aws

開発者ガイド

Amazon Keyspaces (Apache Cassandra 向け)

Amazon Keyspaces (Apache Cassandra 向け): 開発者ガイド

Table of Contents

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を数回クリックするか、数行のコードを入力するだけで、Amazon Keyspaces とテーブルを作成できます。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必要はありません。

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

AWS リージョン 利用可能な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。

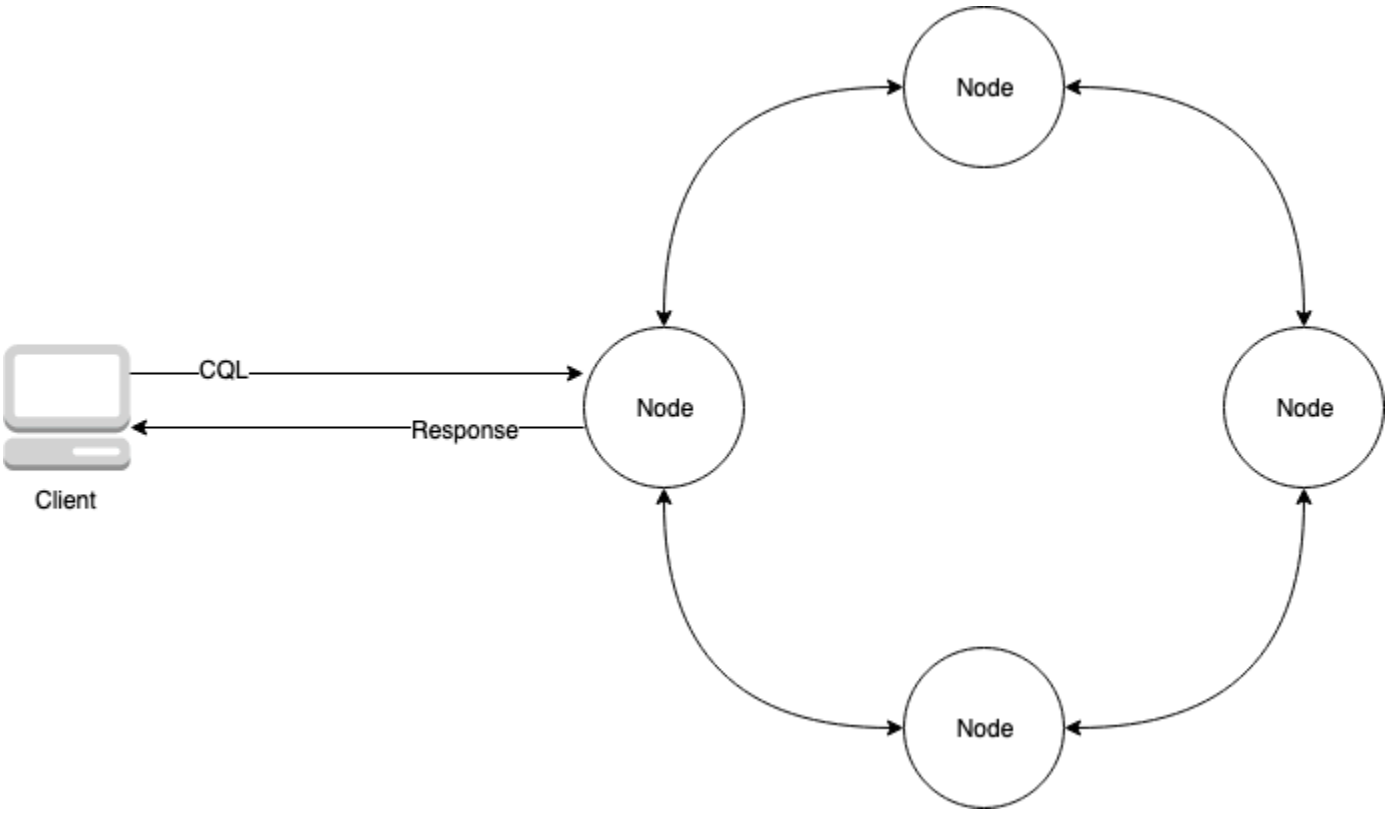
トピック

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

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。

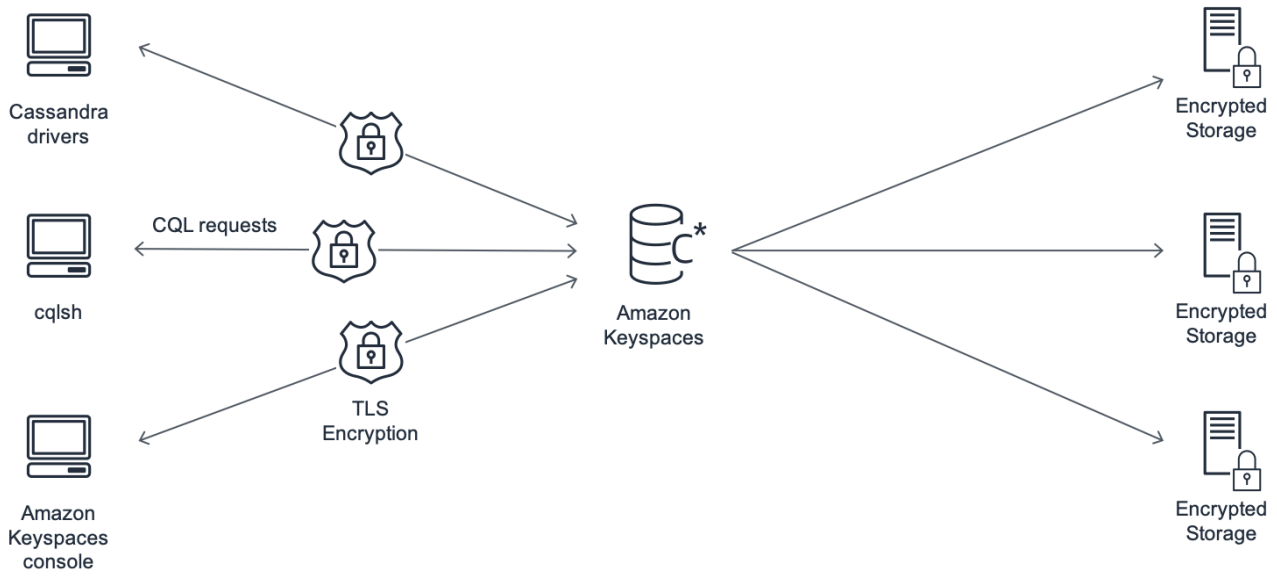
プロビジョンドキャパシティモードでは、予測可能なアプリケーショントラフィックがあり、テーブルのキャパシティ要件を事前に予測できる場合に、スループットの料金を最適化できます。プロビジョンドキャパシティモードを使用する場合、アプリケーションに必要な 1 秒あたりの読み取り回

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”」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。 AWS その他のセキュリティ機能には、監視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。そのためには、すべての Cassandra テーブルには、テーブルの各行のキーであ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、を参照してください。the section called “NoSQL 設計”Amazon Keyspaces とデータモデリングの詳細については、を参照してくださいthe section called “データモデリング”。

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。必要な認証情報について詳しくは、「」を参照してくださいthe section called “認証用の IAM 認証情報 AWS ”。

> **Note**
>
> すぐに始められるように、さまざまな Cassandra クライアントドライバーを使用して Amazon end-to-end キースペースに接続するコードサンプルが、の Amazon Keyspaces コードサンプルリポジトリにあります。GitHub

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の構築 — 産業機器のメンテナンス、取引監視、車両管理、ルート最適化など、 single-digit-millisecondレイテンシーを必要とするアプリケーションの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たアプリケーションの構築 — Java、Python、Ruby、Microsoft .NET、Node.js、PHP、C++、Perl、Goなど、さまざまなプログラミング言語で使用できるオープンソースのCassandra AWS APIとドライバーを使用して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しなくても、Cassandraテーブルをセットアップ、保護、スケーリングできます。 AWS クラウド詳細については、「サーバーレスリソース管理」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 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 エディタを使用します。
- 使用と cqlsh による拡張。 AWS CloudShell
- このクエリを `cqlsh` クライアントで実行します。
- Apache 2.0 ライセンス付きの Cassandra クライアントドライバーを使用してプログラムによりこのクエリを実行します。

さらに、 AWS SDK とを使用して Amazon Keyspaces にアクセスできます。 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。

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。

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、AWS パブリックサービスエンドポイントを使用するか、Amazon 仮想プライベートクラウドのインターフェイス VPC エンドポイント (AWS PrivateLink) を使用するプライベートエンドポイントを使用できます。使用するエンドポイントに応じて、Amazon Keyspaces は以下のいずれかの方法でクライアントに表示されます。

AWS サービスエンドポイント接続

> これは任意のパブリックエンドポイント上で確立される接続です。この場合、Amazon Keyspaces はクライアントには 9 ノードの Apache Cassandra 3.11.2 クラスターとして表示されます。

インターフェイス VPC エンドポイント接続

> これは、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 3 ノードの Apache Cassandra 3.11.2 クラスターとしてクライアントに表示されます。

接続タイプやクライアントに表示されるノードの数に関係なく、Amazon Keyspaces は事実上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、ロードバランサーはクエリを基盤となる多くのストレージパーティションのいずれかに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 はデータをパーティションに保存します。パーティションは、ソリッドステートドライブ (SSD) を基盤とするテーブル用のストレージ割り当てです。Amazon Keyspaces は、耐久性と高可用性を実現するために、 AWS リージョン 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複製します。スループットやストレージのニーズが高まると、Amazon Keyspaces がお客様に代わってパーティション管理を行い、必要な追加パーティションを自動的にプロビジョニングします。詳細については、「the section called “[Storage (ストレージ)]”」を参照してください。

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 はサーバーレスなので、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。そのため、Amazon

Keyspaces では、Cassandra コントロールプレーン API オペレーションを使用してクラスターとノードの設定を管理する必要がありません。

Amazon Keyspaces は、高い可用性、耐久性、パフォーマンスを提供するために、レプリケーション係数や整合性レベルなどの設定を自動的に設定します。 single-digit-millisecond Amazon Keyspaces では耐障害性をさらに高め、ローカル読み取りのレイテンシーを低く抑えるために、マルチリージョンレプリケーションを提供しています。

トピック

- 機能の違い: Amazon Keyspaces と Apache Cassandra
- Amazon Keyspaces で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 でサポートされている Apache Cassandra の整合性レベル

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
- クラスターの設定
- 接続
- IN キーワード
- CQL クエリのスループットチューニング
- FROZEN collections
- 軽量トランザクション
- 負荷分散
- 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

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、「Amazon Keyspaces で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 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) オペレーションを同期なしで実行します。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”」と「the section called “テーブルの作成”」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) は、ユーザーの認証と承認に AWS Identity and Access Management (IAM) を使用し、Apache Cassandra と同等の承認ポリシーをサポートします。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。

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。

接続

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 “接続”」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 Statement”」を参照してください。

CQL クエリのスループットチューニング

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。

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 “接続”」を参照してください。

VPC エンドポイントに接続する場合、使用できるエンドポイントは少なくなります。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 エンドポイント接続のベストプラクティスの詳細については、「」を参照してくださいthe section called “VPC エンドポイント接続”。

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のネストを最大 5 レベルまで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。

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。

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。さらに、範囲削除はアトミックですが、分離されません。

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。

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 ドライバーとデベロッパーツールを使用して 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よって、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、「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。

Amazon Keyspaces で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

API オペレーション	サポート対象
`CREATE KEYSPACE`	はい
`ALTER KEYSPACE`	はい

API オペレーション	サポート対象
DROP KEYSPACE	はい
CREATE TABLE	はい
ALTER TABLE	はい
DROP TABLE	はい
CREATE INDEX	いいえ
DROP INDEX	いいえ
UNLOGGED BATCH	はい
LOGGED BATCH	いいえ
SELECT	はい
INSERT	はい
DELETE	はい
UPDATE	はい
USE	はい
CREATE TYPE	いいえ
ALTER TYPE	いいえ
DROP TYPE	いいえ
CREATE TRIGGER	いいえ
DROP TRIGGER	いいえ
CREATE FUNCTION	いいえ
DROP FUNCTION	いいえ

API オペレーション	サポート対象
CREATE AGGREGATE	いいえ
DROP AGGREGATE	いいえ
CREATE MATERIALIZED VIEW	いいえ
ALTER MATERIALIZED VIEW	いいえ
DROP MATERIALIZED VIEW	いいえ
TRUNCATE	いいえ

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
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	はい
静的列	はい
有効期限 (TTL)	はい

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。

機能	サポート対象
`Aggregate 関数`	いいえ
`Blob 変換`	はい
`Cast`	はい
`Datetime 関数`	はい
時間変換関数	はい
`TimeUuid 関数`	はい
`Token`	はい
`User defined functions (UDF)`	いいえ
`Uuid`	はい

Cassandra データ型サポート

データタイプ	サポート対象	注意
ascii	はい	
bigint	はい	
blob	はい	
boolean	はい	
counter	はい	
date	はい	
decimal	はい	
double	はい	
float	はい	
frozen	はい	
inet	はい	
int	はい	
list	はい	
map	はい	
set	はい	
smallint	はい	
text	はい	
time	はい	
timestamp	はい	

データタイプ	サポート対象	注意
timeuuid	はい	
tinyint	はい	
tuple	はい	
user-defined types (UDT)	いいえ	UDT をプロトコルバッファでリファクタリングするには、「Amazon Keyspaces プロトコルバッファ」を参照してください。
uuid	はい	
varchar	はい	
varint	はい	

Amazon Keyspaces でサポートされている Apache Cassandra の整合性レベル

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	サポート外
QUORUM	サポート外
ALL	サポート外
TWO	サポート外

Apache Cassandra	Amazon Keyspaces
THREE	サポート外
ANY	サポート外
SERIAL	サポート外
LOCAL_SERIAL	サポート外

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 してから、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環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

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] を選んで、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、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 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 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」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、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」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「グループの参加」を参照してください。

Amazon Keyspaces のセットアップ

Amazon Keyspaces リソースへのアクセスは、IAM を使用して管理されます。IAM を使用すると、Amazon Keyspaces の特定のリソースへの読み取りおよび書き込みアクセス許可を付与するポリシーを IAM ユーザー、ロール、およびフェデレーティッド ID にアタッチできます。

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

- AmazonKeyspacesFullAccess – このポリシーは、すべての機能へのフルアクセスを持つ Amazon Keyspaces 内のすべてのリソースにアクセスするアクセス許可を付与します。
- AmazonKeyspacesReadOnlyAccess_v2 – このポリシーは、Amazon Keyspaces に読み取り専用アクセス許可を付与します。

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

IAM アイデンティティが実行できるアクションの範囲を制限したり、アイデンティティがアクセスできるリソースを制限したりするには、`AmazonKeyspacesFullAccess`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し、不要なアクセス許可をすべて削除できます。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することもできます。Amazon Keyspaces でアクションを制限したり、特定のリソースへのアクセスを制限する方法の詳細については、「」を参照してくださいthe section called “Amazon Keyspaces で IAM が機能する仕組み”。

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

- コンソールの使用
- の使用 AWS CloudShell

- プログラムによる接続

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、管理します。
- テーブルの Monitor タブで重要なテーブルメトリクスをモニタリングします。
  - 請求対象テーブルサイズ (バイト)
  - キャパシティメトリクス
- データの挿入、更新、削除など、CQL エディタを使用してクエリを実行します。
- アカウントのパーティショナー設定を変更します。
- 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します。

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。

Amazon Keyspaces AWS CloudShell へのアクセスに使用

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

AWS CloudShell から起動すると AWS Management Console、 AWS コンソールへのサインインに使用した認証情報が新しいシェルセッションで自動的に使用可能になります。 AWS CloudShell このユーザーの事前認証により、cqlsh AWS CLI またはバージョン 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されている) を使用して Amazon Keyspaces AWS などのサービスを操作するときに、認証情報を設定する必要がなくなります。

の IAM アクセス権限の取得 AWS CloudShell

が提供するアクセス管理リソースを使用して AWS Identity and Access Management、管理者は IAM AWS CloudShell ユーザーに権限を付与し、ユーザーが環境の機能にアクセスして使用できるようにすることができます。

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 管理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。 AWS 以下の管理ポリシーフォームを IAM ID CloudShell にアタッチできます。

- `AWSCloudShellFullAccess`: AWS CloudShell すべての機能にフルアクセスして使用する権限を付与します。

IAM ユーザーが実行できるアクションの範囲を制限したい場合は 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。のユーザーが利用できるアクションを制限する方法の詳細については CloudShell、ユーザーガイドの「IAM AWS CloudShell ポリシーによるアクセスと使用の管理」を参照してください。AWS CloudShell

> **Note**
>
> IAM ID には、Amazon Keyspaces を呼び出すアクセス権限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ことも、管理ポリシーをテンプレートとして使用して不要なアクセス権限を削除することもできます。特定のキースペースやテーブルへのアクセスを制限して、カスタムポリシーを作成することもできます。以下の Amazon Keyspaces 管理ポリシーを IAM アイデンティティにアタッチできます。

- AmazonKeyspacesFullAccess— このポリシーは、Amazon Keyspaces をすべての機能にフルアクセスして使用する許可を付与します。

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

Amazon Keyspaces でアクションを制限したり、特定のリソースへのアクセスを制限したりする方法の詳細については、「」を参照してくださいthe section called “Amazon Keyspaces で IAM が機能する仕組み”。

を使用して Amazon Keyspaces を操作する AWS CloudShell

AWS CloudShell から起動すると 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 とのやり取りをすぐに開始できます。をまだインストールしていない場合はcqlsh-expansion、「」the section called “cqlsh-expansionの使用” で詳細な手順を確認してください。

> **Note**
>
> cqlsh-expansionin を使用するときは AWS CloudShell、シェル内ですでに認証されているので、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 Connect し、新しいキースペースを作成します。次に、システムテーブルから読み取り、キースペースが以下を使用して作成されたことを確認します。 AWS CloudShell

1. から AWS Management Console、 CloudShell ナビゲーションバーにある以下のオプションを選択して起動できます。
   - CloudShell アイコンを選択します。
   - 検索ボックスに「cloudshell」と入力し、 CloudShellオプションを選択します。
2. 以下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* は必ず、お住まいのリージョンの正しい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その名前で新しいキースペースを作成します。そのためには、以下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、以下のコマンドを使用してシステムテーブルから読み取ることができます。

```
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)
```

プログラムによる Amazon Keyspaces への接続

このトピックでは、プログラムによる Amazon Keyspaces への接続に必要な手順の概要を説明します。IAM 認証情報の作成手順を示し、使用可能な AWS サービスエンドポイントを一覧表示します。最後のセクションでは、cqlsh を使用して Amazon Keyspaces に接続する方法を説明します。さまざまな Apache Cassandra ドライバーを使用して Amazon Keyspaces に接続する step-by-step 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”。Amazon VPC エンドポイントから Amazon Keyspaces に接続する方法を示す step-by-step チュートリアルについては、「」を参照してくださいthe section called “VPC エンドポイントとの接続”。

> **Note**
>
> 使用開始に役立つように、 の Amazon Keyspaces end-to-end コードサンプルリポジトリで、さまざまな Cassandra クライアントドライバーを使用して Amazon Keyspaces に接続するコードサンプルを見つけることができますGitHub。

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。 AWS のセットアップ手順が既に完了していることを前提としていますAmazon Keyspaces へのアクセス。

を既にお持ちの場合は AWS アカウント、以下のトピックを参照して、cqlsh を使用してプログラムで Amazon Keyspaces にアクセスする方法を確認してください。

トピック

- プログラムにより Amazon Keyspaces にアクセスするための認証情報の作成
- Amazon Keyspaces のサービスエンドポイント
- cqlsh を使用した Amazon Keyspaces への接続
- AWS CLI の使用
- API を使用する場合
- AWS SDK での Amazon Keyspaces の使用
- 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
- チュートリアル: Amazon Elastic Kubernetes Service から Amazon Keyspaces に接続する

プログラムにより Amazon Keyspaces にアクセスするための認証情報の作成

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 ユーザーには長期的な認証情報があり、セキュリティ上のリスクがあります。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、すべてのサービスで使用される IAM ID AWS 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

- サービス固有の認証情報を生成する
- Amazon Keyspaces AWS 認証情報を作成、設定する方法

サービス固有の認証情報を生成する

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おり、他の IAM ユーザーでは使用できません。

> **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的な認証情報であり、作成されたサービスにのみ使用できます。IAM ロールまたはフェデレーティッド ID に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アクセス許可を付与す

るには、AWS Amazon Keyspaces の SigV4 認証プラグインによる認証を使用する必要があります。

サービス固有の認証情報を生成するには、次のいずれかの手順を使用します。

コンソールを使用してサービス固有の認証情報を生成する

コンソールを使用してサービス固有の認証情報を生成するには

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。

を使用してサービス固有の認証情報を生成する AWS CLI

を使用してサービス固有の認証情報を生成するには 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. http://aws.amazon.com/cli AWS CLI で をダウンロードします。

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

2. ユーザーガイドの「CLI のインストール AWS」および「AWS CLI の設定」の手順に従います。 AWS Command Line Interface

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 AWS 認証情報を作成、設定する方法

、 AWS SDK AWS CLI、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。 AWS プログラム的に使用する場合は、 AWS アクセスキーを提供して、プログラムによる呼び出しで ID を確認できるようにします。 AWS アクセスキーは、アクセスキー ID (たとえば AKIAIOSFODNN7EXAMPLE) とシークレットアクセスキー (たとえば、wJalrXUtnFEMI I/K7MDENG/ CYEXAMPLEKEY) で構成されます。bPxRfi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では、制限付きの権限を持つ IAM ユーザーを作成し、代わりに IAM ロールを特定のタスクの実行に必要な権限に関連付けることを推奨しています。これにより、IAM ユーザーは一時的に IAM ロールを引き受け、必要なタスクを実行できます。たとえば、Amazon Keyspaces コンソールを使用するアカウント内のIAMユーザーは、ロールに切り替えて、コンソール内のそのロールの権限を一時的に使用することができます。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ために使用する認証情報は一時的なものです。それどころか、IAM ユーザーは長期的な認証情報を持っているため、ロールを引き受ける代わりに権限が直接割り当てられていると、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。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、サービス」を参照してください。

トピック

- 、 AWS SDK AWS CLI、または Cassandra クライアントドライバ用の Amazon Keyspaces SigV4 プラグインに必要な認証情報
- アカウント内の Amazon Keyspaces にプログラムでアクセスするための IAM ユーザーの作成 AWS
- IAM ユーザーの新しいアクセスキーの作成
- IAM ユーザーのアクセスキーを管理する方法
- 一時認証情報を使用した IAM ロールと SigV4 プラグインによる Amazon Keyspaces への接続

