DROP TABLE ステートメントを使用します。

[Syntax] (構文)

```
drop_table_statement ::=
    DROP TABLE [ IF EXISTS ] table_name
```

Where:

- IF EXISTS により、テーブルが存在しない場合の DROP TABLE の失敗が阻止されます。(オプション)
- *table_name* は削除 (ドロップ) されるテーブルの名前です。

例

```
DROP TABLE my_keyspace.my_table;
```

ユーザー定義タイプ (UDTs)

UDT – Amazon Keyspaces で 1 つの列を定義するために使用できるフィールドとデータ型のグループ化。UDTs の有効なデータ型は、同じキースペースで既に作成したコレクションやその他の UDTs を含む、サポートされているすべての Cassandra データ型です。サポートされている Cassandra データ型の詳細については、「」を参照してくださいthe section called “Cassandra データ型サポート”。

```
user_defined_type::= udt_name
udt_name::= [ keyspace_name '.' ] identifier
```

タイプのステートメント

- CREATE TYPE
- DROP TYPE

CREATE TYPE

CREATE TYPE ステートメントを使用して新しいタイプを作成します。

[Syntax] (構文)

```
create_type_statement ::=  CREATE TYPE [ IF NOT EXISTS ] udt_name
    '('field_definition ( ',' field_definition)* ')'
            field_definition::= identifier cql_type
```

Where:

- IF NOT EXISTS タイプが既に存在する場合、 は失敗CREATE TYPEしません。(オプション)
- *udt_name* は、タイプ形式の UDT の完全修飾名ですmy_keyspace.my_type。例: 。USE ステートメントを使用して現在のキースペースを定義する場合、キースペース名を指定する必要はありません。
- *field_definition* は名前とタイプで構成されます。

次の表は、許可された UDT 名の例を示しています。最初の列はタイプの作成時に名前を入力する方法を示し、2 番目の列は Amazon Keyspaces が内部で名前をフォーマットする方法を示しています。Amazon Keyspaces は、 のようなオペレーションにフォーマットされた名前を想定していますGetType。

入力した名前	フォーマットされた名前	メモ
MY_UDT	my_udt	二重引用符を使用しない場合、Amazon Keyspaces はすべての大文字を小文字に変換します。
"MY_UDT"	MY_UDT	二重引用符を使用すると、Amazon Keyspaces は大文字を尊重し、フォーマットされた名前から二重引用符を削除します。

入力した名前	フォーマットされた名前	メモ
"1234"	1234	二重引用符を使用すると、名前は数字で始まることができ、Amazon Keyspaces はフォーマットされた名前から二重引用符を削除します。
"Special_Ch@r@cters<>!!"	Special_Ch@r@cters<>!!	二重引用符では、名前に特殊文字を含めることができ、Amazon Keyspaces はフォーマットされた名前から二重引用符を削除します。
"nested""""quotes"	nested"""quotes	Amazon Keyspaces は、フォーマットされた名前から外部二重引用符とエスケープ二重引用符を削除します。

例

```
CREATE TYPE my_keyspace.phone (
    country_code int,
    number text
);
```

ネストされた UDTs がフリーズしている場合は、UDT をネストできます。タイプのデフォルト値とクォータの詳細については、「」を参照してくださいthe section called “Amazon Keyspaces UDT クォータとデフォルト値”。

```
CREATE TYPE my_keyspace.user (
    first_name text,
    last_name text,
    phones FROZEN<phone>
);
```

UDTs「」を参照してくださいthe section called “ユーザー定義タイプ (UDTs)”。

DROP TYPE

DROP TYPE ステートメントを使用して UDT を削除します。削除できるのは、別のタイプまたはテーブルで使用されていないタイプのみです。

[Syntax] (構文)

```
drop_type_statement ::=  DROP TYPE [ IF EXISTS ] udt_name
```

Where:

- IF EXISTS タイプが存在しない場合DROP TYPE、 は失敗しません。(オプション)
- *udt_name* は、タイプ形式の UDT の完全修飾名ですmy_keyspace.my_type。例: 。USE ステートメントで現在のキースペースを定義する場合、キースペース名を指定する必要はありません。

例

```
DROP TYPE udt_name;
```

Amazon Keyspaces の DML ステートメント (データ操作言語)

データ操作言語 (DML) は、Amazon Keyspaces (Apache Cassandra 向け) テーブル内でのデータ管理に使用する Cassandra Query Language (CQL) ステートメントのセットです。DML ステートメントを使用して、テーブル内のデータを追加、変更、または削除します。

また、DML ステートメントを使用して、テーブル内のデータをクエリすることもできます。(CQL は結合とサブクエリに対応していないので注意が必要です。)

トピック

- SELECT
- INSERT
- UPDATE
- DELETE

SELECT

SELECT ステートメントを使用してデータのクエリを行います。

[Syntax] (構文)

```
select_statement ::=  SELECT  [ JSON ] ( select_clause | '*' )
                      FROM table_name
```

```
                    [ WHERE 'where_clause' ]
                    [ ORDER BY 'ordering_clause' ]
                    [ LIMIT (integer | bind_marker) ]
                    [ ALLOW FILTERING ]
select_clause    ::=  selector [ AS identifier ] ( ',' selector [ AS identifier ] )
selector         ::=  column_name
                    | term
                    | CAST '(' selector AS cql_type ')'
                    | function_name '(' [ selector ( ',' selector )* ] ')'
where_clause     ::=  relation ( AND relation )*
relation         ::=  column_name operator term
                    TOKEN
operator         ::=  '=' | '<' | '>' | '<=' | '>=' | IN | CONTAINS | CONTAINS KEY
ordering_clause  ::=  column_name [ ASC | DESC ] ( ',' column_name [ ASC | DESC ] )*
```