、 AWS SDK AWS CLI、または Cassandra クライアントドライバ用の 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 AWS ユーザーまたはロールに関連付けられた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 “一時認証情報を使用した Amazon Keyspaces への接続”」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。

アカウント内の Amazon Keyspaces にプログラムでアクセスするための IAM ユーザーの作成 AWS

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、、Windows 用ツールで AWS CLI PowerShell、または API オペレーションを使用してユーザーを作成します。 AWS でユーザーを作成すると AWS Management Console、認証情報は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。

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、「Permissions required to access IAM resources (IAM リソースへのアクセスに必要な許可)」を参照してください。

IAM ユーザーの作成 (コンソール)

を使用して IAM ユーザーを作成できます。 AWS Management Console

プログラムアクセス権がある IAM ユーザーを作成するには (コンソール)

1. AWS Management Console にサインインし、https://console.aws.amazon.com/iam/ にある 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。

   このオプションでは、 AWS アカウントで利用できる管理ポリシーと顧客管理ポリシーのリストが表示されます。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。

   テストや接続チュートリアルに従うには、新しい IAM `AmazonKeyspacesReadOnlyAccess` ユーザー用の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。カスタムアクセス権限ポリシーを作成したら、そのポリシーをロールにアタッチし、ユーザーが一時的に適切なロールを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。

IAM ユーザーの作成 (AWS CLI)

を使用して IAM ユーザーを作成できます。 AWS CLI

プログラムアクセス権がある IAM ユーザーを作成するには (AWS CLI)

1. AWS CLI 次の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
   - Windows 用ツール PowerShell:New-IAMAccessKey
   - 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の作成

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。アクセスキーのローテーションの方法など、キー管理の詳細については、「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 ユーザーのアクセスキーを作成するには (コンソール)

1. AWS Management Console にサインインし、https://console.aws.amazon.com/iam/ にある 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:bPxRfiwJalrXUtnFEMI I/K7MDENG/ CYEXAMPLEKEY

   > **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。

IAM ユーザーのアクセスキーを管理する方法

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 SDK AWS とコマンドラインツールを使用すると、アクセスキーを既知の場所に置くことができるため、コード内に保管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” の必須の前提条件です。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>
```

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>
```

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>"
```

PowerShell 前の例のようにプロンプトで環境変数を設定すると、その値は現在のセッションの間だけ保存されます。 PowerShell 環境変数の設定をすべてのセッションとコマンドプロンプトセッションにわたって保持するには、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環境変数設定を保存します。または、 PowerShell 変数をプロファイルに追加して、future PowerShell すべてのセッションで変数を設定できます。環境変数を保存したり、セッション間で環境変数を保持したりする方法について詳しくは、PowerShell ドキュメントを参照してください。

一時認証情報を使用した IAM ロールと SigV4 プラグインによる Amazon Keyspaces への接続

セキュリティを強化するために、一時認証情報を使用して SigV4 プラグインで認証を行ってください。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 (API) への切り替え」を参照してください。AWS

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. AWS Management Console にサインインし、https://console.aws.amazon.com/iam/ にある 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。

一時的なセキュリティ認証情報をコードで使用するには、プログラムで AWS Security Token Service API `AssumeRole` のようなものを呼び出し、その結果の認証情報とセッショントークンを前のステップで作成した IAM ロールから抽出します。その後、それらの値を以降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 SDK のドキュメントは、AWS メインのドキュメントページの AWS SDK と Toolkit セクションにあります。

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

Amazon Keyspaces にアクセスするには、プログラムで `cqlsh` クライアントを実行するか、Apache 2.0 ライセンス取得済み 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 は以下の AWS リージョン で利用できます。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 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

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.ap-southeast-1.amazonaws.com	HTTPS および TLS	
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	

リージョン名	リージョン	エンドポイント	プロトコル	
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	
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

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

以下の Amazon Keyspaces エンドポイントは、AWS中国リージョンで利用できます。

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。

「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

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。を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 にない場合はPATH、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。Ubuntu Linux の例を次に示します。

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

パッケージがインストールされていることを確認するには、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。

   デフォルトのリージョンを Unix ベースのシステム上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 “認証用の IAM 認証情報 AWS ”」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、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を表示および編集する方法の詳細については、Amazon EC2」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” 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
```

AWS CLI の使用

AWS Command Line Interface (AWS CLI) を使用すると、複数の AWS のサービスをコマンドラインから制御したり、スクリプトで自動化したりできます。Amazon Keyspaces では、テーブルの作成など、データ定義言語 (DDL) の操作に AWS CLI を使用できます。さらに、インフラストラクチャアズコード (IaC) サービスやツール (AWS CloudFormation や Terraform など) を使用できます。

Amazon Keyspaces で AWS CLI を使用するには、事前にアクセスキー 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 “認証用の IAM 認証情報 AWS ”」を参照してください。

AWS CLI で Amazon Keyspace 向けに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 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 での AWS CLI の使用

AWS CLI のダウンロードと設定

AWS CLI は、https://aws.amazon.com/cli で使用できます。Windows、macOS、または Linux 上で実行できます。AWS CLI をダウンロードしたら、以下の手順に従って、インストールと設定を行います。

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 での AWS CLI 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。AWS CLI では、パラメータ値の短縮構文および JSON をサポートしています。以下の Amazon Keyspaces 例では、AWS CLI 省略構文を使用します。詳細については、「CLI での省略構文の使用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、Amazon Keyspaces がそのキーにアクセスできるように、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 を使用する場合

AWS SDK と AWS Command Line Interface (AWS CLI) で、Amazon Keyspaces とインタラクティブに作業できます。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) サービスやツール (Terraform など) AWS CloudFormation も使用できます。

Amazon Keyspaces で AWS CLI を使用するには、事前にアクセスキー 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 “認証用の IAM 認証情報 AWS ”」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。

AWS SDK での Amazon Keyspaces 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	PowerShell コード例のツール
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。

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。これらは、Amazon Keyspaces でテスト済みで使用が推奨されているドライバーと最新バージョンです。

- `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**
>
> 使用開始に役立つように、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 end-to-end コード例を表示およびダウンロードできます。の「Amazon Keyspaces の例」を参照してください GitHub。

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 step-by-step チュートリアルについては、「」を参照してくださいthe section called “VPC エンドポイントとの接続”。

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 は現在、ベータ版でのみサポートされています。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 “認証用の IAM 認証情報 AWS ”」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 DataStax Java ドライバーを使用して Amazon Keyspaces に接続する S tep-by-step 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 S tep-by-step チュートリアル
- Apache Cassandra 用の 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 DataStax Java ドライバーを使用して Amazon Keyspaces に接続する S tep-by-step チュートリアル

次の step-by-step 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 用の Java ドライバーを使用して 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 4.0 バージョンを使用します。

トピック

ステップ 1: 前提条件

このチュートリアルに従うには、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する必要があります。

- 「the section called “サービス固有の認証情報”」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 「Apache Cassandra の 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

DataStax Java Cassandra ドライバーの設定を指定するには、アプリケーションの設定ファイルを作成します。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。と *`ServiceUser`*は、「」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する*`ServicePassword`*必要がありますサービス固有の認証情報を生成する。

> **Note**
>
> ドライバー設定ファイルで認証情報をハードコーディングする代わりに、Apache Cassandra 用 DataStax Java ドライバーの認証プラグインを使用して、短期認証情報を使用できます。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 – 設定ファイルに、 でクラスを指定する 1 行のセクションEngineFactory を追加して、SSL を初期化しますclass = DefaultSslEngineFactory。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 S tep-by-step チュートリアル

次のセクションでは、Apache Cassandra 用オープンソース 4.x DataStax Java ドライバーの SigV4 認証プラグインを使用して Amazon Keyspaces (Apache Cassandra 用) にアクセスする方法について説明します。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 “認証用の IAM 認証情報 AWS ”」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 “認証用の IAM 認証情報 AWS ” 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方法”」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 Apache Cassandra 用 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 用の DataStax Java ドライバーのバージョン 4.x をサポートしています。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**
>
> プラグインのバージョンを、GitHub リポジトリ に示すように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

DataStax Java Cassandra ドライバーの設定を指定するには、アプリケーションの設定ファイルを作成します。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にプラグインによって提供される認証ハンドラーです。
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???。使用可能なすべてのノードに対してロードバランシング`slow-replica-avoidance = false`するように を設定します。
3. Idempotence – アプリケーションのデフォルト`idempotence`を に設定`true`して、失敗した読み取り/書き込み/準備/実行リクエストを常に再試行するようにドライバーを設定します。これは、失敗したリクエストを再試行して一時的な障害を処理するのに役立つ分散アプリケーションのベストプラクティスです。

4. SSL/TLS – でクラスを指定する 1 行で設定ファイルにセクションEngineFactory を追加して、SSL を初期化しますclass = DefaultSslEngineFactory。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. Connections – を設定して、エンドポイントごとに少なくとも 3 つのローカル接続を作成しますlocal.size = 3。これは、アプリケーションがオーバーヘッドとトラフィックバーストを処理するのに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エンドポイントあたりのローカル接続数を計算する方法の詳細については、「」を参照してくださいthe section called “接続を設定する方法”。
6. 再試行ポリシー – Amazon Keyspaces 再試行ポリシーAmazonKeyspacesExponentialRetryPolicyは、Cassandra ドライバーに付属DefaultRetryPolicyする の代替手段です。2 つの再試行ポリシーの主な違いは、 の再試行回数をAmazonKeyspacesExponentialRetryPolicyニーズに合わせて設定できることです。デフォルトでは、 の再試行回数AmazonKeyspacesExponentialRetryPolicyは 3 に設定されています。さらに、Amazon Keyspaces 再試行ポリシーは汎用 を返しませんNoHostAvailableException。代わりに、Amazon Keyspaces 再試行ポリシーは、サービスによって返された元の例外を返します。再試行ポリシーを実装するその他の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。
7. 準備済みステートメント – ネットワークの使用を最適化するには、false prepare-on-all-nodes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 用の 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 用の 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。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 “認証用の IAM 認証情報 AWS ”」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 “認証用の IAM 認証情報 AWS ”」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方法”」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 Apache Cassandra 用 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 用の DataStax Java ドライバーのバージョン 3.x をサポートしています。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。プラグインのバージョンを、GitHub リポジトリ に示されている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 AWS 。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 “認証用の IAM 認証情報 AWS ”」を参照してください。

トピック

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. 「」の手順に従って、*ServiceUser##*と 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*ServicePassword*と一致することを確認しますサービス固有の認証情報を生成する。
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 用の Python ドライバーと SigV4 認証プラグインを使用して DataStax Amazon Keyspaces に接続する

次のセクションでは、Apache Cassandra 用のオープンソース DataStax Python ドライバーに SigV4 認証プラグインを使用して Amazon Keyspaces (Apache Cassandra 用) 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 “認証用の IAM 認証情報 AWS ” 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 “一時認証情報を使用した Amazon Keyspaces への接続”」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ています。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 AWS 。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 “認証用の IAM 認証情報 AWS ”」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 用の 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. 「」の手順に従って、*ServiceUser##*と 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*ServicePassword*と一致することを確認しますサービス固有の認証情報を生成する。
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 用の 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 用のオープンソース 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 “認証用の IAM 認証情報 AWS ” 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。

GitHub リポジトリ から Node.js SigV4 認証プラグインをアプリケーションに追加します。プラグインは Cassandra 用 Node.js ドライバーのバージョン 4.x DataStax をサポートしており、 AWS SDK for Node.js に依存しています。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. *####KeyId*と*######AccessKey*が、 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と一致していることを確認しますAWSCredentialsProvider。詳細については、「SDK for in Node.js」の「Setting CredentialsAWS JavaScript in Node.js」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方法” 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。を使用した例を次に示します PowerShell。

```
$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 を介して 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. 「」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*ServicePassword*と*ServiceUser##*が一致していることを確認しますサービス固有の認証情報を生成する。

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 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、すべてのサービスで使用される IAM ユーザーとロールの IAM アクセスキーを作成することをお勧めします AWS 。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 “認証用の IAM 認証情報 AWS ”」を参照してください。

トピック

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 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. 「」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*ServicePassword*と*ServiceUser##*が一致していることを確認しますサービス固有の認証情報を生成する。

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。

まだ完了していない場合は、the section called “認証用の IAM 認証情報 AWS ” 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。

GitHub リポジトリ から Go SigV4 認証プラグインを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 をサポートしており、 AWS SDK for Go に依存しています。

```
$ 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
    var auth sigv4.AwsAuthenticator = sigv4.NewAwsAuthenticator()
    auth.Region = "us-west-2"
    auth.AccessKeyId = "AKIAIOSFODNN7EXAMPLE"
    auth.SecretAccessKey = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"

    cluster.Authenticator = auth

    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. *AccessKeyID* と*SecretAccess##*が、 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と一致していることを確認しますAwsAuthenticator。詳細については、AWS SDK for Go の「Configuring the AWS SDK for Go (SDK for Go 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方法” 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

「」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*ServicePassword*と*ServiceUser##*が一致していることを確認しますサービス固有の認証情報を生成する。

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
```

チュートリアル: Amazon Elastic Kubernetes Service から Amazon Keyspaces に接続する

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ホストするように Amazon Elastic Kubernetes Service (Amazon EKS) クラスターを設定するために必要な手順について説明します。

Amazon EKS は、独自の Kubernetes コントロールプレーンをインストール、運用、保守する必要がないマネージドサービスです。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 Amazon EKS に設定、ビルド、デプロイするための step-by-step ガイダンスを提供します。最後のステップでは、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

チュートリアルの前提条件

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、 へのサインアップ AWS と、Amazon Keyspaces へのアクセス権を持つ AWS Identity and Access Management (IAM) プリンシパルの作成が含まれます。
2. このチュートリアルの後半で Amazon EKS で実行userされているコンテナ化されたアプリケーションから書き込むことができる名前awsと名前 を持つテーブルを使用して、Amazon Keyspaces キースペースを作成します。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'
   ```

   キー空間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、「 コマンドリファレンス」の「キースペースの作成」と「テーブルの作成」を参照してください。 AWS CLI

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";
```

テーブルは、このステートメントの出力にリストされている必要があります。テーブルが作成されるまで遅延が発生する可能性があることに注意してください。詳細については、「the section called “CREATE TABLE”」を参照してください。

3. Fargate - Linux ノードタイプで Amazon EKS クラスターを作成します。Fargate は、Amazon EC2 インスタンスを管理せずに Kubernetes ポッドをデプロイできるサーバーレスコンピューティングエンジンです。すべてのコマンド例でクラスター名を更新せずにこのチュートリアルに

従うには、「Amazon EKS ユーザーガイドmy-eks-cluster」の「Amazon EKS の開始方法」の手順に従って、名前 でクラスターを作成します。 eksctlクラスターが作成されたら、ノードと 2 つのデフォルトポッドが実行中で正常であることを確認します。これを行うには、次のコマンドを使用します。

```
kubectl get pods -A -o wide
```

この出力のような内容が表示されます。

```
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、Docker イメージをプッシュ、プル、管理するために任意の CLI で使用できる 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、 という名前のリポジトリを作成できますmy-ecr-repository。

   ```
   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 クラスターを設定する” 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が Amazon Keyspaces テーブルに接続するために必要な IAM リソースを作成します。

1. Amazon EKS クラスターの Open ID 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に IAM ロールを使用する場合に必要です。OIDC プロバイダーとその作成方法の詳細については、「Amazon EKS ユーザーガイド」の「クラスターの IAM OIDC プロバイダーの作成」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、クラスター名が であることを前提としていますmy-eks-cluster。別の名前のクラスターがある場合は、以降のすべてのコマンドで名前を更新してください。

   ```
   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) を書き留めておきます。次のステップでサービスアカウントの信頼ポリシーを作成するときに必要になります。

   ```
   {
       "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は、ポッドで実行されるプロセスの ID を提供します。ポッドは、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る最小のシンプルな Kubernetes オブジェクトです。次に、サービスアカウントが リソースへのアクセス許可を取得するために引き受けることができる IAM ロールを作成します。 AWS サービスアカウントを使用するように設定されたポッドから任意のサービ

スにアクセスでき、その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ます。

a. サービスアカウントの新しい名前空間を作成します。名前空間は、このチュートリアル用に作成されたクラスターリソースを分離するのに役立ちます。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. カスタム名前空間を使用するには、それを 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-serviceaccount`に という名前`my-eks-namespace`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、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。この信頼関係は、プリンシパルがロールを引き受ける前に必要です。ファイルに対して次の編集を行う必要があります。

- には`Principal`、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名と名前空間が正しいことを確認します。

IAM ロールを作成するときは、次のステップで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または名前空間がロールを引き受けることを許可することもできます。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 IAM アクセス許可」を参照してください。

3. 引き受ける my-iam-role Amazon EKS サービスアカウントの名前で IAM ロールを作成します。最後のステップ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 プロバイダーを指定します。

```
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、最小特権の原則を実装するカスタム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 ロールに正常に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、引き受けることができる IAM ロールの Amazon リソースネーム (ARN) をアノテーションします。ロール ARN は、必ず アカウント ID で更新してください。

   ```
   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. IAM ロールの信頼ポリシーが次のステートメントで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 ロールの注釈が付けら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、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、「」に進みますthe section called “ステップ 2: アプリケーションを設定する”。

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および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して、すべての設定手順を完了することもできます。

アプリケーションを設定し、必要な依存関係を追加します。

1. 次のコマンドを使用して Github リポジトリをクローンすることで、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` ディレクトリに移動して `application.conf` ファイルを見つけます。

   a. アプリケーション設定

      このステップでは、SigV4 認証プラグインを設定します。アプリケーションで次の例を使用できます。まだ生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) を生成し、 AWS 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。その他のサービスエンドポイントについては、「」を参照してくださいthe section called “サービスエンドポイント”。信頼ストアの場所、信頼ストア名、信頼ストアのパスワードを独自のパスワードに置き換えます。

      ```
      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。これは、次の例に基づいて実行できます。

```
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-sts</artifactId>
            <version>1.11.717</version>
        </dependency>
```

c. SigV4 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ています。

```
        <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。

ログがないと、接続の問題のトラブルシューティングは不可能です。このチュートリアルでは、をログ記録フレームワークslf4jとして使用し、 logback.xml を使用してログ出力を保存します。ログ記録レベルを に設定debug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**
>
> このdebugレベルは、接続障害を調査するために必要です。アプリケーションから Amazon Keyspaces に正常に接続したら、warning必要に応じてログ記録レベルを infoまたは 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、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、Amazon Elastic Container Registry に送信する

1. を定義するビルドの環境変数を設定します AWS リージョン。例のリージョンを独自のリージョンに置き換えます。

```
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、Apache Maven バージョン 3.6.3 以降でアプリケーションをコンパイルします。

```
mvn clean install
```

これにより、targetディレクトリに含まれるすべての依存関係を含む JAR ファイルが作成されます。

3. 次のコマンドを使用して、次のステップに必要な ECR リポジトリ URI を取得します。リージョンは、使用しているリージョンに更新してください。

```
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、前のステップのリポジトリ URI を使用して Docker イメージを構築します。必要に応じて Docker ファイルを変更します。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。これを行うには、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: Amazon EKS にアプリケーションをデプロイし、Amazon Keyspaces 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実行中であり、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、 というファイルで関連するすべての設定を構成する必要がありますdeployment.yaml。このファイルは、Amazon EKS によってアプリケーションをデプロイするために使用されます。ファイル内のメタデータには、次の情報が含まれている必要があります。

- 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、 を使用しますmy-keyspaces-app。
- 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、 を使用しますmy-eks-namespace。

- 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、 を使用しますmy-eks-serviceaccount。
- 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、 を使用しますmy-keyspaces-app。
- イメージ URI Amazon ECR の Docker イメージ URI。
- AWS アカウント ID AWS アカウント ID。
- IAM ロール ARN は、サービスアカウントが引き受ける IAM ロールの ARN です。このチュートリアルでは、 を使用しますmy-iam-role。
- 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て実行します。

1. deployment.yaml ファイルを設定します。次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

   次のファイルを例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内の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
```

詳細については、次のコマンドを使用できます。

```
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 テーブルに接続できることを確認します。これを行うには、次のコマンドを使用します。必ずデプロイの名前を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が表示されます。

```
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、1 行のデータがテーブルに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

このチュートリアルで作成したすべての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. デプロイを削除します。これを行うには、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 クラスターとそれに含まれるすべてのポッドを削除します。これにより、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。これを行うには、次のコマンドを使用します。

```
eksctl delete cluster --name my-eks-cluster --region aws-region
```

3. Amazon Keyspaces へのアクセス許可を持つ Amazon EKS サービスアカウントに使用される 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管理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. リポジトリに保存されているすべてのイメージを含む Amazon ECR リポジトリを削除します。これを行うには、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フラグが必要であることに注意してください。最初にイメージを削除するには、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「イメージの削除」を参照してください。

5. Amazon Keyspaces キースペースとテーブルを削除します。キー空間を削除すると、そのキー空間内のすべてのテーブルが自動的に削除されます。これを行うには、次のいずれかのオプションを使用できます。

   AWS CLI

   ```
   aws keyspaces delete-keyspace --keyspace-name 'aws'
   ```

   キー空間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
   ```

   最初にテーブルを削除するには、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、「 コマンドリファレンス」の「キー空間の削除」と「テーブルの削除」を参照してください。 AWS CLI

   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 ;
   ```

キースペースは、このステートメントの出力にリストされていないはずです。キー空間が削除されるまでには遅延が発生する可能性があることに注意してください。詳細については、「the section called “DROP KEYSPACE”」を参照してください。

最初にテーブルを削除するには、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いようにしてください。テーブルが削除されるまで遅延が発生する可能性があることに注意してください。詳細については、「the section called “DROP TABLE”」を参照してください。

インターフェイス VPC エンドポイントによる Amazon Keyspaces との接続

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 サービス間のプライベート通信を可能にする AWS サービスである VPCs。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

- 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 へのアクセス権を持つ 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、「開始」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 (https://console.aws.amazon.com/ec2/) を開きます。
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

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 トラフィックを認証します。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] を選択します。

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] を選択します。

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と AWS CLI 手順の詳細については、「Amazon EC2 ユーザーガイド」の「Linux インスタンスのインバウンドトラフィックの承認」を参照してください。 Amazon EC2

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。ローカル環境または で Amazon Keyspaces CQL エディタを使用する場合 AWS Management Console、接続は 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。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. Type を CQLSH/CASSANDRA として持つインバウンドルールを追加します。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. ポリシーのリストから、AmazonKeyspacesReadOnlyアクセス を選択し、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: でモニタリングを設定する CloudWatch

このステップでは、Amazon を使用して Amazon Keyspaces CloudWatch への VPC エンドポイント接続をモニタリングする方法を示します。

AWS PrivateLink は、インターフェイスエンドポイント CloudWatch に関するデータポイントを 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。`AWS/PrivateLinkEndpoints` の名前空間には、インターフェイスエンドポイントのメトリクス CloudWatch が含まれます。詳細については、「 AWS PrivateLink ガイドCloudWatch 」の「 のメトリクス AWS PrivateLink」を参照してください。

VPC エンドポイントメトリクスを使用して CloudWatch ダッシュボードを作成するには

1. で CloudWatch コンソールを開きますhttps://console.aws.amazon.com/cloudwatch/。
2. ナビゲーションペインで、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/PrivateLink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「ダッシュボードの作成」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メトリクスを使用して、接続ごとの 1 秒あたりの最大オペレーション数のクォータを超えた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 つのサブネットにまたが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 CloudWatch の NewConnectionsで 7 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 つのサブネットにまたが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 CloudWatch の NewConnectionsで 5 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 (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。
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。

Amazon Keyspaces 用のクロスアカウントアクセスの設定

リソースの分離や、開発環境や本番環境などの異なる環境におけるリソース使用のために、AWS アカウント を個別に作成して使用することができます。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

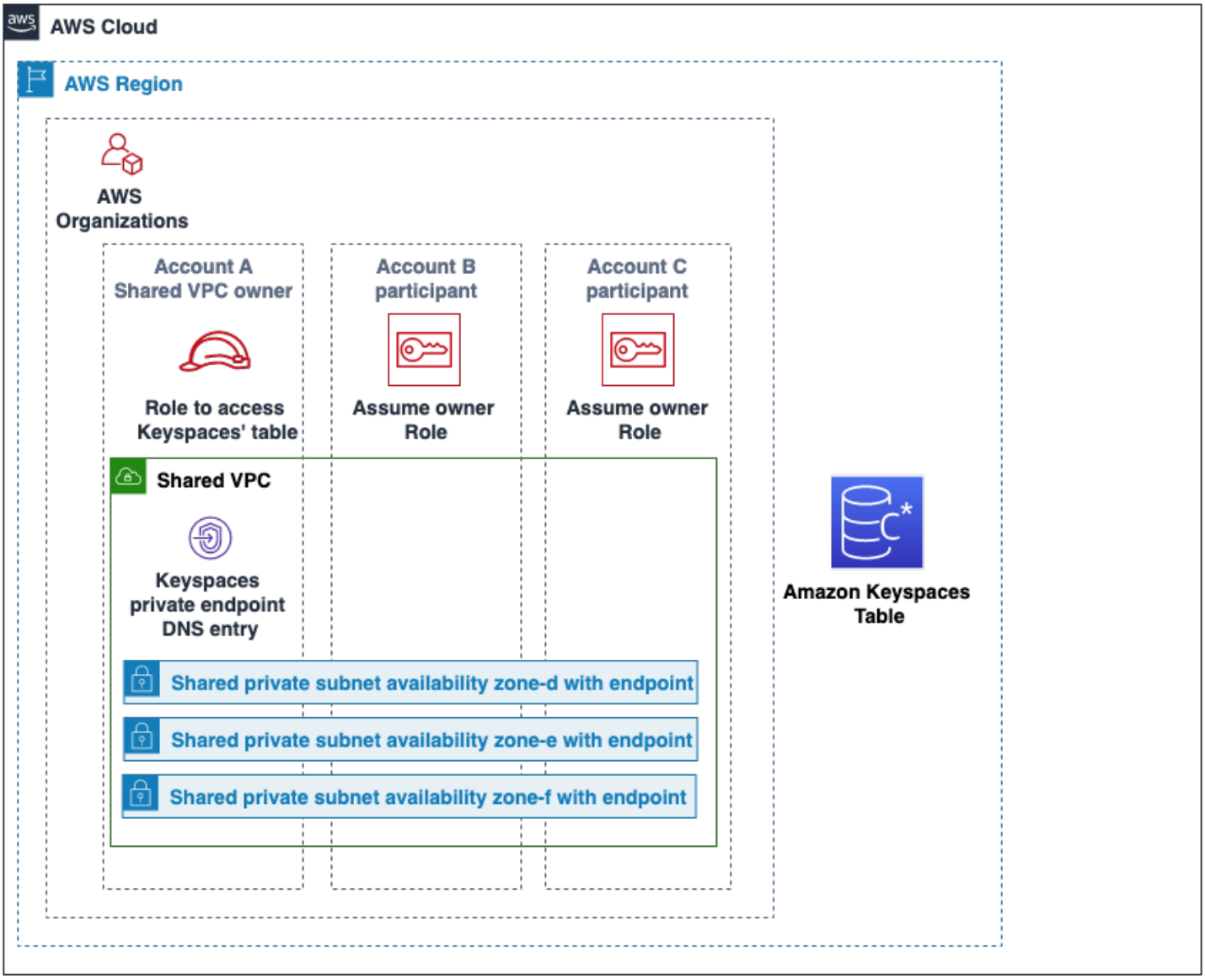
共有 VPC の Amazon Keyspaces 用のクロスアカウントアクセスの設定

さまざまなAWS アカウントを作成して、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 と Account C がアクセスする必要のあるリソースがあるアカウントなので、Account A は 信頼するアカウントです。 Account B と Account C は、Account A 内のリソースにアクセスする必要のあるプリンシパルを持つアカウントなので、 Account B と Account C は、信頼されるアカウントです。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 で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 と Account C と共有します。

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. AWS PrivateLink を利用した Amazon Keyspaces プライベート VPC エンドポイントを作成します。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. 以下のポリシー例に示すように、Amazon Keyspaces テーブルへのフルアクセス権限、Amazon Keyspaces システムテーブルへの読み取りアクセス権限があり、Amazon EC2 VPC リソースを記述できる IAM ロールを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountAccess",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcEndpoints",
                "cassandra:*"
            ],
            "Resource": "*"
        }
    ]
}
```

5. 次の例のように、Account B と Account C が信頼できるアカウントと見なすことができる IAM ロール信頼ポリシーを設定します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "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"
        }
    ]
}
```

プリンシパルによる共有ロールの引き受けを許可するには、AWS Security Token Service (AWS STS) の AssumeRole API を使用します。詳細については、『IAM ユーザーガイド』の「所有している別の AWS アカウント の IAM ユーザーにアクセス権を与える」を参照してください。

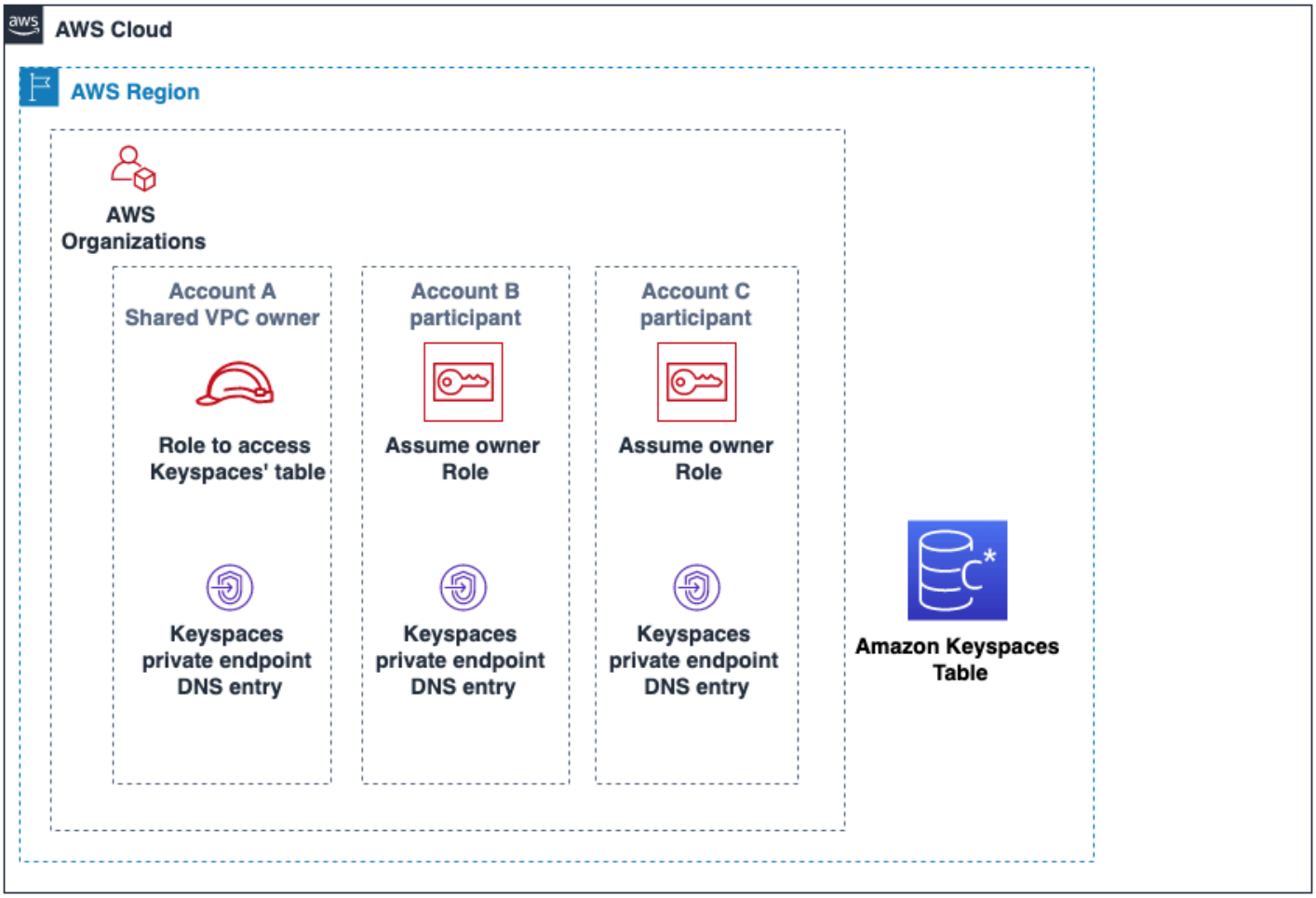
2. Account B と Account C では、SIGV4 認証プラグインを利用するアプリケーションを作成できます。これにより、アプリケーションは共有ロールを引き受け、共有 VPC の VPC エンドポイントを介して Account A にある Amazon Keyspaces テーブルに接続できます。SIGV4 認証プラグインの詳細については、「the section called “認証情報の作成”」を参照してください。

共有 VPC のない Amazon Keyspaces クロスアカウントアクセスの設定

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れば、共有 VPC エンドポイントなしでさまざまなアカウントの 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。



Amazon Keyspaces (Apache Cassandra 向け) の使用開始

このチュートリアルは、Apache Cassandra と Amazon Keyspaces (Apache Cassandra 向け) を初めて使用する方が対象です。このチュートリアルでは、Amazon Keyspaces を正しく使用するために必要なすべてのプログラムとドライバーをインストールします。

別の 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。

トピック

- チュートリアルの前提条件と考慮事項
- チュートリアルステップ 1: Amazon Keyspaces でキー空間とテーブルを作成する
- チュートリアルステップ 2: データの作成、読み取り、更新、削除 (CRUD) を行う
- チュートリアルステップ 3: Amazon Keyspaces でテーブルとキー空間を作成する

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 に記載されている設定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) AWS ユーザーへのサインアップと作成が含まれます。

さらに、cqlsh または Apache 2.0 ライセンス付き Cassandra クライアントドライバーを使用してチュートリアルを実行する場合は、cqlsh を使用した Amazon Keyspaces への接続 のセットアップ手順を実行してください。

前提条件のステップが完了したら、チュートリアルステップ 1: Amazon Keyspaces でキー空間とテーブルを作成する に進みます。

チュートリアルステップ 1: Amazon Keyspaces でキー空間とテーブルを作成する

このセクションでは、コンソールを使用してキー空間を作成し、それをテーブルに追加します。

> **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。

トピック

- キー空間の作成
- テーブルの作成

キー空間の作成

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。キー空間には 1 つ以上のテーブルが含まれており、キー空間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空間によって定義されます。キー空間の詳細については、次のトピックを参照してください。

- キー空間の操作: the section called “キースペースの作成”
- 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

キー空間を作成するときは、キー空間名を指定する必要があります。

> **Note**
>
> キー空間のレプリケーション戦略は `SingleRegionStrategy` でなければなりません。`SingleRegionStrategy` では、 AWS リージョンの 3 つのアベイラビリティーゾーン間にデータがレプリケートされます。

コンソールを使用する場合

コンソールを使用してキー空間を作成するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home にある 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 (キー空間の作成)] を選択します。
4. [Keyspace name (キー空間名)] ボックスに、キー空間の名前として **myGSGKeyspace** を入力します。

   名前の制約:

   - 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. キー空間を作成するには、[Create keyspace (キー空間の作成)] を選択します。
6. 以下の作業により、キー空間 myGSGKeyspace 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. キー空間のリストで該当するキー空間 myGSGKeyspace を見つけます。

CQL の使用

以下の手順では、CQL を使用してキー空間を作成します。

CQL を使用してキー空間を作成するには

1. コマンドシェルを開き、次のコマンドを入力します。

   cqlsh
2. 次の CQL コマンドを使用して、キー空間を作成します。

```
CREATE KEYSPACE IF NOT EXISTS "myGSGKeyspace"
   WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

   SingleRegionStrategy3 のレプリケーション係数を使用し、そのリージョン内の 3 AWS つのアベイラビリティーゾーンにデータを複製します。

> **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。この場合は "myGSGKeyspace" と記述します。

3. キー空間が作成されていることを確認します。

```
SELECT * from system_schema.keyspaces ;
```

そのキー空間が表示されているはずです。

テーブルの作成

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。

- テーブルの操作: the section called “テーブルの作成”
- 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管理
- テーブルのリソース使用率のモニタリング: the section called “によるモニタリング CloudWatch”
- 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

次の手順を使用して、指定した列、データ型、パーティションキー、およびクラスタリング列が含まれているテーブルを作成します。

コンソールを使用する場合

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 `employees_tbl` を作成します。

```
ID           text
```

```
name          text
region        text
division      text
project       text
role          text
pay_scale     int
vacation_hrs  float
manager_id    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 にある 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空間として myGSGKeyspace 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「**employees_tbl**」を入力します。

   名前の制約:

   - 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。

   ```
   id            text
   name          text
   region        text
   division      text
   project       text
   role          text
   pay_scale     int
   vacation_hrs  float
   manager_id    text
   ```

a. [Name (名前)] — 列の名前を入力します。

名前の制約:

- 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. 別の列を追加する場合は、[Add column (列の追加)] を選択します。

7. [Partition Key (パーティションキー)] で、パーティションキーとして `id` 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 `division` 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. [Column (列)] リストで、[division (除算)] 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。)

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。

e. [clustering (クラスタリング)] 列で、[division (除算)] が true で識別されていることを確認します。他のすべてのテーブル列が false になっているはずです。

CQL の使用

以下の手順では、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。id 列はパーティションキーになります。

```
id            text
name          text
region        text
division      text
project       text
role          text
pay_scale     int
vacation_hrs  float
manager_id    text
```

CQL を使用してテーブルを作成するには

1. コマンドシェルを開き、次のコマンドを入力します。

   cqlsh

2. cqlsh プロンプト (cqlsh>) で、テーブルを作成するキー空間を指定します。

   ```
   USE "myGSGKeyspace" ;
   ```

3. キー空間プロンプト (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 IF NOT EXISTS "myGSGKeyspace".employees_tbl (
      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 CLUSTERING ORDER BY (division ASC) ;
```

> **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。

id 列はパーティションキーになるので注意が必要です。次に、division は昇順 (ASC) で並べられたクラスタリング列です。

4. 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='myGSGKeyspace' ;
```

そのテーブルが表示されているはずです。

5. テーブルの構造を確認します。

```
SELECT * FROM system_schema.columns WHERE keyspace_name = 'myGSGKeyspace' AND
 table_name = 'employees_tbl' ;
```

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。

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 “ステップ 2: CRUD オペレーション” に進みます。

チュートリアルステップ 2: データの作成、読み取り、更新、削除 (CRUD) を行う

このセクションでは、コンソールの CQL エディタを使用して、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)) のオペレーションを実行します。cqlsh を使用してコマンドを実行することもできます。

トピック

- チュートリアル: Amazon Keyspaces テーブルへのデータの挿入とロード
- チュートリアル: Amazon Keyspaces テーブルからの読み取り
- チュートリアル: Amazon Keyspaces テーブルのデータを更新する
- チュートリアル: Amazon Keyspaces テーブルのデータを削除する

チュートリアル: Amazon Keyspaces テーブルへのデータの挿入とロード

employees_tbl 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. 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;
   ```

2. 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 "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') ;
   ```

3. 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 "myGSGKeyspace".employees_tbl ;
   ```

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. 次のアーカイブファイル sampledata.zip に含まれているサンプルデータファイル (employees.csv) をダウンロードします。この CSV (カンマ区切り値) ファイルには、次のデータが含まれています。ファイルの保存先のパスを覚えておいてください。

ID	name	project	region	division	role	pay_scale	vacation_hrs	manager_id
123-45-6789	Bob	NightFlight	US	Engineering	Intern	1	0	234-56-7890
234-56-7890	Bob	NightFlight	US	Engineering	Manager	6	72	789-01-2345
345-67-8901	Sarah	Storm	US	Engineering	IC	4	108	234-56-7890
456-78-9012	Beth	NightFlight	US	Engineering	IC	7	100.5	234-56-7890
567-89-0123	Ahmed	NightFlight	US	Marketing	IC	4	88	678-90-1234
678-90-1234	Alan	Storm	US	Marketing	Manager	3	18.4	789-01-2345
789-01-2345	Roberta	All	US	Executive	CEO	15	184	None

2. コマンドシェルを開き、次のコマンドを入力します。

   cqlsh

3. cqlsh プロンプト (cqlsh>) で、キー空間を指定します。

```
USE "myGSGKeyspace" ;
```

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. キー空間プロンプト (cqlsh:keyspace_name>) で、次のクエリを実行します。

```
COPY employees_tbl
 (id,name,project,region,division,role,pay_scale,vacation_hrs,manager_id)
FROM 'path-to-the-csv-file/employees.csv' WITH delimiter=',' AND header=TRUE ;
```

6. 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 employees_tbl ;
```

チュートリアル: Amazon Keyspaces テーブルからの読み取り

「チュートリアル: Amazon Keyspaces テーブルへのデータの挿入とロード」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の選択
- 列のサブセットの選択
- 行のサブセットの選択

テーブル内のすべてのデータの選択

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、本番環境では、通常、このコマンドの実行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

- 次のクエリを実行します。

  ```
  SELECT * FROM "myGSGKeyspace".employees_tbl ;
  ```

  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。

列のサブセットの選択

列のサブセットをクエリするには

- id 列、name 列、manager_id 列のみを検索するには、次のクエリを実行します。

  ```
  SELECT name, id, manager_id FROM "myGSGKeyspace".employees_tbl ;
  ```

  出力には、指定した列だけが SELECT ステートメントに示されている順序で含まれます。

行のサブセットの選択

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

- ID が '234-56-7890' である従業員のレコードのみを検索するには、次のクエリを実行します。

  ```
  SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='234-56-7890' ;
  ```

  前述の SELECT ステートメントにより、id が 234-56-7890 である行のみが返されます。

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、テーブルのプライマリキー内の唯一の列です。
  - 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 句の列はテーブルのプライマリキーの列と完全に一致し、それ以上でも以下でもありません。

- 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、employees_tbl テーブルで以下の検索を実行してみましょう。

- すべての従業員の name、project、id を検索します。
- インターンの Bob が従事しているプロジェクトを検索する (出力には少なくとも彼の氏名、プロジェクト、役割を含める)。
- アドバンスト: インターンの Bob と同じマネージャーを持つすべての従業員を検索するアプリケーションを作成します。ヒント: この作業には複数のクエリが必要になる場合があります。
- アドバンスト: プロジェクト NightFlight で働くすべての従業員の選択された列を検索するアプリケーションを作成します。ヒント: これを解決するために複数のステートメントが必要になる場合があります。

チュートリアル: Amazon Keyspaces テーブルのデータを更新する

employees_tbl 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 id='12345' ;
```

- 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

employees_tbl テーブルを使用して、ID が 567-89-0123 である従業員の給与を引き上げます。

```
UPDATE "myGSGKeyspace".employees_tbl SET pay_scale=5 WHERE id='567-89-0123' AND
 division='Marketing' ;
```

その従業員の給与水準が 5 になっていることを確認します。

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='567-89-0123' ;
```

試してみましょう

アドバンスト: 会社がインターンのボブを雇用しました。彼の役割が 'IC' で給与水準が 2 になるようにレコードを変更します。

チュートリアル: Amazon Keyspaces テーブルのデータを削除する

employees_tbl 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。キー空間を削除すると、キー空間とその中のすべてのテーブルが削除されます。削除したキー空間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。

セルの削除

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 ;
```

employees_tbl テーブルで、マネージャーに関して CEO は "None" であることを確認できます。まず、そのセルを削除して、そのセルにデータを保存しないようにします。

特定のセルを削除するには

1. 次の DELETE クエリを実行します。

```
DELETE manager_id FROM "myGSGKeyspace".employees_tbl WHERE id='789-01-2345' AND
 division='Executive';
```

2. 削除が想定どおりに行われたことを確認します。

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='789-01-2345' AND
 division='Executive';
```

行の削除

従業員が退職する場合など、行全体を削除する必要がある場合があるかもしれません。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. 次の DELETE クエリを実行します。

```
DELETE FROM "myGSGKeyspace".employees_tbl WHERE id='456-78-9012' AND
 division='Engineering';
```

2. 削除が想定どおりに行われたことを確認します。

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='456-78-9012' AND
 division='Engineering';
```

チュートリアルステップ 3: Amazon Keyspaces でテーブルとキー空間を作成する

不要なテーブルとデータに対して課金されないようにするために、使用していないテーブルとキー空間をすべて削除します。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、キー空間は残ります。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。

テーブルの削除

コンソールまたは CQL 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場合

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 にある 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、`employees_tbl` 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. コマンドシェルを開き、次のコマンドを入力します。

cqlsh

2. キー空間プロンプト (cqlsh:*keyspace_name*>) で以下のコマンドを入力して、テーブルを削除します。

```
DROP TABLE IF EXISTS "myGSGKeyspace".employees_tbl ;
```