例

```
SELECT name, id, manager_id FROM "myGSGKeyspace".employees_tbl ;

SELECT JSON name, id, manager_id FROM "myGSGKeyspace".employees_tbl ;
```

エンコードされた JSON データ型が Amazon Keyspaces のデータ型にマッピングされるテーブルについては、「the section called “Amazon Keyspaces データ型の JSON エンコード”」を参照してください。

IN キーワードの使用

IN キーワードは、1 つ以上の値が等しいことを指定します。パーティションキーとクラスタリング列に適用できます。結果は SELECT ステートメントにキーが示されている順序で返されます。

例

```
SELECT * from mykeyspace.mytable WHERE primary.key1 IN (1,2) and clustering.key1 = 2;
SELECT * from mykeyspace.mytable WHERE primary.key1 IN (1,2) and clustering.key1 <= 2;
SELECT * from mykeyspace.mytable WHERE primary.key1 = 1 and clustering.key1 IN (1, 2);
SELECT * from mykeyspace.mytable WHERE primary.key1 <= 2 and clustering.key1 IN (1, 2)
 ALLOW FILTERING;
```

IN キーワードと Amazon Keyspaces がステートメントを処理する方法の詳細については、the section called “IN SELECT の使用” を参照してください。

結果の順序付け

ORDER BY 句は、返された結果のソート順序を指定するものです。列名のリストと各列のソート順序を引数として受け取ります。順序句のクラスタリング列のみ指定できます。非クラスタリング列は許容されません。ソート順序のオプションは、昇順の ASC と降順の DESC です。ソート順序を省略すると、クラスタリング列のデフォルトの順序が使用されます。可能なソート順序については、「the section called “結果の並べ替え”」を参照してください。

例

```
SELECT name, id, division, manager_id FROM "myGSGKeyspace".employees_tbl WHERE id =
 '012-34-5678' ORDER BY division;
```

IN キーワードと一緒に ORDER BY を使用すると、結果はページ内で順序付けられます。ページ分割を無効にした状態での完全な並べ替えはサポートしていません。

トークン

TOKEN関数は、SELECT 節と WHERE 節の PARTITION KEY 列に適用できます。TOKEN 関数を使用すると、Amazon Keyspaces は PARTITION KEY の値ではなくの PARTITION_KEY マッピングされたトークン値に基づいて行を返します。

TOKEN キーワードではリレーションはサポートしていません。IN

例

```
SELECT TOKEN(id) from my_table;

SELECT TOKEN(id) from my_table WHERE TOKEN(id) > 100 and TOKEN(id) < 10000;
```

TTL 関数

TTL 関数を SELECT ステートメントと共に使用すると、列に保存されている有効期限を秒単位で取得できます。どの TTL 値も設定されていない場合、関数は null を返します。

例

```
SELECT TTL(my_column) from my_table;
```

TTL 関数は、コレクションなどのマルチセル列には使用できません。

WRITETIME 関数

SELECT ステートメントで WRITETIME 関数を使用して、列の値のメタデータとして保存されているタイムスタンプを取得できるのは、テーブルがクライアント側のタイムスタンプを使用している場合だけです。詳細については、「the section called “クライアントサイドのタイムスタンプ”」を参照してください。

```
SELECT WRITETIME(my_column) from my_table;
```

WRITETIME 関数は、コレクションなどのマルチセル列には使用できません。

> **Note**
>
> 確立されている Cassandra ドライバーの動作との互換性を保つため、Cassandra ドライバーや開発者ツールから Cassandra クエリーランゲージ (CQL) API コールでシステムテーブルを操作するとき、タグベースの承認ポリシーは適用されません。詳細については、「the section called “ タグに基いた Amazon Keyspaces リソースアクセス”」を参照してください。

INSERT

INSERT ステートメントを使用してテーブルに行を追加します。

[Syntax] (構文)

```
insert_statement ::=  INSERT INTO table_name ( names_values | json_clause )
                      [ IF NOT EXISTS ]
                      [ USING update_parameter ( AND update_parameter )* ]
names_values     ::=  names VALUES tuple_literal
json_clause      ::=  JSON string [ DEFAULT ( NULL | UNSET ) ]
names            ::=  '(' column_name ( ',' column_name )* ')'
```

例

```
INSERT INTO "myGSGKeyspace".employees_tbl (id, name, project, region, division, role,
 pay_scale, vacation_hrs, manager_id)
VALUES ('012-34-5678','Russ','NightFlight','US','Engineering','IC',3,12.5,
 '234-56-7890') ;
```

パラメータを更新する

INSERT は以下の値を update_parameter としてサポートします。

- TTL — 秒単位の時間値。設定可能な最大値は 630,720,000 秒で、20 年に相当します。
- TIMESTAMP — 標準基準時間 epoch 1970 年 1 月 1 日 00:00:00 GMT からのマイクロ秒数を表す bigint 値。Amazon Keyspaces タイムスタンプは、過去 2 日間と将来 5 分の範囲とします。

例

```
INSERT INTO my_table (userid, time, subject, body, user)
        VALUES (B79CB3BA-745E-5D9A-8903-4A02327A7E09, 96a29100-5e25-11ec-90d7-
b5d91eceda0a, 'Message', 'Hello','205.212.123.123')
        USING TTL 259200;
```

JSON サポート

エンコードされた JSON データ型が Amazon Keyspaces のデータ型にマッピングされるテーブルについては、「the section called “Amazon Keyspaces データ型の JSON エンコード”」を参照してください。