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 = 'myGSGKeyspace' ;
```

そのテーブルは表示されていないはずです。

キー空間の削除

キースペースは、または CQL を使用して削除できます。 AWS Management Console キー空間を削除すると、キー空間と、そのすべてのテーブルとデータが削除されます。

AWS Management Consoleの使用

以下の手順では、 AWS Management Consoleを使用してキー空間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空間を削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/keyspaces/home にある 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. 削除する各キー空間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 (キー空間の削除)] 画面で、ボックスに **Delete** を入力します。次に、[Delete keyspace (キー空間の削除)] を選択します。
6. キー空間 myGSGKeyspace 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 (キー空間)] を選択して、そのキー空間が表示されなくなったことを確認します。そのキー空間を削除したため、[Tables (テーブル)] に employees_tbl 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 を使用してキー空間とそのテーブルとデータをすべて削除します。

CQL を使用してキー空間を削除するには

1. コマンドシェルを開き、次のコマンドを入力します。

   cqlsh

2. キー空間プロンプト (cqlsh:*keyspace_name*>) で以下のコマンドを入力して、キー空間を削除します。

```
DROP KEYSPACE IF EXISTS "myGSGKeyspace" ;
```

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 ;
```

   そのキー空間は表示されていないはずです。これは非同期操作であるため、キースペースが削除されるまで時間がかかる場合があることに注意してください。

Amazon Keyspaces への移行

Amazon Keyspaces (Apache Cassandra 向け) への移行は、企業や組織にさまざまな魅力的なメリットをもたらします。ここでは、Amazon Keyspaces を移行のための魅力的な選択肢にする主な利点をいくつか紹介します。

- スケーラビリティ — Amazon Keyspaces は、大規模なワークロードを処理し、増加するデータボリュームとトラフィックに対応するためにシームレスにスケーリングす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実行されないため、将来のピークに備えて計画する必要があります。Amazon Keyspaces を使用すると、オンデマンドに基づいてテーブル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、アプリケーションはパフォーマンスを損なうことなくトラフィックの急増を処理できます。
- パフォーマンス — Amazon Keyspaces は低レイテンシーのデータアクセスを提供するため、アプリケーションは並外れた速度でデータを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、 が提供するフルマネージドサービスです AWS。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。前払いのキャパシティプロビジョニングを必要とせずに消費されたキャパシティに対してのみ料金が発生します。管理するサーバーまたは選択するインスタンスがありません。この pay-per-request モデルは、容量をプロビジョニングしてモニタリングすることなく、消費したリソースに対してのみ料金を支払うため、コスト効率と運用上のオーバーヘッドを最小限に抑えます。
- スキーマによる NoSQL の柔軟性 — Amazon Keyspaces は NoSQL データモデルに従い、スキーマ設計に柔軟性を提供します。Amazon Keyspaces では、構造化、半構造化、非構造化データを保存できるため、多様で進化するデータ型の処理に適しています。さらに、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 は、最大 99.999% の可用性 SLA を提供します。Amazon

Keyspaces では、より耐障害性と低レイテンシーのローカル読み取りを実現するために、マルチリージョンレプリケーション を提供しています。

- セキュリティとコンプライアンス — Amazon Keyspaces は と統合され、きめ細かなアクセスコントロール AWS Identity and Access Management を実現します。保管時と転送時の暗号化を提供し、データのセキュリティを向上させます。Amazon Keyspaces は、HIPAA、PCI DSS、GDPR などのセキュリティ標準とプライバシー法にも準拠しているため、規制要件を満たすことができます。
- AWS エコシステムとの統合 – AWS エコシステムの一部として、Amazon Keyspaces は AWS のサービス、Amazon AWS CloudFormation、 などの他の CloudWatchとシームレスに統合します 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。

Amazon Keyspaces への移行に関する一般的な考慮事項

- 移行を小さな要素に分割します。

  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のデータを移行すると、移行が簡単になる場合があります。

  - 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。
- 簡易性に基づいて最初に移行するデータに優先順位を付けます。

  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

- Apache Cassandra からのデータの移行に関するガイダンス
- Amazon Keyspaces にデータを移行するためのツール

Apache Cassandra からのデータの移行に関するガイダンス

Apache Cassandra から Amazon Keyspaces への移行を成功させるには、利用可能なオプションを慎重に計画して比較することをお勧めします。このトピックでは、移行プロセスの仕組み、利用可能なツール、さまざまな移行戦略を評価して、要件に最も適した移行戦略を選択する方法について説明します。

トピック

機能の互換性

移行前に、Apache Cassandra と Amazon Keyspaces の機能の違いを慎重に検討してください。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 APIs がいくつかあります。サポートされている APIs「」を参照してください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”。Amazon Keyspaces と Apache Cassandra のすべての機能の違いの概要については、「」を参照してくださいthe section called “Apache Cassandra との機能の違い”。

使用している Cassandra APIs とスキーマを Amazon Keyspaces でサポートされている機能と比較するには、 の Amazon Keyspaces ツールキットで利用可能な互換性スクリプトを実行できますGitHub。

互換性スクリプトの使用方法

1. 互換性がある Python スクリプトを からダウンロードGitHubし、既存の Apache Cassandra クラスターにアクセスできる場所に移動します。
2. 互換性スクリプトは、 と同様のパラメータを使用しますCQLSH。`--host` と には、クラスター内のいずれかの Cassandra ノードに接続してクエリを実行するために使用する IP アドレスとポート`--port`を入力します。Cassandra クラスターで認証を使用する場合は、 `-username`とも指定する必要があります`-password`。互換性スクリプトを実行するには、次のコマンドを使用できます。

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

Amazon Keyspaces の料金を見積もる

このセクションでは、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の概要を説明します。テーブルごとに異なるデータ型が必要で、異なる CQL クエリをサポートし、固有の読み取り/書き込みトラフィックを維持する必要があります。テーブルに基づいて要件を考慮すると、Amazon Keyspaces のテーブルレベルのリソース分離および読み取り/書き込みスループットキャパシティモードと一致します。Amazon Keyspaces では、テーブルの読み取り/書き込み容量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ると、機能、コスト、移行作業に基づいて移行テーブルの優先順位を付けることができます。

移行前に、次の Cassandra テーブルメトリクスを収集します。この情報は、Amazon Keyspaces のワークロードのコストを見積もるのに役立ちます。

- テーブル名 – 完全修飾キースペースの名前とテーブル名。
- 説明 – テーブルの使用方法や、テーブルに格納されるデータの種類など、テーブルの説明。
- 1 秒あたりの平均読み取り数 — 大きな時間間隔におけるテーブルに対する座標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 — 大きな時間間隔におけるテーブルに対する座標レベルの書き込みの平均数。
- バイト単位の平均行サイズ — バイト単位の平均行サイズ。
- GB GBsストレージサイズ — 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) と強力な整合性 () を使用する読み取りの割合LOCAL_QUORUM。

この表は、移行を計画する際にまとめる必要があるテーブルに関する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	バイト単位の平均行サイズ	GB GBsストレージサイズ	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に使用されます	10,000	5,000	2,200	2,000	100% LOCAL_ONE
mykeyspace.mytable2	最新のプロファイル情報を保存するために使用されます	20,000	1,000	850	1,000	25% LOCAL_QUORUM 75% LOCAL_ONE

テーブル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方法をステップバイステップで説明します。これらのメトリクスには、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 (RPS) が含まれます。これにより、Amazon Keyspaces テーブルのスループットキャパシティ要件を評価し、料金を見積もることができます。

Cassandra ソーステーブルでテーブルメトリクスを収集する方法

1. 行サイズを決定する

   行サイズは、Amazon Keyspaces の読み取り容量と書き込み容量の使用率を決定するために重要です。次の図は、Cassandra トークン範囲における一般的なデータ分散を示しています。

で使用可能な行サイズのサンプラスクリプトを使用してGitHub、Cassandra クラスター内の各テーブルの行サイズのメトリクスを収集できます。このスクリプトは、 cqlshと を使用して、設定可能なテーブルデータのサンプルセットにおける行サイズの最小、最大、平均、標準偏差awkを計算することで、Apache Cassandra からテーブルデータをエクスポートします。行サイズのサンプラーは引数を に渡すためcqlsh、Cassandra クラスターへの接続と読み取りに同じパラメータを使用できます。

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 での行サイズの計算方法の詳細については、「」を参照してくださいthe section called “行サイズの計算”。

2. テーブルサイズを決定する

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はありません。Amazon Keyspaces は、テーブルの請求対象サイズを継続的にモニタリングして、ストレージ料金を決定します。ストレージは GB/月ごとに請求されます。Amazon Keyspaces のテーブルサイズは、1 つのレプリカの raw サイズ (非圧縮) に基づいています。Amazon Keyspaces のテーブルサイズをモニタリングするには、メトリクス を使用できます。メトリクス はBillableTableSizeInBytes、 の各テーブルに表示されます AWS Management Console。

Amazon Keyspaces テーブルの請求対象サイズを見積もるには、次の 2 つの方法のいずれかを使用できます。

- 平均行サイズを使用し、数値または行を乗算します。

  Amazon Keyspaces テーブルのサイズを見積もるには、平均行サイズに Cassandra ソーステーブルの行数を掛けます。前のセクションの行サイズのサンプルスクリプトを使用して、平均行サイズをキャプチャします。行数をキャプチャするには、 などのツールを使用してdsbulk count、ソーステーブルの行の合計数を決定できます。
- を使用してテーブルメタデータnodetoolを収集します。

  Nodetool は、Apache Cassandra ディストリビューションで提供される管理ツールで、Cassandra プロセスの状態に関するインサイトを提供し、テーブルメタデータを返します。を使用してnodetool、Amazon Keyspaces のテーブルサイズを推定する および に関するメタデータをサンプリングできます。使用するコマンドは ですnodetool tablestats。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tablelivespaceの として保存され、 で分割できますcompression ratio。次に、このサイズ値をノード数で倍します。最後に、レプリケーション係数 (通常は 3) で割ります。これは、テーブルサイズの評価に使用できる計算の完全な式です。

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

  Cassandra クラスターに 12 個のノードがあるとします。nodetool tablestats コマンドを実行すると、200 GB tablelivespaceの と 0.5 compression ratioの が返されます。キースペースのレプリケーション係数は 3 です。この例の計算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. 読み取りと書き込みの数をキャプチャする

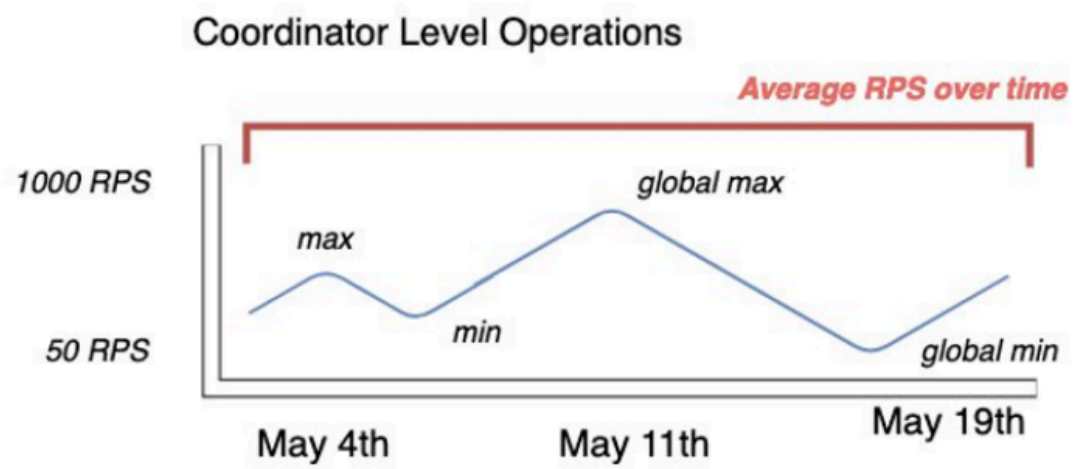
   Amazon Keyspaces テーブルの容量とスケーリング要件を決定するには、移行前に Cassandra テーブルの読み取りおよび書き込みリクエストレートをキャプチャします。

   Amazon Keyspaces はサーバーレスで、使用した分のみお支払いいただきます。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います。Amazon Keyspaces には、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 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

ティモードは、キャパシティプランニングを必要とせずに 1 秒あたり数千件のリクエストに対応できる柔軟な請求オプションです。このオプションでは、読み取りリクエストと書き込みリクエストの pay-per-request 料金が提供されるため、使用した分に対してのみ料金が発生します。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 の使用量を管理することで、定義済みのリクエストレート以下を維持して料金を最適化し、予測可能性を維持することができます。プロビジョンドモードは、自動スケーリング機能を提供し、プロビジョニングされたレートを自動的に調整してスケールアップまたはスケールダウンし、運用効率を向上させます。サーバーレスリソース管理の詳細については、「」を参照してくださいサーバーレスリソース管理。

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、既存のテーブルで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、単一のデータセンター内のすべてのノードに集約されたテーブルに対して、長時間にわたるコーディネーターレベルの読み取りおよび書き込みオペレーションに対する 1 秒あたりの平均リクエスト数 (RPS) をキャプチャします。少なくとも数週間の平均 RPS をキャプチャすると、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がキャプチャされます。



Cassandra テーブルの読み取りおよび書き込みリクエストレートを決定するには、2 つのオプションがあります。

- 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、読み取りおよび書き込みリクエストを監視できます。メトリクス名は、使用しているモニタリングツールに基づいて変更される可能性があることに注意してください。

ディメンション	Cassandra JMX メトリクス
書き込み	`org.apache.cassandra.metrics:type=ClientRequest,scope=Write,name=Latency#Count`
読み取り	`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、テーブルからの平均読み取りおよび書き込みオペレーションをキャプチャします。 は、ノードが開始された時点からの読み取りおよび書き込みの合計数tablestatsを返します。 nodetool info は、ノードの稼働時間を秒単位で提供します。読み取りと書き込みの 1 秒あたりの平均を受け取るには、読み取りと書き込みの数をノードの稼働時間を秒単位で割ります。次に、読み取りの場合は、書き込みの整合性レベルの広告で除算し、レプリケーション係数で除算します。これらの計算は、次の式で表されます。

  1 秒あたりの平均読み取り数の計算式：

  ```
  ((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計算式：

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

4 週間稼働している 12 個のノードクラスターがあるとします。 は 2,419,200 秒の稼働時間をnodetool infoを返し、10 億回の書き込みと 20 億回の読み取りnodetool tablestatsを返します。この例では、次の計算が行われます。

```
((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. テーブルの容量使用率を確認する

平均容量使用率を見積もるには、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から始めます。

Amazon Keyspaces は、読み取りキャパシティーユニット (RCUsと書き込みキャパシティーユニット (WCUsを使用して、テーブルの読み取りと書き込みの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。この見積もりでは、これらのユニットを使用して、移行後の新しい Amazon Keyspaces テーブルの読み取りおよび書き込み容量のニーズを計算します。このトピックの後半では、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選択が請求にどのように影響するかについて説明します。ただし、容量使用率の見積もりについては、テーブルがプロビジョニングモードであると仮定します。

1 つの RCU は、最大 4 KB の行に対して 1 つの読み取りLOCAL_QUORUMリクエスト、または 2 つのLOCAL_ONE読み取りリクエストを表します。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 RCU が使用されます。必要な RCUs の合計数は、行サイズ、および整合性を使用するかLOCAL_QUORUMLOCAL_ONE読み取りするかによって異なります。例えば、8 KB の行を読み取るには、LOCAL_QUORUM読み取り整合性を使用して 2 RCUs が必要であり、LOCAL_ONE読み取り整合性を選択した場合は 1 つの RCU が必要です。

1 つの WCU は、最大 1 KB のサイズの行に対して 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 回の書き込みリクエストを実行するには 2 WCUsが必要です。

次の式を使用して、必要な RCUs と WCUs推定できます。RCUs の読み込み容量は、1 秒あたりの読み込み数に読み込みあたりの行数を平均行サイズで乗算して 4KB にし、最も近い整数に切り上げることで決定できます。

WCUs の書き込み容量は、リクエスト数に平均行サイズを 1KB で割って、最も近い整数に切り上げることで決定できます。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、Cassandra テーブルで行サイズが 2.5 KB の読み取りリクエストを 4,960 件実行する場合、Amazon Keyspaces には 4,960 RCUs が必要です。現在、Cassandra テーブルで行サイズが 2.5 KB の 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は、Amazon Keyspaces で 1 秒あたり 4,959 WCUs が必要です。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` を使用すると、各読み取りリクエストのスループット容量の最大半分を節約できます。結果整合性のある各読み込みは、最大 8KB消費できます。次の式に示すように、前の計算に 0.5 を掛けることで、結果整合性のある読み込みを計算できます。

```
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 の月額料金見積りを計算する

   読み取り/書き込みキャパシティのスループットに基づいてテーブルの月額請求額を見積もるには、さまざまな式を使用してオンデマンドモードとプロビジョニングモードの料金を計算し、テーブルのオプションを比較します。

プロビジョンドモード – 読み取りおよび書き込み容量の消費量は、1 秒あたりの容量単位に基づいて時間単位の料金で請求されます。まず、そのレートを 0.7 で除算して、デフォルトの自動スケーリングターゲット使用率である 70% を表します。次に、30 暦日、1 日あたり 24 時間、およびリージョン別料金で倍増します。この計算は、次の式にまとめられています。

```
(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 – 読み取りおよび書き込み容量は、リクエストごとのレートで請求されます。まず、リクエストレートに 30 暦日、1 日あたり 24 時間掛けます。次に、100 万リクエストユニットで割ります。最後に、リージョン別レートを掛けます。この計算は、次の式にまとめられています。

```
((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
```

移行戦略を選択します

一般に、Apache Cassandra から Amazon Keyspaces に移行するときに、次の 3 つの異なる移行戦略から選択できます。

- オフライン – この移行では、ブルー/グリーンスタイルのアプリケーション移行デプロイを使用して、Cassandra から Amazon Keyspaces にデータセットをコピーします。移行中にアプリケーションがある程度のダウンタイムを許容できる場合、このオプションにより移行プロセスを簡素化できます。オフライン移行の詳細については、「」を参照してください。

  the section called “オフライン移行”.
- オンライン – これは Canary スタイルのデプロイで、通常、アプリケーションロジックに直接書き込まれる二重書き込みが含まれます。移行中にダウンタイムなしで済むアプリケーションでは、ライブの読み取りと書き込みがデータソース間で切り替えられている間、 にコピーされたデータが必要です。
- ハイブリッド — このアプローチでは、変更をほぼリアルタイムでレプリケートできますが、読み取りと書き込みを切り替えるのはアプリケーションの責任です。

利用可能な移行戦略をより詳細に確認したら、要件と利用可能なリソースを考慮して、決定木にオプションを配置してプロセスを簡素化できます。

Amazon Keyspaces へのオフライン移行

オフライン移行は、移行を実行するためのダウンタイムを許容できる場合に適しています。パッチ適用、大規模なリリース、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のダウンタイムのためのメンテナンスウィンドウを設けることは、企業間で一般的です。オフライン移行では、このウィンドウを使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。オフライン移行では、Cassandra と Amazon Keyspaces の両方に同時に通信する必要がないため、アプリケーションへの変更が減ります。さらに、データフローを一時停止すると、ミューテーションを維持せずに正確な状態をコピーできます。

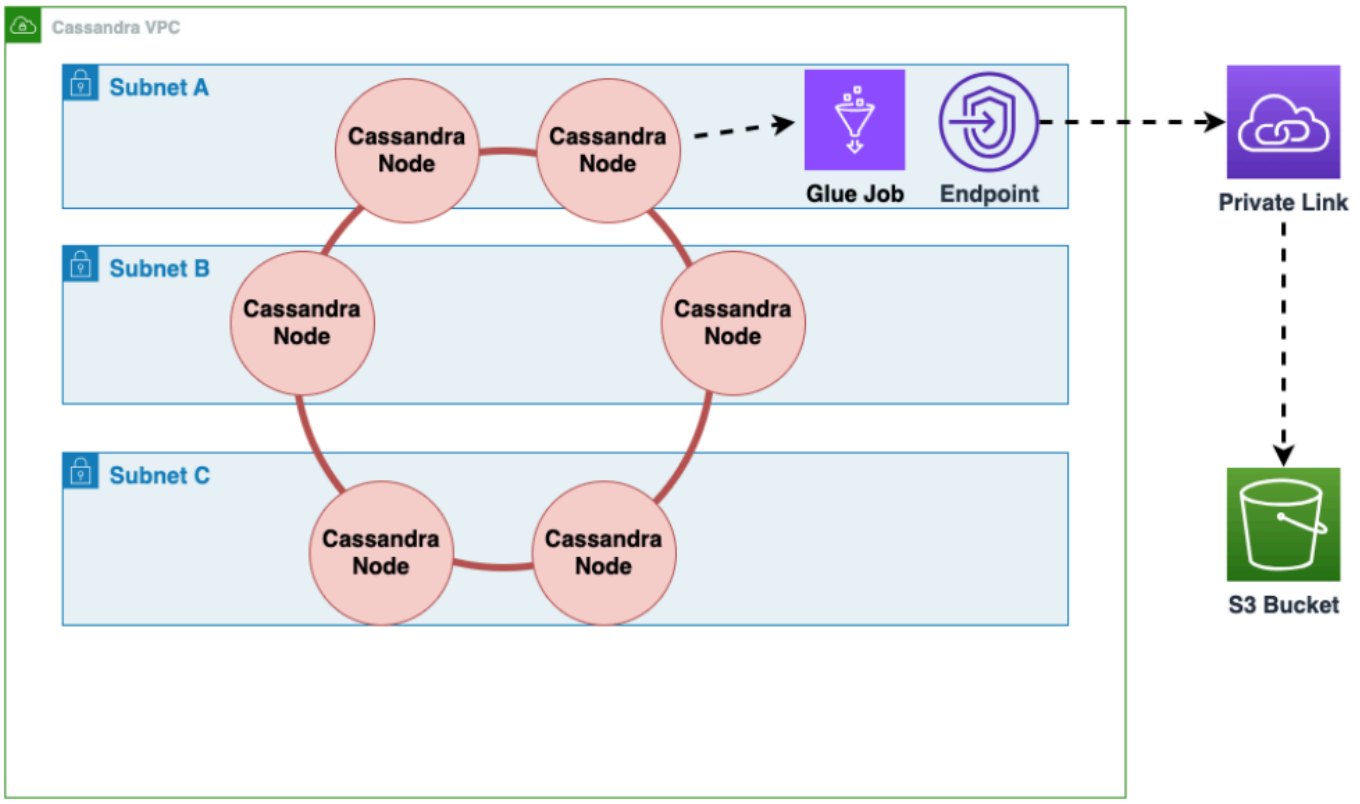
この例では、ダウンタイムを最小限に抑えるために、オフライン移行中のデータのステージング領域として Amazon Simple Storage Service (Amazon S3) を使用します。Spark Cassandra コネクタとを使用して、Amazon S3 で Parquet 形式で保存したデータを Amazon Keyspaces テーブルに自動的にインポートできます AWS Glue。次のセクションでは、プロセスの概要を示します。このプロセスのコード例は、Github にあります。

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 番目のジョブ。
3. 増分データをインポートする 3 番目のジョブ。

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を実行する方法

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。次に、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、これらのステップを示しています。



2. Amazon S3 バケット内のデータをシャッフルして、データのランダム化を改善します。均等にインポートされたデータにより、ターゲットテーブルに分散されたトラフィックが増えます。このステップは、Amazon Keyspaces にデータを挿入する際のホットキーパターンを避けるために、大きなパーティション (1000 行を超えるパーティション) を持つ Cassandra からデータをエクスポートする場合に必要です。Amazon Keyspaces `WriteThrottleEvents`でホットキーの問題が発生すると、ロード時間が長くなります。

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。Amazon S3 バケット内のシャッフルされたデータは Parquet 形式で保存されます。

Amazon Keyspaces にデータを移行するためのツール

Amazon Keyspaces にデータを移行するためのさまざまなツールが利用可能に

- 移行ツール
  - 大規模な移行の場合は、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。
  - 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、「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。 step-by-step 手順については、「」を参照してくださいthe section called “cqlsh を使用したデータのロード”。
  - また、Apache Cassandra 用 DataStax Bulk Loader を使用して、 `dsbulk` コマンドを使用して Amazon Keyspaces にデータをロードすることもできます。DSBulk は cqlsh よりも堅牢なインポート機能を提供し、GitHub リポジトリ から利用できます。 step-by-step 手順については、「」を参照してくださいthe section called “DSBulk を使用したデータのロード”。

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 step-by-step チュートリアルでは、 `cqlsh COPY` 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説明します。このチュートリアルでは、以下の作業を行います。

トピック

前提条件

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 “コンソールを使用してサービス固有の認証情報を生成する” 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: 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 cqlsh 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

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 “行サイズの計算”」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工コーバックします。

   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。容量不足の例外の診断に役立つように、Amazon Keyspaces は WriteThrottleEventsおよびReadThrottledEventsメトリクスを Amazon に発行します CloudWatch。詳細については、「the section called "によるモニタリング CloudWatch"」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

step-by-step このチュートリアルでは、 DataStax で利用できるバルクローダー (DSBulk) を使用して Apache Cassandra から Amazon Keyspaces にデータを移行する方法を説明します。GitHubこのチュートリアルでは、次の手順を実行します。

トピック

- ステップ 2: 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: DSBulk を設定する
- ステップ 5: DSBulk load コマンドを実行する

前提条件

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. まだ行っていない場合は、の手順に従ってアカウントにサインアップしてください。 AWS the section called “セットアップ AWS Identity and Access Management”
2. the section called “認証用の IAM 認証情報 AWS ” 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、jarGitHubシェードファイルをからダウンロードして 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: 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”」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: 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 “行サイズの計算”」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: 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: 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

"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、local-datacenter AWS リージョン 接続先のものに設定します。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情報については 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。バッチモードを無効にすることをおすすめします。バッチモードはホットキーなシナリオや原因となる可能性があるからですWriteThrottleEvents。
   - *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”」を参照してください。

AWS SDKs を使用した Amazon Keyspaces のコード例

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。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

AWS SDK デベロッパー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 Amazon Keyspaces 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

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスListKeyspaces」の「」を参照してください。AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスListKeyspaces」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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ListKeyspacesの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 ListKeyspaces AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

コードの例

- AWS SDKs を使用した Amazon Keyspaces のアクション
  - AWS SDK または CLI CreateKeyspaceで を使用する
  - AWS SDK または CLI CreateTableで を使用する
  - AWS SDK または CLI DeleteKeyspaceで を使用する
  - AWS SDK または CLI DeleteTableで を使用する
  - AWS SDK または CLI GetKeyspaceで を使用する
  - AWS SDK または CLI GetTableで を使用する
  - AWS SDK または CLI ListKeyspacesで を使用する
  - AWS SDK または CLI ListTablesで を使用する
  - AWS SDK または CLI RestoreTableで を使用する
  - AWS SDK または CLI UpdateTableで を使用する
- AWS SDKs を使用する Amazon Keyspaces のシナリオ
  - AWS SDK を使用して Amazon Keyspaces キースペースとテーブルの使用を開始する

AWS SDKs を使用した Amazon Keyspaces のアクション

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。これらは Amazon Keyspaces API を呼び出すもので、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。各例には GitHub、コードの設定と実行の手順を示す へのリンクが含まれています。

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 または CLI CreateKeyspaceで を使用する

- AWS SDK または CLI CreateTableで を使用する
- AWS SDK または CLI DeleteKeyspaceで を使用する
- AWS SDK または CLI DeleteTableで を使用する
- AWS SDK または CLI GetKeyspaceで を使用する
- AWS SDK または CLI GetTableで を使用する
- AWS SDK または CLI ListKeyspacesで を使用する
- AWS SDK または CLI ListTablesで を使用する
- AWS SDK または CLI RestoreTableで を使用する
- AWS SDK または CLI UpdateTableで を使用する

AWS SDK または CLI **CreateKeyspace**で を使用する

以下のコード例は、CreateKeyspace の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスCreateKeyspace」の「」を参照してください。 AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスCreateKeyspace」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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CreateKeyspaceの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、CreateKeyspaceAWS 「 SDK for Python (Boto3) API リファレンス」の「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 または CLI `CreateTable`で を使用する

以下のコード例は、CreateTable の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスCreateTable」の「」を参照してください。 AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスCreateTable」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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CreateTableの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 CreateTable AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 または CLI `DeleteKeyspace`で を使用する

以下のコード例は、DeleteKeyspace の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスDeleteKeyspace」の「」を参照してください。 AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスDeleteKeyspace」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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DeleteKeyspaceの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 DeleteKeyspace AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 または CLI **DeleteTable**で を使用する

以下のコード例は、DeleteTable の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスDeleteTable」の「」を参照してください。AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスDeleteTable」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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DeleteTableの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 DeleteTable AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 または CLI **GetKeyspace**で を使用する

以下のコード例は、GetKeyspace の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスGetKeyspace」の「」を参照してください。 AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスGetKeyspace」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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GetKeyspaceの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 GetKeyspace AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 または CLI **GetTable**で を使用する

以下のコード例は、GetTable の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスGetTable」の「」を参照してください。 AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスGetTable」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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GetTableの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 GetTable AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 または CLI `ListKeyspaces`で を使用する

以下のコード例は、`ListKeyspaces` の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスListKeyspaces」の「」を参照してください。AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスListKeyspaces」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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ListKeyspacesの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 ListKeyspaces AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 または CLI `ListTables`で を使用する

以下のコード例は、ListTables の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスListTables」の「」を参照してください。 AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスListTables」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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ListTablesの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 ListTables AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 または CLI **RestoreTable**で を使用する

以下のコード例は、RestoreTable の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスRestoreTable」の「」を参照してください。 AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスRestoreTable」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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RestoreTableの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 RestoreTable AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 または CLI **UpdateTable**で を使用する

以下のコード例は、UpdateTable の使用方法を示しています。

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。

- キースペースとテーブルの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスUpdateTable」の「」を参照してください。 AWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。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、「 API リファレンスUpdateTable」の「」を参照してください。 AWS SDK for Java 2.x

Kotlin

SDK for Kotlin

> **Note**
>
> については、「」を参照してください GitHub。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 { 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UpdateTableの「」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。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、 UpdateTable AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパー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 Amazon Keyspaces 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 を使用する Amazon Keyspaces のシナリオ

次のコード例は、 AWS SDKs を使用して Amazon Keyspaces で一般的なシナリオを実装する方法を示しています。これらのシナリオは、Amazon Keyspaces 内で複数の関数を呼び出すことによって特定のタスクを実行する方法を示しています。各シナリオには GitHub、コードの設定と実行の手順を示す へのリンクが含まれています。

例

- AWS SDK を使用して Amazon Keyspaces キースペースとテーブルの使用を開始する

AWS SDK を使用して Amazon Keyspaces キースペースとテーブルの使用を開始する

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。テーブルスキーマは映画データを保持し、 point-in-time 復旧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

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。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**
>
> については、「」を参照してください GitHub。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**
>
> については、「」を参照してください GitHub。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 { 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 { 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 { 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 { 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 { 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 { 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 { 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 { 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 { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.createTable(tableRequest)
        println("The table ARN is ${response.resourceArn}")
    }
}

suspend fun listKeyspacesPaginator() {
    KeyspacesClient { 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 { 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 { 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**
>
> については、「」を参照してください GitHub。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 デベロッパー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 Amazon Keyspaces 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

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、オプションの cqlsh 用 AWS 認証拡張子、一般的なタスクを実行するためのヘルパーツールなどが含まれています。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

このプラグインでは、AWS Identity and Access Management (IAM) のユーザーとロールを使用することで 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

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 AWS CloudFormation テンプレート

このリポジトリには、Amazon Keyspaces 向けとして CloudWatch メトリクスをすばやく設定できる AWS CloudFormation テンプレートがあります。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.

AWS Lambda と Amazon Keyspaces (Apache Cassandra 向け) の使用

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

AWS Glue と Amazon Keyspaces (Apache Cassandra 向け) の使用

これは AWS Glue 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/scala/datastax-v4/aws-glue

Amazon Keyspaces (Apache Cassandra 向け) Cassandra クエリ言語 (CQL) を AWS CloudFormation に変換するコンバーター

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 と Apache Spark の統合

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

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-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 “auto スケーリングによるスループット容量の管理”」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”」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。まだ行っていない場合は、 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) を生成し、 AWS 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"」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”」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 “によるモニタリング CloudWatch”」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。

Amazon Keyspaces (Apache Cassandra 向け) のトラブルシューティング

次のセクションでは、Amazon Keyspaces (Apache Cassandra 向け) の使用時に発生する可能性がある設定全般の問題を解決する方法について説明します。

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 での接続に関するトラブルシューティング
- Amazon Keyspaces でのキャパシティ管理に関するトラブルシューティング
- Amazon Keyspaces のデータ定義言語に関する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する 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

さまざまな理由で発生する可能性がある以下の最上位の例外のいずれかが表示されます。

- `NoNodeAvailableException`
- `NoHostAvailableException`
- `AllNodesFailedException`

これらの例外はクライアントドライバーによって生成され、制御接続を確立するとき、または読み取り/書き込み/準備/実行/バッチリクエストを実行するとき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発生すると、アプリケーションで指定されたすべてのコンタクトポイントに到達できないことを示します。読み取り/書き込み/準備/実行クエリの実行中にエラーが

発生すると、そのリクエストのすべての再試行が使い果たされたことを示します。デフォルトの再試行ポリシーを使用している場合、各再試行は別のノードで試行されます。

基になるエラーを最上位の Java ドライバーの例外から分離する方法

これらの一般的なエラーは、接続の問題、または読み取り/書き込み/準備/実行オペレーションの実行中に発生する可能性があります。一時的な障害は分散システムで予期する必要があり、リクエストを再試行して処理する必要があります。Java ドライバーは、接続エラーが発生したときに自動的に再試行しないため、アプリケーションでドライバー接続を確立するときに再試行ポリシーを実装することをお勧めします。接続のベストプラクティスの詳細な概要については、「」を参照してくださいthe section called “接続”。

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 false idempotenceに設定します。つまり、Java ドライバーは失敗した読み取り/書き込み/準備リクエストを自動的に再試行しません。idempotence を に設定trueし、失敗したリクエストを再試行するようにドライバーに指示するには、いくつかの異なる方法で実行できます。以下は、Java アプリケーションで 1 つのリクエストに対して冪等性をプログラムで設定する方法の例です。

```
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冪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
```

もう 1 つの推奨事項は、アプリケーションレベルで再試行ポリシーを作成することです。この場合、アプリケーションは をキャッチNoNodeAvailableExceptionしてリクエストを再試行する必要があります。エクスポネンシャルバックオフを 10 ミリ秒から開始し、最大 100 ミリ秒まで動作し、すべての再試行で合計 1 秒の再試行を行うことをお勧めします。

もう 1 つのオプションは、Github で使用できる Java ドライバー接続を確立するときに Amazon Keyspaces エクスポネンシャル再試行ポリシーを適用することです。

デフォルトの再試行ポリシーを使用するときに、複数のノードへの接続が確立されていることを確認します。これを行うには、Amazon Keyspaces で次のクエリを使用します。

```
SELECT * FROM system.peers;
```

このクエリのレスポンスが空の場合、Amazon Keyspaces の単一ノードを使用していることを示します。デフォルトの再試行ポリシーを使用している場合、デフォルトの再試行は常に別のノードで発生するため、再試行は行われません。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、「」を参照してくださいthe section called “VPC エンドポイント接続”。

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示す step-by-step チュートリアルについては、「」を参照してくださいthe section called “Java 4.x の認証プラグイン”。

Amazon Keyspaces での接続に関するトラブルシューティング

接続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。

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 エンドポイントとの接続” 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みます。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と、Amazon EC2 インスタンスが正しく設定されません。

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、 を使用して AWS CloudShell に接続しますcqlsh。その接続が確立された場合は、Amazon EC2 インスタンスを設定する必要があります。

この問題を解決するには、Amazon EC2 インスタンスに 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があることを確認します。そうでない場合は、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、Type ドロップダウンメニューから CQLSH/CASSANDRA を選択します。

アウトバウンドトラフィックルールを使用して新しいセキュリティグループを作成したら、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の変更 を選択します。新しいセキュリティグループをアウトバウンドルールに追加しますが、デフォルトのグループも引き続き使用できるようにします。

EC2 アウトバウンドルールを表示および編集する方法の詳細については、Amazon EC2」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、 AWS Support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していますが、**NodeUnavailableException**エラーが表示されます。

リクエストが試行され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、ハートビート値を見つけ、それが大きい場合は 30 秒に下げます。

```
advanced.heartbeat.interval = 30 seconds
```

次に、関連するタイムアウトを探し、値が少なくとも 5 秒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 “認証用の IAM 認証情報 AWS ”」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 "認証用の IAM 認証情報 AWS "」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。

これは、間違った認証情報を使用し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。

```
cassandra.cluster.NoHostAvailable: ('Unable to connect to any servers',
 {'44.234.22.134:9142':
 AuthenticationFailed('Failed to authenticate to 44.234.22.134:9142: Error from server:
 code=0100
 [Bad credentials] message="Authentication failure: Invalid signature"')})
```

この問題を解決するには、渡す認証情報が、Amazon Keyspaces にアクセスするように設定した IAM ユーザーまたはロールに関連付けられていること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 "認証用の IAM 認証情報 AWS "」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 のインターフェイス VPC エンドポイント経由で接続を設定する step-by-step 方法については、「」を参照してくださいthe section called "VPC エンドポイントとの接続"。