JSON キーワードを使用すれば、エンコードされた JSON マップを 1 行として挿入できます。テーブル内に存在しても、JSON insert ステートメントでは省略されている列については、DEFAULT UNSET を使用して既存の値を保持します。DEFAULT NULL を使用して、省略された列の各行に NULL 値を書き込み、既存の値をオーバーライドします (標準の書き込み料金が適用されます)。DEFAULT NULL はデフォルトのオプションです。

例

```
INSERT INTO "myGSGKeyspace".employees_tbl JSON '{"id":"012-34-5678",
                                                 "name": "Russ",
                                                 "project": "NightFlight",
                                                 "region": "US",
                                                 "division": "Engineering",
                                                 "role": "IC",
                                                 "pay_scale": 3,
                                                 "vacation_hrs": 12.5,
                                                 "manager_id": "234-56-7890"}';
```

JSON データに重複するキーが含まれている場合、Amazon Keyspaces ではキーの最後の値が保存されます (Apache Cassandra と同様)。以下の例で、重複キーは id であり、値 234-56-7890 を使用します。

例

```
INSERT INTO "myGSGKeyspace".employees_tbl JSON '{"id":"012-34-5678",
                                                 "name": "Russ",
                                                 "project": "NightFlight",
                                                 "region": "US",
                                                 "division": "Engineering",
                                                 "role": "IC",
                                                 "pay_scale": 3,
                                                 "vacation_hrs": 12.5,
                                                 "id": "234-56-7890"}';
```

UPDATE

UPDATE ステートメントを使用して、テーブル内の行を変更します。

[Syntax (構文)]

```
update_statement ::=  UPDATE table_name
                      [ USING update_parameter ( AND update_parameter )* ]
                      SET assignment ( ',' assignment )*
                      WHERE where_clause
                      [ IF ( EXISTS | condition ( AND condition )*) ]
update_parameter ::=  ( integer | bind_marker )
assignment       ::=  simple_selection '=' term
                     | column_name '=' column_name ( '+' | '-' ) term
                     | column_name '=' list_literal '+' column_name
simple_selection ::=  column_name
                     | column_name '[' term ']'
                     | column_name '.' `field_name
condition        ::=  simple_selection operator term
```

例

```
UPDATE "myGSGKeyspace".employees_tbl SET pay_scale = 5 WHERE id = '567-89-0123' AND
 division = 'Marketing' ;
```

counter を増やすには、次の構文を使用します。詳細については、「the section called “カウンタ”」を参照してください。

```
UPDATE ActiveUsers SET counter = counter + 1  WHERE user = A70FE1C0-5408-4AE3-
BE34-8733E5K09F14 AND action = 'click';
```

パラメータを更新する

UPDATE は以下の値を update_parameter としてサポートします。

- TTL — 秒単位の時間値。設定可能な最大値は 630,720,000 秒で、20 年に相当します。
- TIMESTAMP — 標準基準時間 epoch 1970 年 1 月 1 日 00:00:00 GMT からのマイクロ秒数を表す bigint 値。Amazon Keyspaces タイムスタンプは、過去 2 日間と将来 5 分の範囲とします。

例

```
UPDATE my_table (userid, time, subject, body, user)
        VALUES (B79CB3BA-745E-5D9A-8903-4A02327A7E09, 96a29100-5e25-11ec-90d7-
b5d91eceda0a, 'Message', 'Hello again','205.212.123.123')
        USING TIMESTAMP '2022-11-03 13:30:54+0400';
```

DELETE

DELETE ステートメントを使用してテーブルから行を削除します。

[Syntax (構文)]

```
delete_statement ::=  DELETE [ simple_selection ( ',' simple_selection ) ]
                      FROM table_name
                      [ USING update_parameter ( AND update_parameter )* ]
                      WHERE where_clause
                      [ IF ( EXISTS | condition ( AND condition )*) ]

simple_selection ::=  column_name
                     | column_name '[' term ']'
                     | column_name '.' `field_name

condition        ::=  simple_selection operator term
```

Where:

- *table_name* は削除する行が含まれているテーブルです。

例

```
DELETE manager_id FROM "myGSGKeyspace".employees_tbl WHERE id='789-01-2345' AND
 division='Executive' ;
```

DELETE は、以下の値を update_parameter のようにサポートします。

- TIMESTAMP — 標準基準時間 epoch 1970 年 1 月 1 日 00:00:00 GMT からのマイクロ秒数を表す bigint 値。

Amazon Keyspaces の組み込み関数

Amazon Keyspaces (Apache Cassandra 向け) では、Cassandra クエリ言語 (CQL) ステートメントで使用できるさまざまな組み込み関数がサポートされています。

トピック

- スカラー関数

スカラー関数

スカラー関数は、1 つの値に対して計算を実行し、その結果を 1 つの値として返す関数です。Amazon Keyspaces では以下のスカラー関数がサポートされています。

関数	説明
blobAsType	指定されたデータ型の値を返します。
cast	あるネイティブデータ型を別のネイティブデータ型に変換します。
currentDate	現在の日時を日付として返します。
currentTime	現在の日時を時刻として返します。
currentTimestamp	現在の日時をタイムスタンプとして返します。
currentTimeUUID	現在の日時を timeuuid として返します。