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 個のノードを返します。

```
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 でのキャパシティ管理に関する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。

Cassandra エラーを基盤となるキャパシティイベントにマッピング CloudWatch するには、Amazon を使用して関連する Amazon Keyspaces メトリクスをモニタリングできます。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”」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 の接続を設定する方法については、「」を参照してくださいthe section called “接続を設定する方法”。VPC エンドポイントを介した接続の最適化の詳細については、「」を参照してくださいthe section called “VPC エンドポイント接続”。

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、コンソールは、テーブルのキャパシティタブのキャパシティおよび関連 CloudWatch メトリクスセクションで、最も一般的なキャパシティおよびトラフィック関連のメトリクスを集約したビューを提供します。

Amazon を使用して独自のダッシュボードを作成するには CloudWatch、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。

の詳細については、 CloudWatch「」を参照してくださいthe section called “によるモニタリング CloudWatch”。Amazon Keyspaces で使用できるすべての CloudWatch メトリクスのリストについては、「」を参照してくださいthe section called “メトリクスとディメンション”。

> **Note**
>
> Amazon Keyspaces で一般的に観察されるすべてのメトリクスを表示するカスタムダッシュボードの使用を開始するには、AWS サンプルリポジトリの GitHub で利用できる構築済み 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、プロビジョニングされたスループットメトリクスと実際のスループットメトリクス、およびテーブルの容量不足イベントをモニタリングします。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。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 “auto スケーリングによるスループット容量の管理”」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。

エラーが表示されるとStoragePartitionThroughputCapacityExceeded、パーティション容量が一時的に超過します。これは、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場合があります。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す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。

次のいずれかの原因PerConnectionRequestRateExceededが原因で が表示されます。

- 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、ドライバーがピアノード (DataStax Java ドライバーなど) の SSL ホスト名の検証を試みている場合、または VPC エンドポイント経由で接続している場合に発生する可能性があります。

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。
- 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。容量不足の例外を診断しやすくするために、Amazon Keyspaces は Amazon に次のメトリクスを発行します CloudWatch。

- `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、GitHub リポジトリ から入手できる DSBulk の使用を検討してください。手順については step-by-step、「」を参照してくださいthe section called “DSBulk を使用したデータのロード”。

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、「」を参照してくださいthe section called “テーブルレベルでコストを評価する”。テーブル内の行サイズの計算を開始することで、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 “行サイズの計算”」を参照してください。

Amazon Keyspaces のデータ定義言語に関する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。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 Recovery point-in-time (PITR) を使用してテーブルを復元しようとしていますが、復元は失敗します

- 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”」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”」を参照してください。

Amazon Keyspaces Recovery point-in-time (PITR) を使用してテーブルを復元しようとしていますが、復元は失敗します

Amazon Keyspaces テーブルを point-in-time 復旧 (PITR) で復元しようとしたときに、復元プロセスが開始され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がすべて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 “復元許可”」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 “カスタムプロパティを使用して既存のテーブルの有効期限 (TTL) を有効にする方法”」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) でのサーバーレスリソース管理

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。

この章では、Amazon Keyspaces でのサーバーレスリソース管理について詳しく説明します。Amazon 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 CloudWatch、「」を参照してくださいthe section called “によるモニタリング CloudWatch”。

トピック

- Amazon Keyspaces のストレージ
- Amazon Keyspaces での読み取り/書き込みキャパシティモード
- Amazon Keyspaces auto スケーリングでスループット容量を自動的に管理します
- Amazon キKeyspaces でバースト容量を効果的に使用する
- Amazon Keyspaces 容量消費量を推定する方法

Amazon Keyspaces のストレージ

Amazon Keyspaces (Apache Cassandra 向け) では、テーブルに保存されている実際のデータに基づいて、テーブルにストレージが自動的にプロビジョニングされます。あらかじめテーブルにストレージをプロビジョニングしておく必要はありません。Amazon Keyspaces では、アプリケーションによりデータの書き込み、更新、削除が実行されるときに、テーブルストレージのスケールが自動で調整されます。従来の Apache Cassandra クラスターとは異なり、Amazon Keyspaces は、圧縮などといった低レベルのシステムオペレーションをサポートするために追加のストレージを必要としません。使用したストレージに対してのみ課金されます。

Amazon Keyspaces では、デフォルトで、レプリケーション係数が 3 であるキースペースが設定されます。このレプリケーション係数は変更できません。Amazon Keyspaces は、高可用性を実現するために、テーブルデータを複数の AWS アベイラビリティーゾーンに自動的に 3 回レプリケートします。Amazon Keyspaces ストレージの GB あたりの料金には、あらかじめレプリケーション料が含まれています。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 “行サイズの計算”」を参照してください。

Amazon Keyspaces での読み取り/書き込みキャパシティモード

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

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。このオプションは、読み取りおよび書き込みリクエストの pay-per-request 料金を提供するため、使用した分のみお支払いいただきます。

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

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および書き込み容量の消費量を見積もる方法については、「」を参照してくださいthe section called “キャパシティ消費量の見積もり”。

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性能モード

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 の使用量を管理することで、定義済みのリクエストレート以下を維持して料金を最適化し、予測可能性を維持することができます。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 “auto スケーリングによるスループット容量の管理”」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。
- キャパシティの要件を予測して価格を最適化することができます。

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および書き込み容量の消費量を見積もる方法の詳細については、「」を参照してくださいthe section called “キャパシティ消費量の見積もり”。

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 “行サイズの計算”」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 を使用して CloudWatch 、プロビジョニングされたスループットメトリクスと実際のスループットメトリクス、および容量不足イベントをモニタリングできます。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**
>
> 容量不足によるエラーが繰り返し発生すると、 DataStax Java ドライバーが で失敗するなど、クライアント側のドライバー固有の例外が発生する可能性がありますNoHostAvailableException。

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。

キャパシティモードの管理および表示

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。

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。

テーブルのプロビジョンドスループット性能を変更するには、the section called “ALTER TABLE” を使用します。

キャパシティモードの変更時の考慮事項

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。

> **Note**
>
> キャパシティモードをプロビジョニングモードからオンデマンドモー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。

Amazon Keyspaces auto スケーリングでスループット容量を自動的に管理します

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

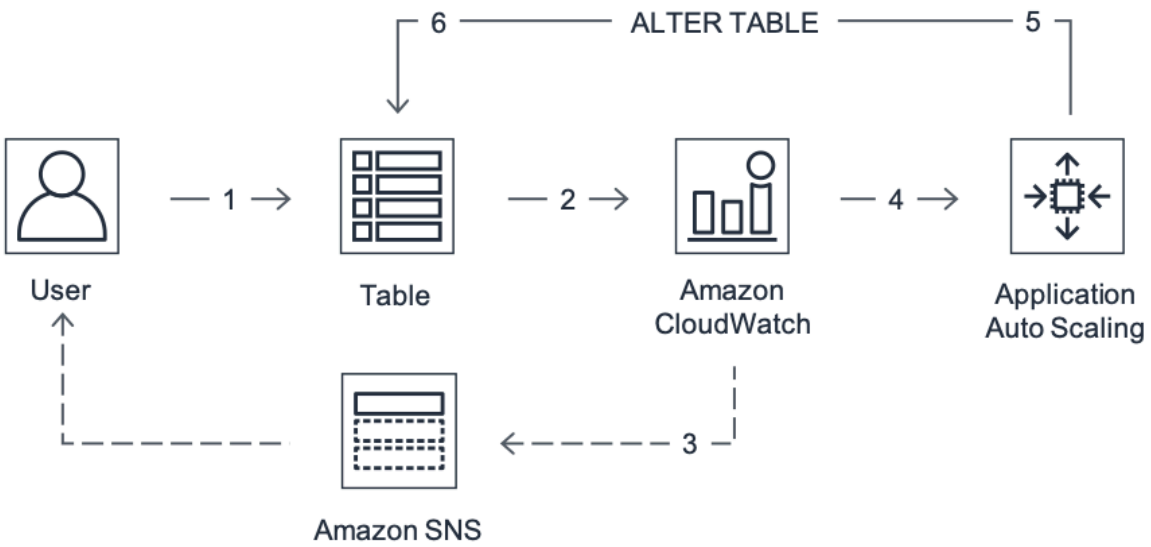
を感じるようになります。この種のワークロードでは多くの場合、手動介入によってデータベースリソースを使用レベルに応じて上下させる必要があります。

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 “コンソールを使用する場合”」を参照してください。Cassandra クエリ言語 (CQL) を使用して Amazon Keyspaces のスケーリングポリシーを管理するには、を参照してください。the section called “CQL の使用”CLI を使用して Amazon Keyspaces のスケーリングポリシーを管理する方法については、を参照してくださいthe section called “CLI の使用”。

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。

スケーリングポリシーの詳細については、『アプリケーション Auto Scaling ユーザーガイド』の「Application Auto Scaling のターゲット追跡Application Auto Scaling ポリシー」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 がユーザーに代わって 2 組の Amazon CloudWatch アラームを作成します。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 これらのアラームは、テーブルの実際の使用率が一定期間にわたって目標使用率から逸脱したときにトリガーされます。Amazon について詳しくは CloudWatch、Amazon CloudWatch ユーザーガイドをご覧ください。

CloudWatch アラームの 1 つがトリガーされると、Amazon Simple Notification Service (Amazon SNS) から通知が送信されます (有効になっている場合)。次に、 CloudWatch アラームは Application Auto Scaling を呼び出して、スケーリングポリシーを評価します。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 auto Scaling は、実際のワークロードが数分間持続的に上昇 (または低下) している場合にのみ、プロビジョニングされたスループット設定を変更します。 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でのauto スケーリングの仕組み

プロビジョニングされたキャパシティーモードの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常に十分な読み取りおよび書き込みキャパシティーを確保するために、Amazon Keyspaces auto Scaling を設定することをお勧めします。 AWS リージョン

マルチリージョンテーブルを auto Scaling のプロビジョンドモードで使用する場合、1 つのテーブルレプリカのauto 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auto スケーリング設定はリージョンごとに調整できます。たとえば、テーブルがレプリケートされるリージョンごとに異なる読み込み容量と読み込みauto スケーリング設定を指定できます。

指定したリージョンのテーブルレプリカに設定した読み取りauto スケーリング設定は、テーブルの一般的なauto スケーリング設定を上書きします。ただし、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容量が確保されるように、書き込み容量はすべてのテーブルレプリカ間で同期されたままである必要があります。

Amazon Keyspaces auto Scaling は、 AWS リージョン そのリージョンでの使用状況に基づいて、それぞれのテーブルのプロビジョニングされた容量を個別に更新します。その結果、auto Scaling がアクティブな場合、マルチリージョンテーブルの各リージョンでプロビジョニングされる容量が異なる場合があります。

Amazon Keyspaces コンソール、API、または CQL を使用して、マルチリージョンテーブルとそのレプリカのauto スケーリング設定を構成できます。 AWS CLIマルチリージョンテーブルの auto Scaling 設定を作成および更新する方法の詳細については、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の使用方法”。

> **Note**
>
> マルチリージョンテーブルにauto スケーリングを使用する場合は、必ず Amazon Keyspaces API オペレーションを使用してauto スケーリング設定を行う必要があります。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を指定することはできません。 AWS リージョン これにより、サポートされていない構成になることがあります。

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。このような手動調整は、 CloudWatch スケーリングポリシーに添付されている既存のアラームには影響しません。
- 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 “コンソールを使用する場合”」を参照してください。
- AWS CloudFormation スケーリングポリシーの作成に使用する場合は、 AWS CloudFormation スタックがスタックテンプレートと同期するように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。
- Amazon Keyspaces CloudTrail の自動スケーリングをモニタリングする場合、設定検証プロセスの一環としてApplication Auto Scalingが行った呼び出しに対して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。

コンソールによる Amazon Keyspaces オートスケーリングポリシーの管理

コンソールを使用して、新規および既存のテーブルに対して Amazon Keyspaces のオートスケーリングを有効にすることができます。コンソールを使用して、オートスケーリング設定の変更や、オートスケーリングの無効化を行うこともできます。

> **Note**
>
> 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、より高度な機能については、CQL または AWS Command Line Interface （ AWS CLI) を使用して Amazon Keyspaces のスケーリングポリシーをプログラムで管理します。詳細については、Cassandra クエリ言語 (CQL) による Amazon Keyspaces Auto Scaling の管理またはCLI を使用した Amazon Keyspaces スケーリングポリシーの管理を参照してください。

トピック

開始する前に: Amazon Keyspaces オートスケーリングのアクセス許可をユーザーに付与する

開始にあたって、オートスケーリング設定を作成して管理するための適切なアクセス許可がユーザー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**
>
> テーブルのオート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前に、テーブルの Auto Scaling をオフにする必要があります。

Amazon Keyspaces コンソールへのアクセスと Amazon Keyspaces オートスケーリングに関して IAM ユーザーの設定を行うには、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 [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。
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. ポリシーのリストから を選択しAmazonKeyspacesFullAccess、次へ: レビュー を選択します。
6. [Add permissions (許可の追加)] を選択します。

Amazon Keyspaces オートスケーリングが有効になっている新規テーブルの作成

> **Note**
>
> Amazon Keyspaces のオートスケーリングでは、ユーザーに代わってオート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 “サービスリンクロールの使用”」を参照してください。

オートスケーリングが有効になっている新規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 [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、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、オプションの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 (オートスケーリング)] 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. 「容量の書き込み」セクションで、読み込み容量について前のステップで定義したのと同じ設定を選択するか、容量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。指定したオート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。

既存のテーブルでの Amazon Keyspaces のオートスケーリングの有効化

> **Note**
>
> Amazon Keyspaces のオートスケーリングでは、ユーザーに代わってオート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 “サービスリンクロールの使用”」を参照してください。

既存のテーブルに対して Amazon Keyspaces のオートスケーリングを有効にするには

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. 使用するテーブルを選択し、キャパシティ タブに移動します。
3. 「容量設定」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 (オートスケーリング)] を選択し、Amazon Keyspaces オートスケーリングが有効になっている新規テーブルの作成 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. オートスケーリング設定が定義されたら、[Save (保存)] を選択します。

Amazon Keyspaces オートスケーリング設定の変更または無効化

を使用して AWS Management Console 、Amazon Keyspaces のオートスケーリング設定を変更できます。これを行うには、編集するテーブルを選択し、キャパシティタブに移動します。「容量設定」セクションで、「編集」を選択します。読み込み容量または書き込み容量セクションの設定を変更できます。これらの設定の詳細については、「Amazon Keyspaces オートスケーリングが有効になっている新規テーブルの作成」をご参照ください。

Amazon Keyspaces のオートスケーリングを無効にするには、スケールの自動チェックボックスをオフにします。自動スケーリングを無効にすると、Application Auto Scaling でテーブルがスケーラブルターゲットとして登録解除されます。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 が使用するサービスにリンクされたロールを削除するには、すべての でアカウント内のすべてのテーブルのオートスケーリングを無効にする必要がありますAWS リージョン。

コンソールでの Amazon Keyspaces オートスケーリングアクティビティの表示

Amazon Keyspaces オートスケーリングがリソースをどのように使用するかをモニタリングするには、Amazon を使用します。これにより CloudWatch、使用状況とパフォーマンスに関するメトリクスが生成されます。Application Auto Scaling ユーザーガイドの手順に従ってダッシュボードを作成します CloudWatch。

Cassandra クエリ言語 (CQL) による Amazon Keyspaces Auto Scaling の管理

Cassandra クエリ言語 (CQL) を使用して Amazon Keyspaces テーブルの自動スケーリング設定を作成および管理するには、 を使用できますcqlsh。このトピックでは、CQL を使用してプログラムで管理できる Auto Scaling タスクの概要を説明します。

このトピックで説明する CQL ステートメントの詳細については、「」を参照してくださいthe section called “DDL ステートメント”。

トピック

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。

のアクセス許可を設定します。

まだアクセス許可を設定していない場合は、ユーザーがオートスケーリング設定の作成と管理を行えるように、適切なアクセス許可を設定する必要があります。 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。詳細なステップについては、「the section called “開始する前に: Amazon Keyspaces オートスケーリングのアクセス許可をユーザーに付与する”」を参照してください。

cqlsh を設定する

まだインストールして設定していない場合は、 をインストールして設定する必要がありますcqlsh。これを行うには、「」の手順に従いますthe section called “cqlsh-expansionの使用”。その後、 を使用して、以下のセクションのコマンド AWS CloudShell を実行できます。

CQL を使用して自動スケーリングで新しいテーブルを作成する

新しい Amazon Keyspaces テーブルを作成すると、 CREATE TABLEステートメントで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できます。これにより、Amazon Keyspaces はユーザーに代わって Application Auto Scaling に連絡し、テーブルをスケーラブルターゲットとして登録し、プロビジョニングされた書き込みまたは読み取り容量を調整できます。

マルチリージョンテーブルを作成し、テーブルレプリカのさまざまな Auto Scaling 設定を構成する方法の詳細については、「」を参照してくださいthe section called “デフォルト設定でマルチリージョンテーブルを作成する (CQL)”。

> **Note**
>
> Amazon Keyspaces Auto Scaling 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ために、サービスにリンクされたロール (AWSServiceRoleForApplicationAutoScaling_CassandraTable) 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 “サービスリンクロールの使用”」を参照してください。

テーブルの Auto Scaling 設定をプログラムで設定するには、Amazon Keyspaces Auto Scaling のパラメータを含む AUTOSCALING_SETTINGSステートメントを使用します。パラメータは、テーブルのプロビジョニングされたスループットを調整するように Amazon Keyspaces に指示する条件と、実行する追加のオプションアクションを定義します。この例では、mytable の Auto Scaling 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 – Amazon Keyspaces がユーザーに代わってスループットキャパシティを調整できるかどうかを指定します。以下の値が必要です。
  - 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をサポートしています。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 Auto Scaling により、プロビジョンドキャパシティに対する消費キャパシティの比率がこの値またはその近くに留まります。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booleanを指定する 。このパラメータはデフォルトで無効になっています。をオンにするにはscale-in、boolean値を に設定しますFALSE。つまり、ユーザーに代わってテーブルの容量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。ターゲット追跡および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。Auto Scaling
  - 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。ターゲット追跡および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。Auto Scaling

> **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。 consumed-to-provisioned 比率は現在 (150/200)、つまり 75% です。この比率はターゲットを超えているため、Auto Scaling はプロビジョニングされた書き込み容量を 215 に増やし、比率が (150/215)、69.77% になるようにします。これは、target_value可能な限り に近いですが、それを超えないようにします。

mytable の場合、読み込みキャパシティーと書き込みキャパシティーTargetValueの両方を 50% に設定します。Amazon Keyspaces Auto Scaling は、テーブルのプロビジョニングされたスループットを 5～10 キャパシティーユニットの範囲内で調整し、 consumed-to-provisioned 比率が 50% 前後になるようにします。読み込み容量については、 ScaleOutCooldownと の値を 60 ScaleInCooldown 秒に設定します。

次のステートメントを使用して、Auto Scaling 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
            }
        }
    }
};
```

CQL を使用して既存のテーブルで自動スケーリングを有効にする

既存の Amazon Keyspaces テーブルでは、 ALTER TABLEステートメントを使用して、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オートスケーリングを有効にできます。現在オンデマンドキャパシティモードになっているテーブルを更新する場合は、 capacity_modeが必要です。テーブルがすで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は、このフィールドを省略できます。

> **Note**
>
> Amazon Keyspaces のオートスケーリングでは、ユーザーに代わってオート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 “サービスリンクロールの使用”」を参照してください。

次の例では、 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。ステートメントは、自動スケーリングを有効にして、テーブルのキャパシティモードをプロビジョニングモードに変更します。

書き込みキャパシティは、5～10 キャパシティーユニットの範囲内で、ターゲット値が 50% に設定されます。読み取りキャパシティーは、5～10 キャパシティーユニットの範囲内で、ターゲット値が 50% に設定されます。読み込み容量については、 scale_out_cooldownと の値を 60 scale_in_cooldown 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
            }
        }
    }
};
```

CQL を使用してテーブルの Amazon Keyspaces Auto Scaling 設定を表示する

テーブルの Auto Scaling 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}}}
```

CQL を使用してテーブルの Amazon Keyspaces Auto Scaling を無効にする

テーブルの Amazon Keyspaces Auto Scaling はいつでもオフにできます。テーブルの読み込みまたは書き込み容量をスケーリングする必要がなくなった場合は、Amazon Keyspaces がテーブルの読み込みまたは書き込み容量設定を変更し続けないように、自動スケーリングをオフにすることを検討してください。`ALTER TABLE` ステートメントを使用してテーブルを更新できます。

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ーの Auto Scaling をオフにします。また、ユーザーに代わって作成された CloudWatch アラームも削除されます。

```
ALTER TABLE mykeyspace.mytable
WITH AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'autoscaling_disabled': true
    }
};
```

> **Note**
>
> Application Auto Scaling が使用するサービスにリンクされたロールを削除するには、すべての でアカウント内のすべてのテーブルのオートスケーリングを無効にする必要があります AWS リージョン。

CLI を使用した Amazon Keyspaces スケーリングポリシーの管理

Amazon Keyspaces の Auto Scaling 設定をプログラムで更新および管理するには、 AWS Command Line Interface （ AWS CLI) または AWS API を使用できます。Cassandra クエリ言語 (CQL) を使用して Amazon Keyspaces Auto Scaling ポリシーを管理するには、「」を参照してくださいthe section called “CQL の使用”。このトピックでは、 を使用してプログラムで管理できる Auto Scaling タスクの概要を説明します AWS CLI。

このトピックで説明する Amazon Keyspaces AWS CLI コマンドの詳細については、AWS CLI「 コマンドリファレンス」を参照してください。

トピック

- 開始する前に
- を使用して自動スケーリングで新しいテーブルを作成する AWS CLI

- を使用して既存のテーブルでオートスケーリングを有効にする AWS CLI
- を使用してテーブルの Amazon Keyspaces Auto Scaling 設定を表示する AWS CLI
- を使用してテーブルの Amazon Keyspaces Auto Scaling をオフにする AWS CLI

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。

のアクセス許可を設定します。

まだアクセス許可を設定していない場合は、ユーザーがオートスケーリング設定の作成と管理を行えるように、適切なアクセス許可を設定する必要があります。 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。詳細なステップについては、「the section called “開始する前に: Amazon Keyspaces オートスケーリングのアクセス許可をユーザーに付与する”」を参照してください。

AWS CLI のインストール

まだ AWS CLI をインストールして設定していない場合は、インストールして設定する必要があります。これを行うには、「 AWS Command Line Interface ユーザーガイド」に移動し、以下の手順に従います。

- AWS CLI のインストール
- AWS CLI の設定

を使用して自動スケーリングで新しいテーブルを作成する AWS CLI

新しい Amazon Keyspaces テーブルを作成すると、 `CreateTable`オペレーションで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できます。これにより、Amazon Keyspaces はユーザーに代わって Application Auto Scaling に連絡し、スケーラブルターゲットとして指定したテーブルを登録し、プロビジョニングされた書き込みまたは読み取り容量を調整できます。

Auto Scaling 設定でマルチリージョンテーブルを作成する方法の詳細については、「」を参照してくださいthe section called “Auto Scaling によるプロビジョニングモードでの新しいマルチリージョンテーブルの作成 (CLI)”。

> **Note**
>
> Amazon Keyspaces Auto Scaling 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ために、サービスにリンクされたロール (AWSServiceRoleForApplicationAutoScaling_CassandraTable) 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 “サービスリンクロールの使用”」を参照してください。

テーブルの Auto Scaling 設定をプログラムで設定するには、Amazon Keyspaces Auto Scaling のパラメータを定義する autoScalingSpecificationアクションを使用します。パラメータは、テーブルのプロビジョニングされたスループットを調整するように Amazon Keyspaces に指示する条件と、実行する追加のオプションアクションを定義します。この例では、mytable の Auto Scaling 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 – Amazon Keyspaces がユーザーに代わって容量スループットを調整できるかどうかを指定します。読み込みキャパシティーと書き込みキャパシティーの Auto Scaling を個別に有効にできます。次に、 に次のパラメータを指定する必要がありますautoScalingSpecification。
  - writeCapacityAutoScaling - 最大および最小書き込みキャパシティユニット。
  - readCapacityAutoScaling - 最大読み込みキャパシティーユニットと最小読み込みキャパシティーユニット。
  - scalingPolicy — Amazon Keyspaces はターゲット追跡ポリシーをサポートしています。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 Auto Scaling により、プロビジョンドキャパシティに対する消費キャパシティの比率がこの値またはその近くに留まります。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booleanを指定する 。このパラメータはデフォルトで無効になっています。をオンにするにはscale-in、boolean値を に設定しますFALSE。つまり、ユーザーに代わってテーブルの容量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

追跡および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。 Auto Scaling

- 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。ターゲット追跡および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。 Auto Scaling

> **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。 consumed-to-provisioned 比率は現在 (150/200)、つまり 75% です。この比率はターゲットを超えているため、Auto Scaling はプロビジョニングされた書き込み容量を 215 に増やし、比率が (150/215) または 69.77% になるようにします。これは、TargetValue可能な限り に近いが、それを超えないようにします。

mytable の場合、読み込みキャパシティーと書き込みキャパシティーTargetValueの両方を 50% に設定します。Amazon Keyspaces Auto Scaling は、テーブルのプロビジョニングされたスループットを 5～10 キャパシティーユニットの範囲内で調整し、 consumed-to-provisioned 比率が 50% 前後に維持されるようにします。読み込み容量については、 ScaleOutCooldownと の値を 60 ScaleInCooldown 秒に設定します。

複雑な Auto Scaling 設定でテーブルを作成する場合、Auto Scaling 設定を JSON ファイルからロードすると便利です。次の例では、サンプル JSON ファイルを auto-scaling.zip からダウンロードしてを抽出しauto-scaling.json、 ファイルへのパスを書き留めます。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
```

を使用して既存のテーブルでオートスケーリングを有効にする AWS CLI

既存の Amazon Keyspaces テーブルでは、 `UpdateTable`オペレーションを使用して、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オートスケーリングを有効にできます。マルチリージョンテーブルの Auto Scaling 設定を更新する方法の詳細については、「」を参照してくださいthe section called “マルチリージョンテーブルのプロビジョンドキャパシティと Auto Scaling 設定の更新 (CLI)”。

> **Note**
>
> Amazon Keyspaces のオートスケーリングでは、ユーザーに代わってオート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 “サービスリンクロールの使用”」を参照してください。

既存のテーブルに対して Amazon Keyspaces Auto Scaling を有効にするには、次のコマンドを使用します。テーブルの Auto Scaling 設定は、JSON ファイルからロードされます。次の例では、サンプル JSON ファイルを auto-scaling.zip からダウンロードして を抽出し`auto-scaling.json`、ファイルへのパスを書き留めます。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 Auto Scaling 設定の詳細については、「」を参照してくださいthe section called “を使用して自動スケーリングで新しいテーブルを作成する AWS CLI”。

```
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
```

を使用してテーブルの Amazon Keyspaces Auto Scaling 設定を表示する AWS CLI

テーブルの Auto Scaling 設定を表示するには、 `get-table-auto-scaling-settings`オペレーションを使用できます。次の CLI コマンドはその一例です。

```
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
                }
            }
        }
    }
}
```

を使用してテーブルの Amazon Keyspaces Auto Scaling をオフにする AWS CLI

テーブルの Amazon Keyspaces Auto Scaling はいつでも無効にできます。テーブルの読み込みまたは書き込み容量をスケーリングする必要がなくなった場合は、Amazon Keyspaces がテーブルの読み込みまたは書き込み容量設定を変更し続けないように、自動スケーリングをオフにすることを検討してください。テーブルは `UpdateTable`オペレーションで更新できます。

次のコマンドは、テーブルの読み込みキャパシティーの Auto Scaling をオフにします。また、ユーザーに代わって作成された 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}
```

> **Note**
>
> Application Auto Scaling が使用するサービスにリンクされたロールを削除するには、すべての でアカウント内のすべてのテーブルのオートスケーリングを無効にする必要がありますAWS リージョン。

Amazon キKeyspaces でバースト容量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 容量消費量を推定する方法

Amazon Keyspaces でデータを読み書きする際、クエリが消費する読み取り/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ーユニット (RCU/WCU) の量は、クエリを実行するために Amazon Keyspaces が処理しなければならないデータの合計量によって異なります。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らなければならなかったデータのサブセットである可能性があります。条件付き書き込みの場合、条件付きチェックが失敗しても Amazon Keyspaces は書き込み容量を消費します。

1 つのリクエストで処理されるデータの総量を見積もるには、エンコードされた行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、Amazon Keyspaces がクエリを処理する方法と、それが容量消費にどのように影響するかを示すために、一般的なシナリオとアクセスパターンの例をいくつか紹介します。例に従ってテーブルの容量要件を見積もり、Amazon CloudWatch を使用してこれらのユースケースの読み取りおよび書き込みキャパシティー消費量を観察できます。

Amazon Keyspaces でエンコードされた行のサイズを計算する方法については、を参照してくださいthe section called “行サイズの計算”。

トピック

レンジクエリ

レンジクエリの読み込みキャパシティー消費量を調べるには、オンデマンドキャパシティモードを使用した以下のサンプルテーブルを使用します。

```
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>
```

次に、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を受け取り、Amazon Keyspaces によって実行される読み取りオペレーションは、コンシステンシーモードで LOCAL_QUORUM 2 つの RRU を消費します。

```
pk1 | pk2 | pk3 | ck1 | ck2 | ck3 | value
-----+-----+-----+-----+-----+-----+-------
a | b | 1 | a | b | 60 | value_1
```

Amazon Keyspaces esは、ck3=60ck3=70値を含む行を評価してクエリを処理するために、2つのRRUを消費します。ただし、Amazon Keyspaces は、WHEREクエリで指定された条件が真である行、ck3=60つまり値のある行のみを返します。クエリで指定された範囲を評価するために、Amazon Keyspaces は範囲の上限と一致する行を読み取ります。この場合ck3 = 70、結果にはその行は返されません。読み込み容量の消費量は、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いています。