関数	説明
fromJson	JSON 文字列を選択した列のデータ型に変換します。
maxTimeuuid	タイムスタンプまたは日付文字列として可能な最大 timeuuid を返します。
minTimeuuid	タイムスタンプまたは日付文字列の最小 timeuuid を返します。
now	新しい一意の timeuuid を返します。INSERT、UPDATE、および DELETE ステートメントでサポートし、WHEREステートメントの句の一部としてもサポートしています。SELECT
toDate	timeuuid またはタイムスタンプのいずれかを日付型に変換します。
toJson	選択した列の列値を JSON 形式で返します。
token	パーティションキーのハッシュ値を返します。
toTimestamp	timeuuid または日付のいずれかをタイムスタンプに変換します。
TTL	列の有効期限を秒単位で返します。
typeAsBlob	指定されたデータ型を blob に変換します。
toUnixTimestamp	timeuuid またはタイムスタンプのいずれかを bigInt に変換します。
uuid	ランダムバージョン 4 UUID を返します。INSERT、UPDATE、および DELETE ステートメントでサポートし、WHEREステートメントの句の一部としてもサポートしています。SELECT

関数	説明
writetime	指定した列の値のタイムスタンプを返します。
dateOf	(非推奨) timeuuid のタイムスタンプを抽出し、値を日付として返します。
unixTimestampOf	(非推奨) timeuuid のタイムスタンプを抽出し、値を生の 64 ビット整数タイムスタンプとして返します。

Amazon Keyspaces (Apache Cassandra 向け) のクォータ

このセクションでは、Amazon Keyspaces (Apache Cassandra 向け) の現在のクォータとデフォルト値について説明します。

トピック

- Amazon Keyspaces サービスクォータ
- スループットの増加または減少 (プロビジョニングされたテーブルの場合)
- Amazon Keyspaces の保管データ暗号化
- Amazon Keyspaces のユーザー定義タイプ (UDTsクォータとデフォルト値

Amazon Keyspaces サービスクォータ

次のテーブルには、Amazon Keyspaces (Apache Cassandra 向け) のクォータとデフォルト値が含まれています。調整できるクォータに関する情報は、Service Quotas コンソールで入手できます。クォータの増加をリクエストすることも可能です。クォータの詳細については、 にお問い合わせください AWS サポート。

クォータ	説明	Amazon Keyspaces デフォルト
あたりの最大キースペース AWS リージョン	各リージョンにおけるこのサブスクライバーの最大キースペース数。このデフォルト値は Service Quotas コンソールで調整できます。	256
あたりの最大テーブル数 AWS リージョン	各リージョンにおけるこのサブスクライバーの全キースペースの最大テーブル数。このデフォルト値は Service Quotas コンソールで調整できます。	256

クォータ	説明	Amazon Keyspaces デフォルト
ADD REGION オペレーションを使用して復元されるデータの最大量	ADD REGION オペレーションによって同時に復元できるデータの最大サイズ。同時に復元するデータの量を増やすには、 にお問い合わせください サポート。	10 TB
最大テーブルスキーマサイズ	テーブルスキーマの最大サイズ。	350 KB
最大同時 DDL オペレーション数	各リージョンにおいてこのサブスクライバーに対して許可される同時実行 DDL オペレーションの最大数。	50
接続別の最大クエリ数	単一クライアント TCP 接続により処理できる 1 秒あたりの最大 CQL クエリ数。	3000
最大行サイズ	静的列データを除く最大行サイズ。詳細については、「the section called “行サイズの推定”」を参照してください。	1 MB
INSERT ステートメントと UPDATE ステートメントの最大列数	CQL INSERT ステートメントまたは UPDATE ステートメントで許可される最大列数。存続可能時間 (TTL) がオフになっている場合、INSERT ステートメントまたは UPDATE ステートメントは最大 225 列の標準列をサポートします。TTL をオンにすると、1 回の操作で最大 166 列の標準列を変更できます。	225/166

クォータ	説明	Amazon Keyspaces デフォルト
論理パーティションあたりの最大静的データ	論理パーティション内の静的データの最大集計サイズ。詳細については、「the section called “各論理パーティションの静的列サイズの計算”」を参照してください。	1 MB
IN SELECT ステートメントあたりのサブクエリの最大数	SELECT ステートメントの IN キーワードに使用できるサブクエリの最大数。このデフォルト値は Service Quotas コンソールで調整できます。	100
あたりのネストされたフリーズコレクションの最大数 AWS リージョン	コレクションデータ型の列に FROZEN キーワードを使用する場合にサポートされるネストされたコレクションの最大数。フリーズコレクションの詳細については、「the section called “コレクション型”」を参照してください。ネストレベルを上げるには、 にお問い合わせください サポート。	8
1 秒あたりの最大読み取りスループット	各リージョンのテーブル 1 つに割り当てられる 1 秒あたりの最大読み取りスループット (読み取りリクエストユニット (RRU) または読み取りキャパシティユニット (RCU))。このデフォルト値は Service Quotas コンソールで調整できます。	40,000

クォータ	説明	Amazon Keyspaces デフォルト
1 秒あたりの最大書き込みスループット	各リージョンのテーブル 1 つに割り当てられる 1 秒あたりの最大書き込みスループット (書き込みリクエストユニット (WRU) または書き込みキャパシティユニット (WCU))。このデフォルト値は Service Quotas コンソールで調整できます。	40,000
change-data-capture (CDC) を使用したテーブルの最大書き込みスループット	CDC が有効になっている Amazon Keyspaces テーブルの最大書き込み容量。このデフォルトのクォータは、プロビジョニングされた読み取り/書き込みキャパシティモードのテーブルにのみ適用されます。このデフォルト値は Service Quotas コンソールで調整できます。	40,000
アカウントレベルの読み取りスループット (プロビジョニング済み)	リージョンごとにアカウントに割り当てられる合計読み取りキャパシティユニット (RCU) の最大数。読み取り/書き込みのプロビジョンドキャパシティモードのテーブルにのみ適用されます。このデフォルト値は Service Quotas コンソールで調整できます。	80,000