クエリを制限する

LIMITこの句を使用するクエリを処理する場合、Amazon Keyspaces は、クエリで指定された条件に一致させようとすると、最大ページサイズまで行を読み取ります。Amazon Keyspaces LIMIT が最初のページの値と一致する十分なデータを見つけられない場合は、ページ分割された呼び出しが 1 回以上必要になる可能性があります。次のページを読み続けるには、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句を使用するときに消費する読み込み容量を減らすには、ページサイズを小さくすることができます。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;”
```

ページサイズを設定しない場合、Amazon Keyspaces は 1 行しか返しませんが、1 MB のデータを読み取ります。Amazon Keyspaces が 1 行だけを読み込むようにするには、このクエリのページサイズを 1 に設定します。この場合、T ime-to-live 設定またはクライアント側のタイムスタンプに基づいて期限切れの行がない限り、Amazon Keyspaces は 1 行だけを読み取ります。読み込み容量の消費を抑えるには、ページサイズを Amazon Keyspaces LIMIT が読み取るデータ量を減らす値と同じに設定することをお勧めします。

テーブルスキャン

ALLOW FILTERINGオプションを使用するクエリなど、テーブル全体のスキャンにつながるクエリは、結果として返されるよりも多くの読み取りを処理するクエリの 1 つです。また、読み込み容量の消費量は、返されたデータではなく、読み取られたデータに基づいています。

テーブルスキャンの例では、以下のテーブル例をオンデマンドキャパシティモードで使用しています。

```
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 は、デフォルトで 4 つのパーティションを使用してオンデマンドキャパシティモードでテーブルを作成します。このテーブル例では、すべてのデータが 1 つのパーティションに格納され、残りの 3 つのパーティションは空です。

次に、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、Amazon Keyspaces はテーブルの 4 つのパーティションすべてをスキャンし、コンシステンシーモードで 6 つの RRU を消費するテーブルスキャンオペレーションになります。LOCAL_QUORUMまず、Amazon Keyspaces は 3 つの行の読み取りに 3 つの RRU を消費します。pk='pk'次に、Amazon Keyspaces は追加の 3 つの RRU を消費して、テーブルの 3 つの空のパーティションをスキャンします。このクエリの結果はテーブルスキャンになるため、Amazon Keyspaces はデータのないパーティションを含め、テーブル内のすべてのパーティションをスキャンします。

軽量トランザクション

ライトウェイトトランザクション (LWT) では、テーブルデータに対して条件付き書き込み操作を実行できます。条件付き更新操作は、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 キKeyspaces では、すべての書き込みオペレーションに LOCAL_QUORUM の整合性が必要であり、LWT を使用しても追加料金は発生しません。LWT の違いは、LWT の条件チェック

の結果が FALSE になると、書き込みキャパシティーユニットを消費する点です。消費される書き込みキャパシティーユニットの数は、行のサイズによって異なります。行サイズが 2 KB の場合、条件付き書き込みが失敗すると 2 つの書き込みキャパシティーユニットが消費されます。その行が現在テーブルに存在しない場合、操作は 1 つの書き込みキャパシティーユニットを消費します。`ConditionalCheckFailed`内のメトリクスを監視することで CloudWatch 、LWT 条件チェックの失敗によって消費される容量を判断できます。

Amazon による読み取り/書き込み容量の消費量の見積もり CloudWatch

読み取り/書き込みキャパシティーの消費量を推定して監視するには、 CloudWatch ダッシュボードを使用できます。Amazon Keyspaces で利用できる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 “メトリクスとディメンション”。

特定のステートメントによって消費される読み取りおよび書き込みキャパシティーユニットをでモニタリングするには CloudWatch、以下の手順に従います。

1. サンプルデータを含む新しいテーブルを作成します。
2. テーブルの Amazon Keyspaces CloudWatch ダッシュボードを設定します。はじめに、Github にあるダッシュボードテンプレートを使用できます。
3. `ALLOW FILTERING`たとえばオプションを使用して CQL ステートメントを実行し、ダッシュボードでテーブル全体のスキャンで消費された読み込みキャパシティーユニットをチェックします。

Amazon Keyspaces (Apache Cassandra 向け) でのキースペース、テーブル、行の操作

この章では、Amazon Keyspaces (Apache Cassandra 向け) でのキースペース、テーブル、行などの詳しい操作について説明します。Amazon でキースペースとテーブルをモニタリングする方法については CloudWatch、「」を参照してくださいthe section called “によるモニタリング CloudWatch”。

トピック

- Amazon Keyspaces でのキー空間の使用
- Amazon Keyspaces でのテーブルの操作
- Amazon Keyspaces での行の操作
- Amazon Keyspaces でのクエリの使用
- Amazon Keyspaces でのパーティショナーの操作
- Amazon Keyspaces リソースのタグとラベルの使用

Amazon Keyspaces でのキー空間の使用

このセクションでは、Amazon Keyspaces (Apache Cassandra 向け) におけるキー空間の詳しい使用方法について説明します。

トピック

- Amazon Keyspaces でのシステムキースペースの操作
- Amazon Keyspaces でのキースペースの作成

Amazon Keyspaces でのシステムキースペースの操作

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`

以下のセクションでは、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する場合、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する場合は、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

テーブル名	列名	コメント
		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。
tables	keyspace_name, table_name, bloom_filter_fp_chance, caching, comment,	特定のトリガーに関する情報です。

テーブル名	列名	コメント
	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	
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格納する 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

テーブル名	列名	コメント
		section called “キースペースの作成”」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, 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”」を参照してください。 このテーブルをクエリして、Amazon Keyspaces に固有で、 として保存されている設定を一覧表示することもできますcustom_properties 。例: • capacity_mode • client_side_timestamps • encryption_specification • point_in_time_recovery • ttl
tables_history	keyspace_name, table_name, event_time, creation_time, custom_properties, event	このテーブルをクエリすると、特定のテーブルのスキーマ変更について調べることができます。

テーブル名	列名	コメント
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 “CQL を使用した新規キースペースおよびテーブルへのタグの追加”」を参照してください。
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、プロビジョニングされたテーブルの Auto Scaling 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使用できないことに注意してください。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 “CQL の使用”」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ている 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、 として保存されている Amazon Keyspaces に固有の設定を一覧表示することもできますcustom_properties 。例: • 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空間の作成 (CQL)”」を参照してください。
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、マルチリージョンプロビジョニングテーブルの Auto Scaling 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使用できないことに注意してください。このテーブルをクエリするには、WHERE 句に keyspace_name および

テーブル名	列名	コメント
		table_name を指定する必要があります。詳細については、「the section called “CQL の使用”」を参照してください。

Amazon Keyspaces でのキースペースの作成

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、キースペースがいつ保留中またはアクティブであるかを示します。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、使用可能な状態になるとsystem_schema_mcskeyspaces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';
```

正常に作成されたキースペースの場合、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 でのテーブルの操作

このセクションでは、Amazon Keyspaces (Apache Cassandra 向け) におけるテーブルの詳しい操作方法について説明します。

トピック

Amazon Keyspaces でのテーブルの作成

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なタイミングを示します。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 でのマルチリージョンテーブルの操作

マルチリージョンテーブルには、次の 2 つの方法のいずれかで書き込みスループットキャパシティが設定されている必要があります。

- 書き込みリクエストユニット (WRUsで測定されるオンデマンドキャパシティモード
- Auto Scaling による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 (WCUsで測定

プロビジョンドキャパシティモードを Auto Scaling 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 への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に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と予想される読み取り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 は、以前に到達したトラフィックレベルまで拡張または縮小されるため、ワークロードに即座に対応できます。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すると、Amazon Keyspaces はワークロードに対応するために迅速に適応します。

テーブルにプロビジョンドキャパシティモードを選択する場合は、アプリケーションが必要とする 1 秒あたりの読み込みキャパシティユニット (RCUs) と書き込みキャパシティユニット (WCUs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、まず各リージョンに必要な 1 秒あたりの WCUs数を見積もる必要があります。次に、テーブルがレプリケートされているすべてのリージョンからの書き込みを追加し、その合計を使用して各リージョンの容量をプロビ

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、必須です。

テーブルにすべてのリージョンからの書き込みを処理するのに十分な容量がない場合、容量例外が発生します。さらに、リージョン間の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 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、テーブルは各リージョンで 20 WCUs を消費することを想定する必要があります。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) 。つまり、この例では、テーブルのレプリカごとに 20 WCUs をプロビジョニングする必要があります。Amazon を使用して、テーブルの容量消費をモニタリングできます CloudWatch。詳細については、「the section called “によるモニタリング CloudWatch”」を参照してください。

各マルチリージョン書き込みは 1.25 倍の WCUsとして請求されるため、この例では合計 75 WCUs 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 Auto Scaling によるプロビジョンドキャパシティの詳細については、「」を参照してくださいthe section called “auto スケーリングによるスループット容量の管理”。

> **Note**
>
> テーブルが Auto Scaling でプロビジョニングされたキャパシティモードで実行されている場合、プロビジョニングされた書き込みキャパシティは、各リージョンのそれらの Auto Scaling 設定内でフロートできます。

Amazon Keyspaces の静的列

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。

Amazon Keyspaces の各論理パーティションの静的列サイズの計算

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 “行サイズの計算”」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 “行サイズの計算” 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 “行サイズの計算” 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 での静的データの読み取り/書き込みオペレーションの計測

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 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 CloudWatch、「」を参照してくださいthe section called “によるモニタリング CloudWatch”。

Amazon Keyspaces での行の操作

このセクションでは、Amazon Keyspaces (Apache Cassandra 向け) における行の詳しい操作方法について説明します。テーブルは Amazon Keyspaces の主要なデータ構造であり、テーブル内のデータは列と行で構成されています。

トピック

- Amazon Keyspaces での行サイズの計算

Amazon Keyspaces での行サイズの計算

Amazon Keyspaces には、1 桁台のミリ秒の読み取り/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渡って永続的にデータを保存するフルマネージドストレージがあります。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。

このセクションでは、Amazon Keyspaces でエンコードされた行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。また、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も、エンコードされた

行サイズを使用すべきです。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。

- プライマリキー、クラスタリング列、または STATIC 列ではない通常の列の場合は、データ型に基づいた生のセルデータサイズを使用し、必要なメタデータを追加します。Amazon Keyspaces デベロッパーガイドでサポートされているのデータ型の詳細については、「the section called “データ型”」参照してください。Amazon Keyspaces がデータ型値とメタデータを保存する方法の主な違いを以下に示します。
- 各列名に必要な容量は列識別子を使用して保存され、列に格納されている各データ値に加算されます。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。
- 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。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。
- 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。余分なオーバーヘッドは、効率的なクエリと組み込みのインデックス作成に使用されます。
- 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。
- Cassandra INT、BIGINT、SMALLINT、および TINYINT データ型は、有効桁数が最大 38 桁の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。
- Amazon Keyspaces の BLOB は、値の生のバイト長で保存されます。
- Null 値または Boolean 値のサイズは 1 バイトです。

- その内容にかかわらず、LIST または MAP 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。
- STATIC 列データは、最大行サイズである 1 MB に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。
- この機能を有効にすると、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は約 20 ~ 40 バイト (データによって異なります) を占め、行のストレージとスループットのコストに影響します。詳細については、「the section called “Amazon Keyspaces でのクライアント側のタイムスタンプ”」を参照してください。
- 行メタデータのために各行のサイズに 100 バイトを追加します。

エンコードされたデータ行の合計サイズは、次の式に基づいています。

```
partition key columns + clustering columns + regular columns + row metadata = total
 encoded size of row
```

> **Important**
>
> 列 ID、パーティションキーメタデータ、クラスタリング列のメタデータなどの列のメタデータ、クライアント側のタイムスタンプ、行メタデータなど、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。

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サイズの合計を出すには、すべての列のバイトを合計し、行のメタデータに追加で 100 バイトを足します。

```
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 “によるモニタリング CloudWatch”」を参照してください。

Amazon Keyspaces でのクエリの使用

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 での SELECT ステートメントでの IN 演算子の使用
- Amazon Keyspaces での結果の順序付け
- Amazon Keyspaces での結果のページ分割

Amazon Keyspaces での **SELECT** ステートメントでの **IN** 演算子の使用

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 での結果の順序付け

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 での結果のページ分割

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。PAGE SIZEof LIMIT クエリが読み込み容量に与える影響の例については、を参照してくださいthe section called “クエリを制限する”。

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。

AWS Management Console または Cassandra クエリ言語 (CQL) を使用してパーティショナーを変更できます。

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. AWS Management Console にサインインし、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 リソースのタグとラベルの使用

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 請求情報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. Amazon Keyspaces のタグ付け制限 について理解します。
2. Amazon Keyspaces のタグ付けオペレーション を使用してタグを作成します。
3. を使用してAmazon Keyspaces のコスト配分レポート、アクティブなタグごとの 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。

Amazon Keyspaces のタグ付け制限

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。

- Amazon Keyspaces の各キースペースまたはテーブルには、同一のキーを含むタグを 1 つだけ付けることができます。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は適用されません。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。
- 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 が割り当てたタグの名前と値には自動的に aws: プレフィックスが付けられますが、これをユーザーが割り当てることはできません。AWS が割り当てたタグの名前は、ユーザーが定義したリソースタグの制限 50 個に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 のタグ付けオペレーション

Amazon Keyspaces (Apache Cassandra 向け) コンソール、AWS CLI、または Cassandra クエリ言語 (CQL) を使用して、キースペースとテーブルのタグの追加、一覧表示、編集、削除を実行することができます。次に、これらのユーザー定義タグをアクティブ化し、コスト配分の追跡のため、AWS Billing and Cost Management コンソールに表示できます。詳細については、「Amazon Keyspaces のコスト配分レポート」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。

トピック

- コンソールを使用した新規キースペースおよびテーブルへのタグの追加
- AWS CLI を使用した新規キー空間およびテーブルへのタグの追加
- CQL を使用した新規キースペースおよびテーブルへのタグの追加

コンソールを使用した新規キースペースおよびテーブルへのタグの追加

Amazon Keyspaces コンソールを使用して、新しいキースペースおよびテーブルの作成時にタグを追加することができます。既存のテーブルのタグの追加、一覧表示、編集、削除も実行できます。

キー空間の作成時にキー空間にタグを付けるには (コンソール)

1. AWS Management Console にサインインし、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。タグのキーと値を入力して、[Add new tag (新しいタグの追加)]を選択します。
4. [Create keyspace] (キースペースの作成) を選択します。

テーブルの作成時にテーブルにタグを付けるには (コンソール)

1. AWS Management Console にサインインし、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。

既存のリソースにタグを付けるには (コンソール)

1. AWS Management Console にサインインし、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)] または [Tables (テーブル)] を選択します。
3. リスト内のキー空間またはテーブルを選択します。次に [Manage tags (タグの管理)] を選択して、タグの追加、編集または削除を行います。

タグ構造の詳細については、「Amazon Keyspaces のタグ付け制限」を参照してください。

AWS CLI を使用した新規キー空間およびテーブルへのタグの追加

このセクションの例では、キースペースとテーブルの作成時に AWS CLI を使ってタグを指定する方法、既存のリソースにタグを追加または削除する方法、およびタグを一覧表示する方法を示しています。

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'
```

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'
```

次の例では、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/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"
        }
    ]
}
```

CQL を使用した新規キースペースおよびテーブルへのタグの追加

次の例では、キースペースとテーブルの作成時に CQL を使ってタグを指定する方法、既存のリソースにタグを付ける方法、およびタグを読み取る方法を示しています。

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'} ;
```

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'};
```

他のコマンドを使用してステートメント内のリソースにタグを付けるには

```
CREATE KEYSPACE mykeyspace WITH REPLICATION = {'class': 'Simple Strategy'} AND TAGS
 = {'key1':'val1', 'key2':'val2'};
```

次の例では、既存のキースペースとテーブルでタグの追加または削除を行う方法を示します。

```
ALTER KEYSPACE mykeyspace ADD TAGS {'key1':'val1', 'key2':'val2'};
```

```
ALTER TABLE mytable DROP TAGS {'key1':'val1', 'key2':'val2'};
```

リソースにアタッチ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 句は必須であり、次のいずれかの形式でなければなりません。

- keyspace_name = **'mykeyspace'** AND resource_type = 'keyspace'
- keyspace_name = **'mykeyspace'** AND resource_name = **'mytable'**
- resource_id = *arn*

例:

次のクエリは、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'}
```

次のクエリはテーブルのタグを示しています。

```
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'}
```

Amazon Keyspaces のコスト配分レポート

AWS はタグを使用して、コスト配分レポートでリソースコストを分類します。AWS には 2 つのタイプのコスト配分タグがあります。

- AWS で生成されたタグ。AWS はユーザーのためにこのタグを定義、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 で生成されたタグをアクティブ化するには:

1. AWS Management Console にサインインし、https://console.aws.amazon.com/billing/home#/ で請求およびコスト管理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. AWS Management Console にサインインし、https://console.aws.amazon.com/billing/home#/ で請求およびコスト管理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、AWS は、アクティブなタグごとにグループ化された使用量とコストを含むコスト配分レポートを生成します。コスト配分レポートには、ご利用の AWS のサービスのコストすべてが請求期間別に出力されます。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。

Amazon Keyspaces 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

Amazon Keyspaces 用の NoSQL 設計

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

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。

- 関連するデータをまとめてください。 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。

Amazon Keyspaces (Apache Cassandra 用) とのクライアントドライバー接続

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

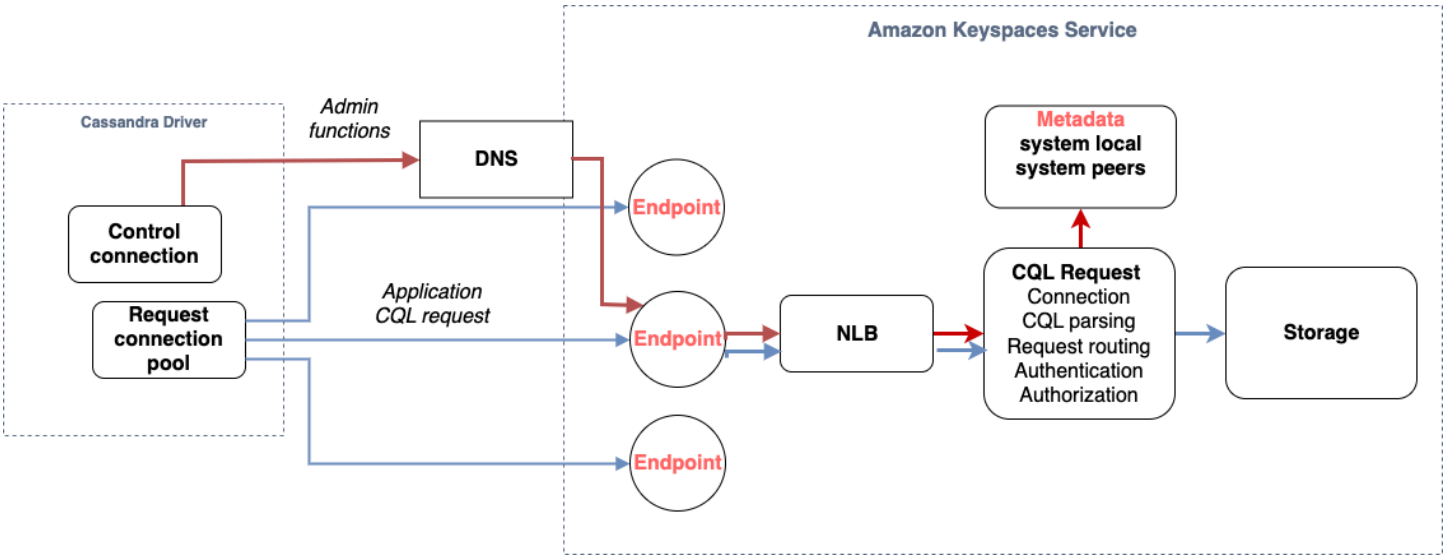
- Amazon Keyspaces における接続の働き
- Amazon Keyspaces で接続を設定する方法
- Amazon Keyspaces の VPC エンドポイント経由接続の設定方法
- Amazon Keyspaces で接続をモニタリングする方法。
- Amazon Keyspaces での接続エラーの処理方法

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 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 エンドポイント経由で接続する場合、ほとんどの場合、3 つのエンドポイントを使用できます。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基づいて、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 で使用できる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 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、Amazon Keyspaces エンドポイントは 1 つだけ表示される可能性があります。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”」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。