クォータ	説明	Amazon Keyspaces デフォルト
アカウントレベルの書き込みスループット (プロビジョニング済み)	リージョンごとにアカウントに割り当てられる合計書き込みキャパシティユニット (WCU) の最大数。読み取り/書き込みのプロビジョンドキャパシティモードのテーブルにのみ適用されます。このデフォルト値は Service Quotas コンソールで調整できます。	80,000
アカウントのリージョンあたりのスケーラブルターゲットの最大数	該当アカウントのリージョンあたりのスケーラブルターゲットの最大数。Amazon Keyspaces テーブルは、読み取りキャパシティに対して自動スケーリングが有効になっている場合は 1 つのスケーラブルターゲットとしてカウントされ、書き込みキャパシティに対して自動スケーリングが有効になっている場合は別のスケーラブルターゲットとしてカウントされます。このデフォルト値は、Service Quotas コンソールで Application Auto Scaling に対して [Scalable targets for Amazon Keyspaces] を選択し、調整できます。	1,500

クォータ	説明	Amazon Keyspaces デフォルト
パーティションキーの最大サイズ	複合パーティションキーの最大サイズ。メタデータのパーティションキーに含まれる各列の生サイズに、最大 3 バイトのストレージが追加されます。	2048 バイト
クラスタリングキーの最大サイズ	すべてのクラスタリング列を合わせた最大サイズ。メタデータの各クラスタリング列の生サイズに、最大 4 バイトのストレージが追加されます。	850 バイト
ポイントインタイムリカバリ (PITR) を使用した最大同時実行テーブルリストア	サブスクライバー 1 名あたりの PITR による最大同時テーブル復元数は 4 です。このデフォルト値は Service Quotas コンソールで調整できます。	4
ポイントインタイムリカバリ (PITR) を使用して復元されるデータの最大量	24 時間以内に PITR を使用して復元できるデータの最大サイズ。このデフォルト値は Service Quotas コンソールで調整できます。	5 TB

スループットの増加または減少 (プロビジョニングされたテーブルの場合)

プロビジョニングされたスループットの増加

ReadCapacityUnits または WriteCapacityUnits は、必要に応じて何回でもコンソールまたは ALTER TABLE ステートメントを使用して増やすことができます。新しい設定は、ALTER TABLE オペレーションが完了するまでは有効になりません。

プロビジョニング済みキャパシティを追加する場合、アカウントあたりのクォータを超えることはできません。また、テーブルのプロビジョニング済みキャパシティは、必要なだけ増やすことができます。アカウントごとのクォータの詳細については、前述の「the section called “Amazon Keyspaces サービスクォータ”」セクションを参照してください。

プロビジョニングされたスループットの減少

ALTER TABLE ステートメント内の各テーブルに対して ReadCapacityUnits または WriteCapacityUnits (あるいは両方) を減らすことができます。新しい設定は、ALTER TABLE オペレーションが完了するまでは有効になりません。

1 日に 4 回までいつでも減らすことができます。日付は、協定世界時 (UTC) に従って定義されます。さらに、過去 1 時間に減少されていない場合はさらに減少できます。1 日の減少の最大数は 27 になります (最初の 1 時間で 4 回、一日の残り時間で 1 時間ごとに 1 回)。

Amazon Keyspaces の保管データ暗号化

AWS 所有 AWS KMS キーとカスタマーマネージド AWS KMS キー間の暗号化オプションは、テーブルが作成された時点から、テーブルごとに 24 時間以内に最大 4 回変更できます。また、過去 6 時間以内に変更がなかった場合は、追加で変更することができます。これにより、1 日で変更できる最大の回数は 8 回になります (1 日の中で最初の 6 時間は 4 回、その後は 6 時間ごとに 1 回)。

以前のクォータを使い果たした場合でも、必要に応じて AWS 所有 AWS KMS キーを使用するように暗号化オプションを変更できます。

クォータの拡大をリクエストしない限り、以下のクォータが適用されます。サービスクォータ引き上げをリクエストするには、「サポート」を参照してください。

Amazon Keyspaces のユーザー定義タイプ (UDTsクォータとデフォルト値

Amazon Keyspaces UDT クォータとデフォルト値

次の表に、Amazon Keyspaces の UDTs に関連するクォータとデフォルト値を示します。これらのクォータの詳細については、 にお問い合わせください AWS サポート。

クォータ	説明	Amazon Keyspaces デフォルト
あたりの UDTsの最大数 AWS リージョン	リージョンあたりのこのサブスクライバーのすべてのキースペースにおける UDTs の最大数。	256
UDT あたりのテーブルの最大数	同じ UDT を参照できるテーブルの最大数。	100
テーブルあたりの UDTs の最大数	テーブルが参照できる UDTs の最大数。	50
UDTs の最大ネストレベル	UDTs でサポートされるネストの最大深度。	8
UDTsあたりの直接子 UDT の最大数	UDTsサポートされている子 UDT の最大数。	10
UDTsあたりの直接親 UDT の最大数	UDTsサポートされている親 UDT の最大数。	10
UDT スキーマの最大サイズ	UDT のスキーマの最大サイズ。	25 KB
UDT 名の最大長	UDT 名の最大文字数。	48
UDT フィールド名の最大長	UDT フィールド名の最大文字数。	128

Amazon Keyspaces (Apache Cassandra 向け) のドキュメント履歴

次の表で、Amazon Keyspaces の直近のリリース以後にドキュメントに対して行われた重要な変更について説明します。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

- ドキュメントの最終更新日： 2025 年 7 月 2 日

変更	説明	日付
Amazon Keyspaces Streams Kinesis Adapter が利用可能になりました	Kinesis Client Library (KCL) を使用して Amazon Keyspaces CDC ストリームを処理できるようになりました。	2025 年 7 月 2 日
Amazon Keyspaces の新しい管理ポリシー	Amazon Keyspaces は、ユーザーに代わって Amazon Keyspaces CDC ストリームメトリクスデータを CloudWatch に発行`AWSServiceRoleForAmazonKeyspacesCDC` するために必要なアクセス許可をサービスにリンク`KeyspacesCDCServiceRolePolicy` されたロールに付与する新しい管理ポリシーを追加しました。詳細については、「Amazon Keyspaces CDC ストリームのロールの使用」を参照してください。	2025 年 7 月 2 日
Amazon Keyspaces マネージドポリシーの更新	Amazon Keyspaces は、IAM プリンシパルがストリームを表示できるように、	2025 年 7 月 2 日

	AmazonKeyspacesReadOnlyAccess_v2 と AmazonKeyspacesReadOnlyAccess 管理ポリシーに新しいアクセス許可を追加しました。Amazon Keyspaces は、プリンシパルに代わって CloudWatch メトリクスを発行AWSServiceRoleForAmazonKeyspacesCDC するサービスにリンクされたロールを作成するアクセス許可で AmazonKeyspacesFullAccess マネージドポリシーを更新しました。	
Amazon Keyspaces 変更データキャプチャ (CDC) ストリームが利用可能になりました	CDC ストリームを有効にして、Amazon Keyspaces テーブルに変更を記録します。	2025 年 7 月 2 日
Amazon Keyspaces マルチリージョンレプリケーションで、アフリカ (ケープタウン) リージョンがサポートされるようになりました。	マルチリージョンキースペースのアフリカ (ケープタウン) リージョンを定義できるようになりました。	2025 年 3 月 31 日
Amazon Keyspaces マルチリージョンレプリケーションが無制限をサポートするようになりました AWS リージョン。	マルチリージョンキースペースに必要な数のリージョンを定義できるようになりました。	2025 年 3 月 25 日
Amazon Keyspaces がマルチリージョンキースペースのユーザー定義タイプ (UDTs) をサポート	マルチリージョンキースペースで UDTs を作成して使用できるようになりました。	2025 年 3 月 24 日

Amazon Keyspaces マネージドポリシーの更新	Amazon Keyspaces は、IAM プリンシパルが既存のキースペースに新しいリージョンを追加できるように、AmazonKeyspacesFullAccess マネージドポリシーに新しいアクセス許可を追加しました。これには、サービスにリンクされたロール AWSServiceRoleForAmazonKeyspacesReplication の更新が含まれます。	2024 年 11 月 19 日
Amazon Keyspaces のマルチリージョンテーブルにレプリカを追加する	既存の単一およびマルチリージョンのキースペース AWS リージョン に新しい を追加できるようになりました。	2024 年 11 月 19 日
Amazon Keyspaces でのユーザー定義タイプ (UDTs) のサポート	Amazon Keyspaces での UDTs のネイティブサポートにより、実際のデータ階層を表すアプリケーション内のデータ構造を定義できるようになりました。	2024 年 10 月 30 日
ADD COLUMN Amazon Keyspaces マルチリージョンレプリケーションのサポート	Amazon Keyspaces がマルチリージョンテーブルのスキーマ変更に対応しました。	2024 年 9 月 17 日
Amazon Keyspaces の移行ガイダンスの改訂	改訂後の移行ガイダンスでは、オフライン移行とオンライン移行のさまざまな戦略を含め、Apache Cassandra から Amazon Keyspaces に正常に移行するための移行計画の立て方を概説しています。	2024 年 6 月 28 日

Amazon Elastic Kubernetes Service から Amazon Keyspaces への接続	ステップバイステップのチュートリアルに従って、Amazon EKS から Amazon Keyspaces に接続できるようになりました。	2024 年 2 月 7 日
プロビジョニングされたテーブルの Amazon Keyspaces マルチリージョンレプリケーションのサポート	Amazon Keyspaces で、マルチリージョンテーブルがプロビジョンドキャパシティモードに対応しました。	2024 年 1 月 23 日
プロビジョンドテーブル用の Amazon Keyspaces 自動スケーリング API	Amazon Keyspaces では、プロビジョニングされたキャパシティモードで自動スケーリングを設定するための CQLおよび AWS API サポートが提供されるようになりました。	2024 年 1 月 23 日
CloudTrail ログに含まれる Amazon Keyspaces DML アクティビティ	AWS CloudTrailで Amazon Keyspaces データ操作言語 (DML) API コールを監査できるようになりました。	2023 年 12 月 20 日
Amazon Keyspaces による FROZEN キーワードのサポート	Amazon Keyspaces は、コレクションデータ型の FROZEN キーワードをサポートするようになりました。	2023 年 11 月 15 日

Amazon Keyspaces マネージドポリシーの更新	Amazon Keyspaces は AmazonKeyspacesFullAccess マネージドポリシーに新しいアクセス権限を追加し、インターフェイス VPC エンドポイントを介して Amazon Keyspaces に接続するクライアントが Amazon EC2 インスタンスにアクセスし、VPC からのネットワーク情報で Amazon Keyspaces system.peers テーブルを更新できるようにしました。	2023 年 10 月 3 日
Amazon Keyspaces マネージドポリシーの更新	Amazon Keyspaces は、インターフェイス VPC エンドポイントを介して Amazon Keyspaces に接続するクライアントが Amazon EC2 インスタンスにアクセスし、VPC からのネットワーク情報で Amazon Keyspaces system.peers テーブルを更新できるようにする新しい AmazonKeyspacesReadOnlyAccess_v2 マネージドポリシーを作成しました。	2023 年 9 月 12 日
Amazon Keyspaces で接続を作成するためのベストプラクティス	ここでは、Amazon Keyspaces のクライアントドライバー設定を改善し、最適化する方法を説明します。	2023 年 6 月 30 日

Amazon Keyspaces のシステムキー空間を文書化しました	ここでは、システムキー空間に何が保存されているか、そして Amazon Keyspaces で役立つ情報を取得するためにそれらをクエリする方法を説明します。	2023 年 6 月 21 日
Amazon Keyspaces がマルチリージョンレプリケーションをサポートするようになりました	Amazon Keyspaces マルチリージョンレプリケーションは、耐障害性、安定性、耐障害性を向上させることで、グローバルに分散されたアプリケーションを維持するのに役立ちます。	2023 年 6 月 5 日
Amazon Keyspaces マネージドポリシーの更新	管理者がマルチリージョンキー空間を作成すると、Amazon Keyspaces でサービスにリンクされたロールが自動的に作成できる新しいアクセス権限が、Amazon Keyspaces で `AmazonKeyspacesFullAccess` マネージドポリシーに追加されました。	2023 年 6 月 5 日
Amazon Keyspaces による IN キーワードのサポート	Amazon Keyspaces では、SELECT ステートメント内の IN キーワードをサポートするようになりました。	2023 年 4 月 25 日
Amazon Keyspaces とインターフェイス VPC エンドポイントへのクロスアカウントアクセス	ここでは、VPC エンドポイントで Amazon Keyspaces にクロスアカウントアクセスを実装する方法を説明します。	2023 年 4 月 20 日

Amazon Keyspaces によるクライアントサイドのタイムスタンプのサポート	Amazon Keyspaces のクライアントサイドのタイムスタンプは、Cassandra と互換性のあるセルレベルのタイムスタンプです。このタイムスタンプは、さまざまなクライアントによって同じデータが変更されたときに、分散アプリケーションによって書き込み操作の順序を決定する際に役立ちます。	2023 年 3 月 14 日
Amazon Keyspaces と VPC エンドポイントの使用開始	このステップバイステップのチュートリアルでは、VPC から Amazon Keyspaces に接続する方法を説明します。	2023 年 3 月 1 日
Amazon Keyspaces テーブルのコストの最適化	既存の Amazon Keyspaces テーブルのコスト最適化戦略の策定に役立つベストプラクティスとガイダンスを利用できます。	2023 年 2 月 17 日
`Murmur3Partitioner` がデフォルトになりました	`Murmur3Partitioner` が Amazon Keyspaces でデフォルトのパーティショナーになりました。	2022 年 11 月 17 日
Amazon Keyspaces `Murmur3Partitioner` でサポートされるようになりました	Amazon Keyspaces で `Murmur3Partitioner` を利用できます。	2022 年 11 月 9 日

空の文字列と BLOB 値のサポートの更新	Amazon Keyspaces では、クラスターリング列値として、空の文字列と BLOB 値をサポートするようになりました。	2022 年 10 月 19 日
Amazon Keyspaces が で利用可能になりました AWS GovCloud (US)	Amazon Keyspaces が で利用可能になり AWS GovCloud (US) Region 、FedRAMP-High コンプライアンスの対象になりました。利用可能なエンドポイントの詳細については、「AWS GovCloud (US) Region FIPS エンドポイント」を参照してください。	2022 年 8 月 4 日
Amazon CloudWatch で Amazon Keyspaces テーブルストレージコストをモニタリングする	Amazon Keyspaces では、`BillableTableSizeInBytes` CloudWatch メトリクスを使用して、テーブルストレージのコストを経時的にモニタリングし、追跡できるようになりました。	2022 年 6 月 14 日
Amazon Keyspaces が Terraform をサポートするようになりました	Terraform を使用して、Amazon Keyspaces でデータ定義言語 (DDL) オペレーションを実行できるようになりました。	2022 年 6 月 9 日
Amazon Keyspaces `token` 関数のサポート	Amazon Keyspaces では、`token` 関数でアプリケーションクエリを最適化できるようになりました。	2022 年 4 月 19 日

Amazon Keyspaces と Apache Spark の統合	Amazon Keyspaces では、オープンソースの Spark Cassandra コネクタの使用により、Apache Spark で簡単にデータを読み書きできるようになりました。	2022 年 4 月 19 日
Amazon Keyspaces API リファレンス	Amazon Keyspaces は、 AWS SDK と を使用してキースペースとテーブルを管理するコントロールプレーンオペレーションをサポートしています AWS CLI。API リファレンスガイドでは、サポート対象のコントロールプレーンのオペレーションについて詳しく説明します。	2022 年 3 月 2 日
Amazon Keyspaces を使用する際の一般的な設定問題に対するトラブルシューティング方法。	Amazon Keyspaces の使用時に発生する可能性のある一般的な設定問題を解決する方法が説明されています。	2021 年 11 月 22 日
Amazon Keyspaces による有効期限 (TTL) のサポート。	Amazon Keyspaces の有効期限 (TTL) を使用すると、テーブルのデータを自動的に期限切れにして、アプリケーションロジックを簡素化し、ストレージの料金を最適化できます。	2021 年 10 月 18 日
DSBulk を使用した Amazon Keyspaces へのデータ移行。	DataStax Bulk Loader (DSBulk) を使用して Apache Cassandra から Amazon Keyspaces にデータを移行するためのステップバイステップのチュートリアル。	2021 年 8 月 9 日

Amazon Keyspaces では、`system.peers` テーブルの VPC エンドポイントエントリがサポートされています。	Amazon Keyspaces では、ロードバランシングを改善し、読み取り/書き込みスループットを向上させるために、`system.peers` テーブルに、使用可能なインターフェイス VPC エンドポイント情報を入力できます。	2021 年 7 月 29 日
IAM 管理ポリシーを更新して、カスタマー管理 AWS KMS キーをサポートします。	Amazon Keyspaces の IAM 管理ポリシーに、 に保存されている利用可能なカスタマー管理 AWS KMS キーを一覧表示および表示するアクセス許可が含まれるようになりました AWS KMS。	2021 年 6 月 1 日
カスタマーマネージド AWS KMS キーの Amazon Keyspaces サポート。	Amazon Keyspaces では、保管時の暗号化 AWS KMS のために に保存されているカスタマーマネージド AWS KMS キーを制御できます。	2021 年 6 月 1 日
Amazon Keyspaces による JSON 構文のサポート	Amazon Keyspaces では、INSERT と SELECT のオペレーションに関して JSON 構文がサポートされることで、JSON ドキュメントの読み取りと書き込みがより簡単になります。	2021 年 1 月 21 日
Amazon Keyspaces による静的列のサポート	Amazon Keyspaces では、静的列を使用することで、複数の行の間で共通データの更新と保存を効率的に実行できるようになりました。	2020 年 11 月 9 日

Amazon Keyspaces による NoSQL Workbench のサポートの GA リリース	NoSQL Workbench は、Amazon Keyspaces の非リレーショナルデータモデルの設計と可視化をより簡単にするクライアントサイドアプリケーションです。NoSQL Workbench クライアントは、Windows、macOS、Linux で使用できます。	2020 年 10 月 28 日
Amazon Keyspaces による NoSQL Workbench のサポートのプレビューリリース	NoSQL Workbench は、Amazon Keyspaces の非リレーショナルデータモデルの設計と可視化をより簡単にするクライアントサイドアプリケーションです。NoSQL Workbench クライアントは、Windows、macOS、Linux で使用できます。	2020 年 10 月 5 日
Amazon Keyspaces へのプログラムによるアクセスのための新たなコード例	Amazon Keyspaces へのプログラムによるアクセスのためのコード例を引き続き追加しています。Apache Cassandra バージョン 3.11.2 がサポートされている Java、Python、Go、C#、および Perl Cassandra ドライバーで、サンプルを利用できるようになりました。	2020 年 7 月 17 日

Amazon Keyspaces ポイントインタイムリカバリ (PITR)	Amazon Keyspaces では、ポイントインタイムリカバリ (PITR) により、テーブルデータの継続的なバックアップが可能になり、テーブルの誤った書き込みオペレーションや削除オペレーションを防止できます。	2020 年 7 月 9 日
Amazon Keyspaces の一般向け販売	以前はプレビュー時に Amazon Managed Apache Cassandra Service (MCS) と呼ばれていた Amazon Keyspaces では、Cassandra クエリ言語 (CQL) コード、Apache 2.0 ライセンスの Cassandra ドライバー、現在すでに使用しているデベロッパーツールを使用できます。	2020 年 4 月 23 日
Amazon Keyspaces オートスケーリング	Amazon Keyspaces (Apache Cassandra 向け) をアプリケーションオートスケーリングと統合することで、スループットキャパシティの自動調整が可能になり、実際のアプリケーショントラフィックに対応する可変ワークロードのスループットキャパシティを効率的にプロビジョニングすることができます。	2020 年 4 月 23 日

Amazon Keyspaces のインターフェイス仮想プライベートクラウド (VPC) エンドポイント	Amazon Keyspaces は、ネットワークトラフィックが Amazon ネットワークから離れないように、サービスと VPC 間のプライベート通信を提供します。	2020 年 4 月 16 日
タグベースのアクセスポリシー	IAM ポリシーでリソースタグを使用して Amazon Keyspaces へのアクセスを管理できるようになりました。	2020 年 4 月 8 日
カウンターデータ型	Amazon Keyspaces では、カウンターを使用して、列の値の増減を調整できるようになりました。	2020 年 4 月 7 日
リソースのタグ付け	Amazon Keyspaces では、タグを使用してリソースのラベル付けと分類が可能になりました。	2020 年 3 月 31 日
AWS CloudFormation のサポート	Amazon Keyspaces では、AWS CloudFormationの使用によりリソースの作成と管理を自動化できるようになりました。	2020 年 3 月 25 日

IAM ロール/ポリシーおよび SigV4 認証のサポート	AWS Identity and Access Management (IAM) を使用したアクセス許可の管理と Amazon Keyspaces のセキュリティポリシーの実装、および、DataStax Java Driver for Cassandra の認証プラグインを使用した Amazon Keyspaces (IAM ロールとフェデレーションアイデンティティを使用) へのプログラムアクセスに関する情報が追加されました。	2020 年 3 月 17 日
読み取り/書き込みキャパシティモード	Amazon Keyspaces で 2 つの読み取り/書き込みスループットキャパシティモードをサポートするようになりました。読み取り/書き込みキャパシティモードで、読み取りと書き込みのスループットの料金体系とテーブルスループットキャパシティの管理方法を制御します。	2020 年 2 月 20 日
初回リリース	このドキュメントでは、Amazon Keyspaces (Apache Cassandra 向け) の初回リリースについて説明します。	2019 年 12 月 3 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。