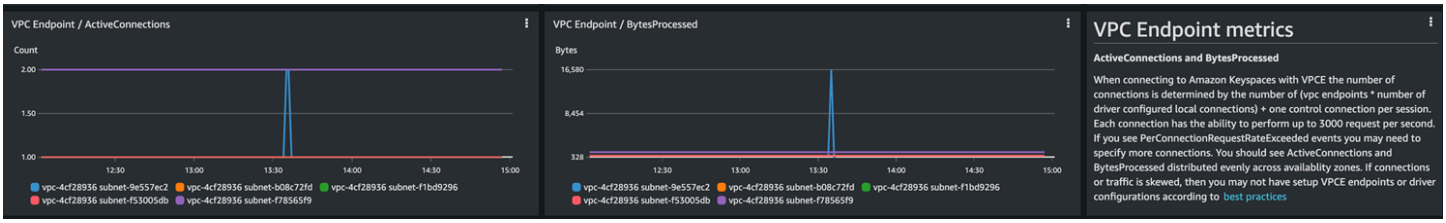
アプリケーションが接続されているエンドポイントの数がすぐにわかるように、検出されたピアの数を `system.peers` テーブルに記録できます。次の例は、接続が確立されるとピア数を出力する Java コードの例です。

```
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ことで、VPC エンドポイントを使用するときにピアの数を確認することもできます。では CloudWatch、VPC エンドポイントに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。以下の図は、ドライバー設定で設定された 1 つの接続で Amazon Keyspaces に接続した後の VPC エンドポイント 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 で GitHub 利用できる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。

での Java のエクスポネンシャル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 GitHub 。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。

Amazon Keyspaces (Apache Cassandra 向け) でのデータモデリング

このトピックでは、Amazon Keyspaces (Apache Cassandra 向け) のデータモデリングの概念を紹介します。このセクションを利用して、アプリケーションのデータアクセスパターンに適合するデータモデルの設計に関する推奨事項について説明します。データモデリングのベストプラクティスを実施すると、Amazon Keyspaces を使用する際のパフォーマンスが向上し、スループットコストが最小限に抑えられます。

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。

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” 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た Amazon Keyspaces でのワークロードの均等分散

書き込みシャーディングを使用した Amazon Keyspaces でのワークロードの均等分散

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

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 の一般的な課題の 1 つは、特定のテーブルのコストだけでは簡単にコストを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でフィルタリングまたはグループ化できないためです。Amazon Keyspaces コンソールのテーブルの Monitor タブで、各テーブルの請求対象テーブルサイズ (バイト) のメトリクスを表示できます。 テーブルごとにさらにコスト関連情報が必要な場合は、このセクションでは、タグ付けを使用して Cost Explorer で個々のテーブルのコスト分析を実行する方法を示します。

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

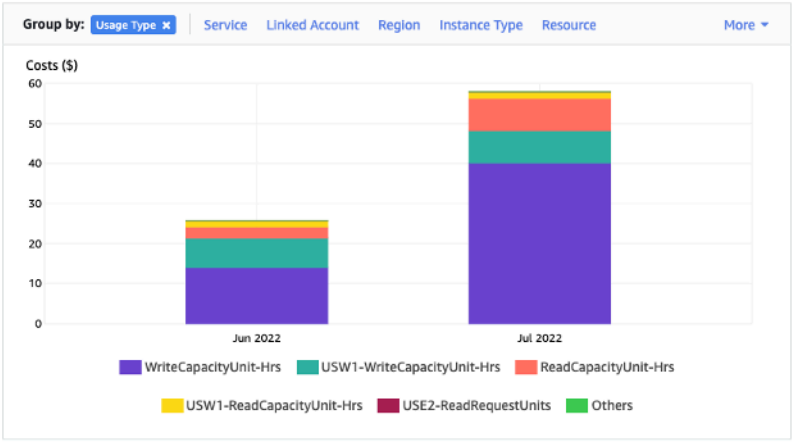
1 つの Amazon Keyspaces テーブルのコストを表示する方法

Amazon Keyspaces テーブルに関する基本情報は、プライマリキースキーマ、請求対象テーブルサイズ、容量関連のメトリクスなど、コンソールで確認できます。テーブルのサイズを使用して、テーブルの月次ストレージコストを計算できます。例えば、米国東部 (バージニア北部) では、1 GB あたり 0.25 USD です AWS 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) の設定も返ります。この情報を使用して、テーブルの現在の読み取りおよび書き込みコストを計算できます。これらのコストは、特に Amazon Keyspaces 自動スケーリングでテーブルを設定している場合に変わる可能性があることに注意してください。

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 “タグの使用”」を参照してください。

この例では、 という名前のテーブルを使用しますMyTable。

1. table_name のキーと の値を持つタグを設定しますMyTable。
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 “Amazon Keyspaces のコスト配分レポート”」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

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”」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。

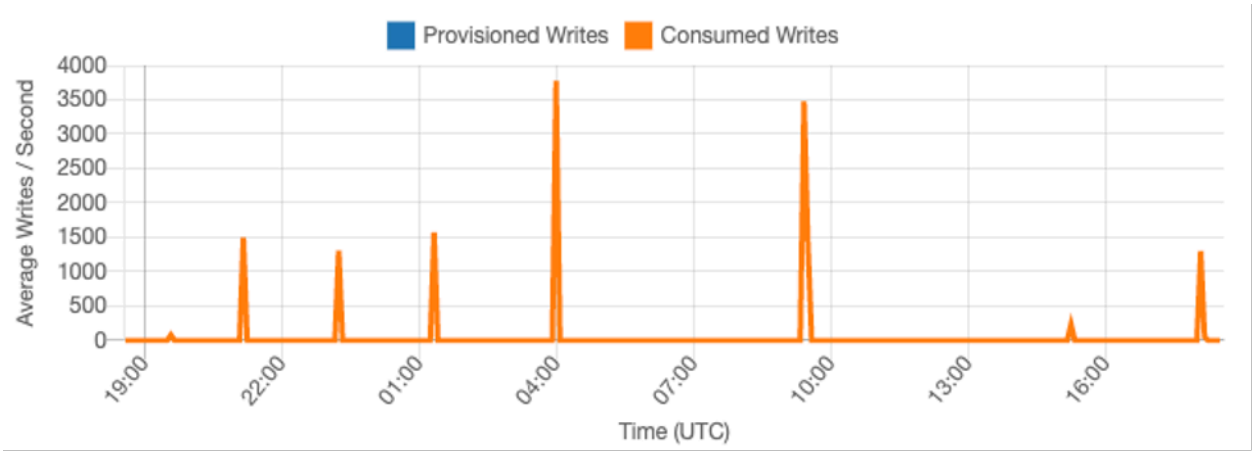
プロビジョンドキャパシティモードでは、テーブルをオーバープロビジョニングとアンダープロビジョニングの双方の間で調整して、スループットキャパシティエラーの発生率の軽減と、コスト最適化の両方を実現する必要があります。

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。

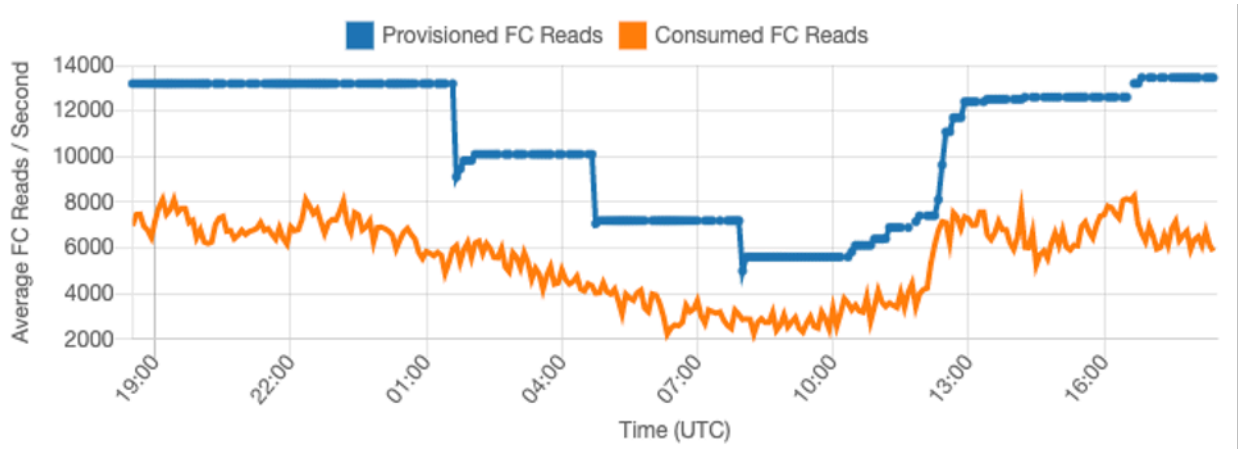
オンデマンドテーブルはリクエストに応じて請求されるため、コスト最適化のためにテーブルレベルでこれ以上の対応は不要です。オンデマンドテーブルを定期的に評価して、ワークロードに上記の現象が引き続き残っていることを確認してください。ワークロードが安定したら、コスト最適化のためにプロビジョンドモードへの変更を検討してください。

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の概要を説明します。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。

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」のサービス CloudWatch 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。

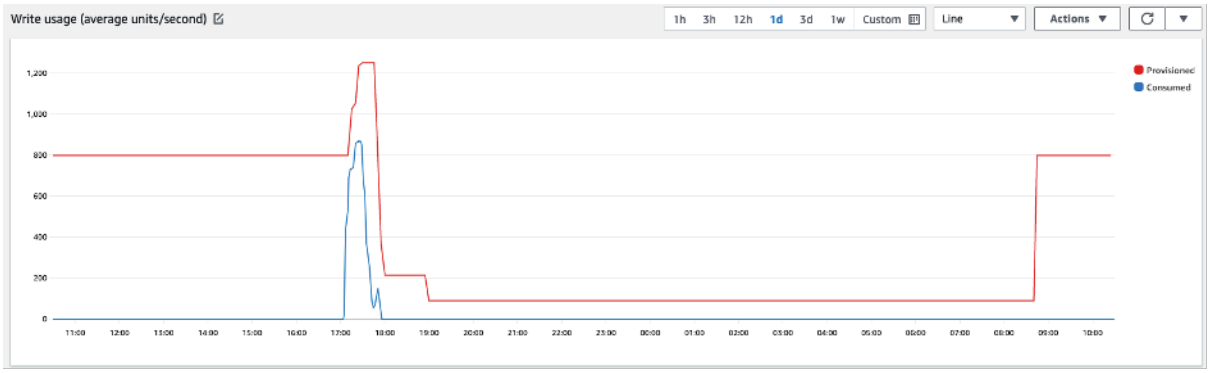
```
aws application-autoscaling describe-scheduled-actions --service-namespace
 cassandra
```

   次のような結果が表示されます。

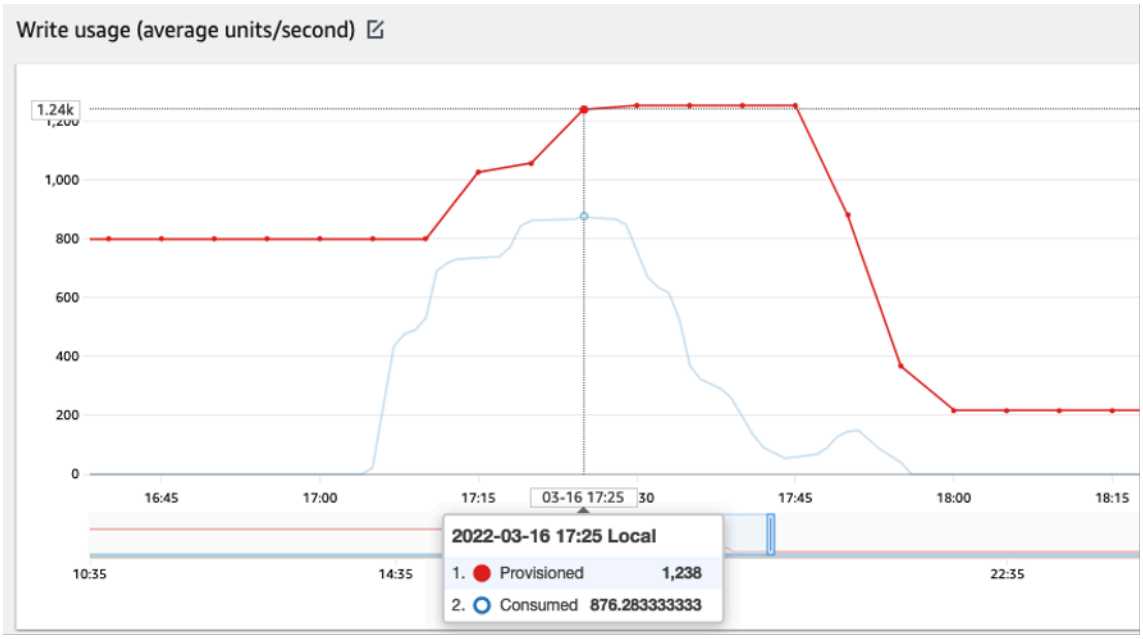
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/keyspaces/
table/table-name:scheduledActionName/my-5-8-scheduled-down-action",
```

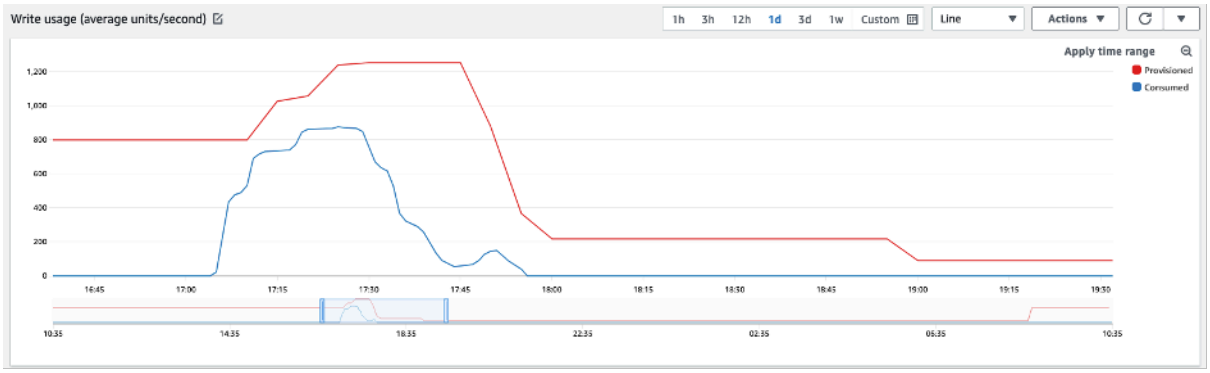
```
            "ServiceNamespace": "cassandra",
            "Schedule": "cron(15 17 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "keyspace/keyspace-name/table/table-name",
            "ScalableDimension": "cassandra:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "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 Auto Scaling アクションは最大値と最小値に影響します。設定する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。特定のアプリケーションのニーズに対応するために、 Amazon EventBridge や Step Functions などの AWS サービスを組み合わせて活用できます。

未使用のリソースを特定する

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。以下で説明する手順では、Amazon CloudWatch メトリクスを使用して未使用のリソースを特定し、コストを削減するためのアクションを実行します。

を使用して Amazon Keyspaces をモニタリングできます。これにより CloudWatch、Amazon Keyspaces から raw データを収集し、ほぼリアルタイムの読み取り可能なメトリクスに加工できます。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 ユーザーガイド」の「Amazon CloudWatchとは」および「メトリクスの保持 CloudWatch 」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 point-in-time 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、30 日間にわたって次の CloudWatch メトリクスを調べて、特定のテーブルにアクティブな読み取りまたは書き込みがあるかどうかを確認できます。

`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、未使用のリソースを識別するために使用できる Amazon Keyspaces テーブルメトリクスを提供するモニタリングおよびオブザーバビリティサービスです。 CloudWatch メトリクスは、 AWS Management Console および を通じて表示できます AWS Command Line Interface。

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/ CloudWatch の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。

コールドストレージ階層化を活用 AWS Backup することで、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 point-in-time リカバリ (PITR) バックアップのクリーンアップ

Amazon Keyspaces は P oint-in-time リカバリを提供します。P リカバリは 35 日間連続バックアップを提供し、偶発的な書き込みや削除から保護します。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、ポイントインタイムリカバリ 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

テーブルの容量を評価するには、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、 CloudWatch まず API を使用していくつかの履歴データポイントをキャプチャ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	SampleTable
<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」のサービス CloudWatch 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、グラフの統計関数 Average CloudWatch が選択されます。
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、名前を AWS CLI セクションで使用した名前と一致させる必要があります。[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”」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。

使用率が低いワークロードと高いスロットリング率 (間隔のMax(ThrottleEvents)/Min(ThrottleEvents)) がある場合、これは、特定の日 (または時間帯) にトラフィックが大幅に増加するが、それ以外は一貫して低いという非常に急激なワークロードがある場合に発生する可能性があります。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) 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。

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 “データモデラー”」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 “データビジュアライザー”」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
- データモデルを構築する方法
- データモデルを可視化する方法
- 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。

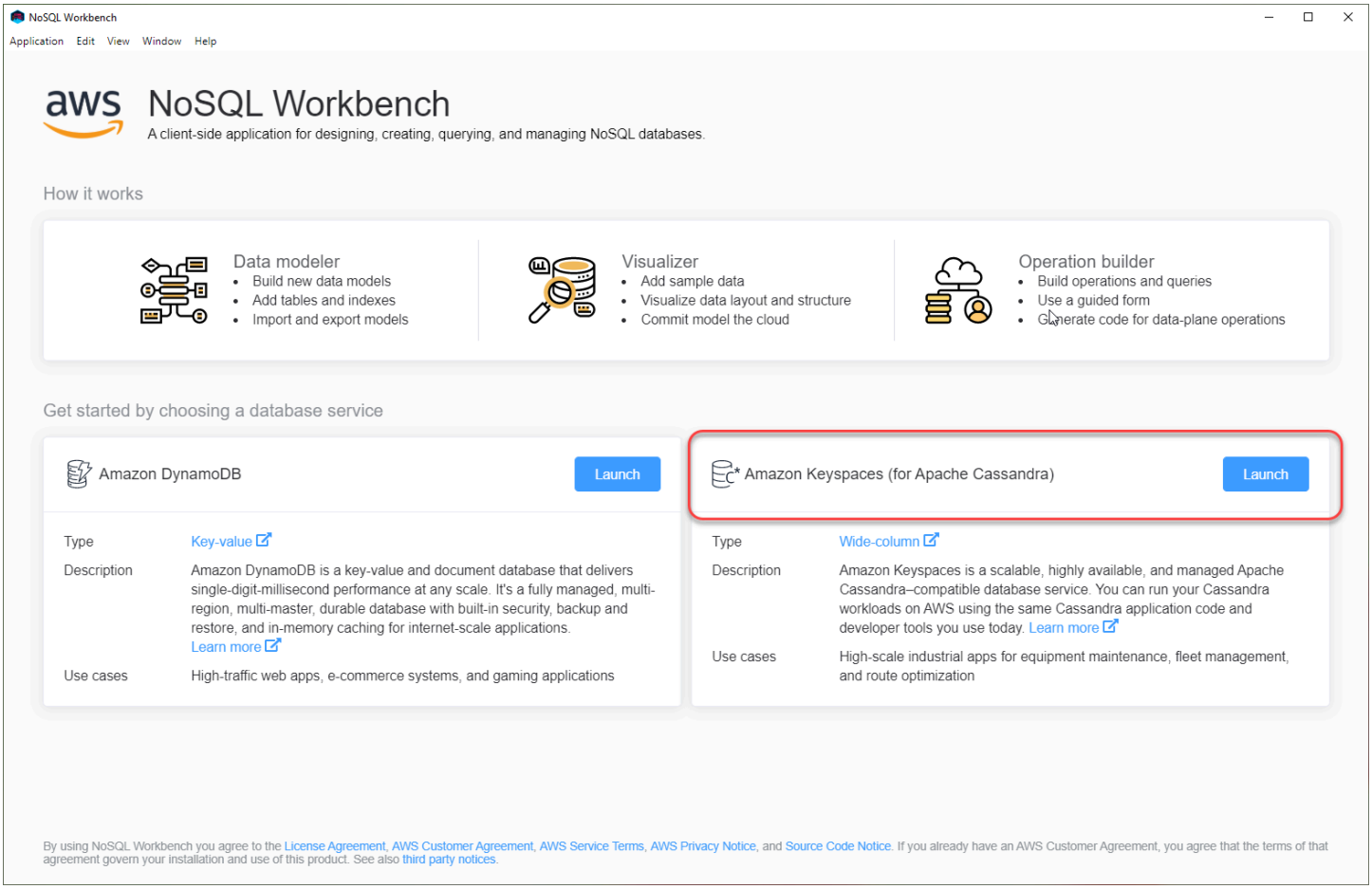
オペレーティングシステム	ダウンロードリンク
macOS	macOS 用のダウンロード
Linux*	Linux 用のダウンロード
Windows	Windows 用のダウンロード

   * 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。

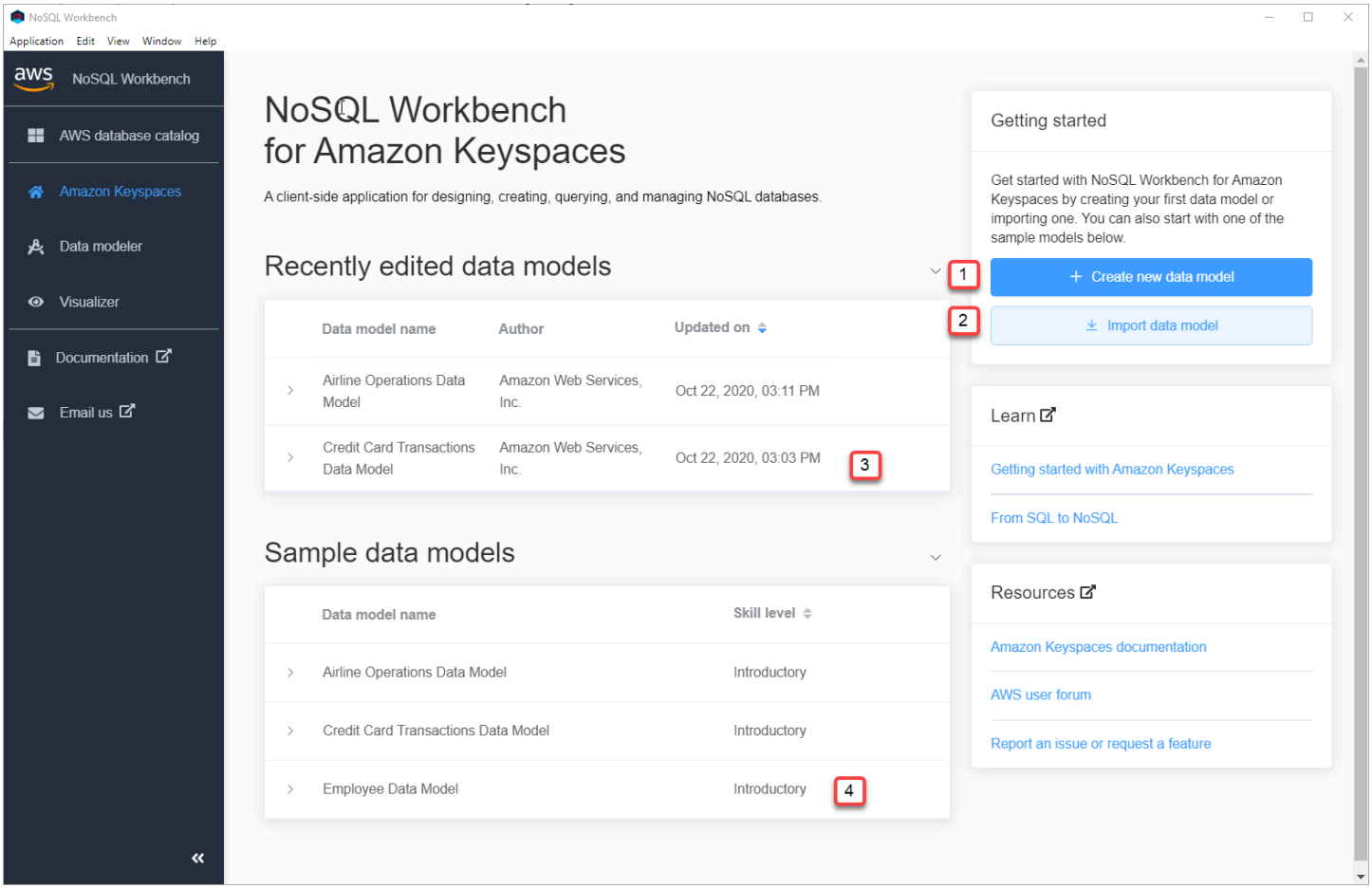
NoSQL Workbench の使用開始

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。

データモデルを構築する方法

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。このデータモデラーを使用すれば、新規モデルの設計、および、NoSQL Workbench を使用して作成された既存データモデルのインポートや変更が可能です。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。

トピック

- NoSQL Workbench を使用した新規データモデルの構築
- NoSQL Workbench での既存のデータモデルの編集

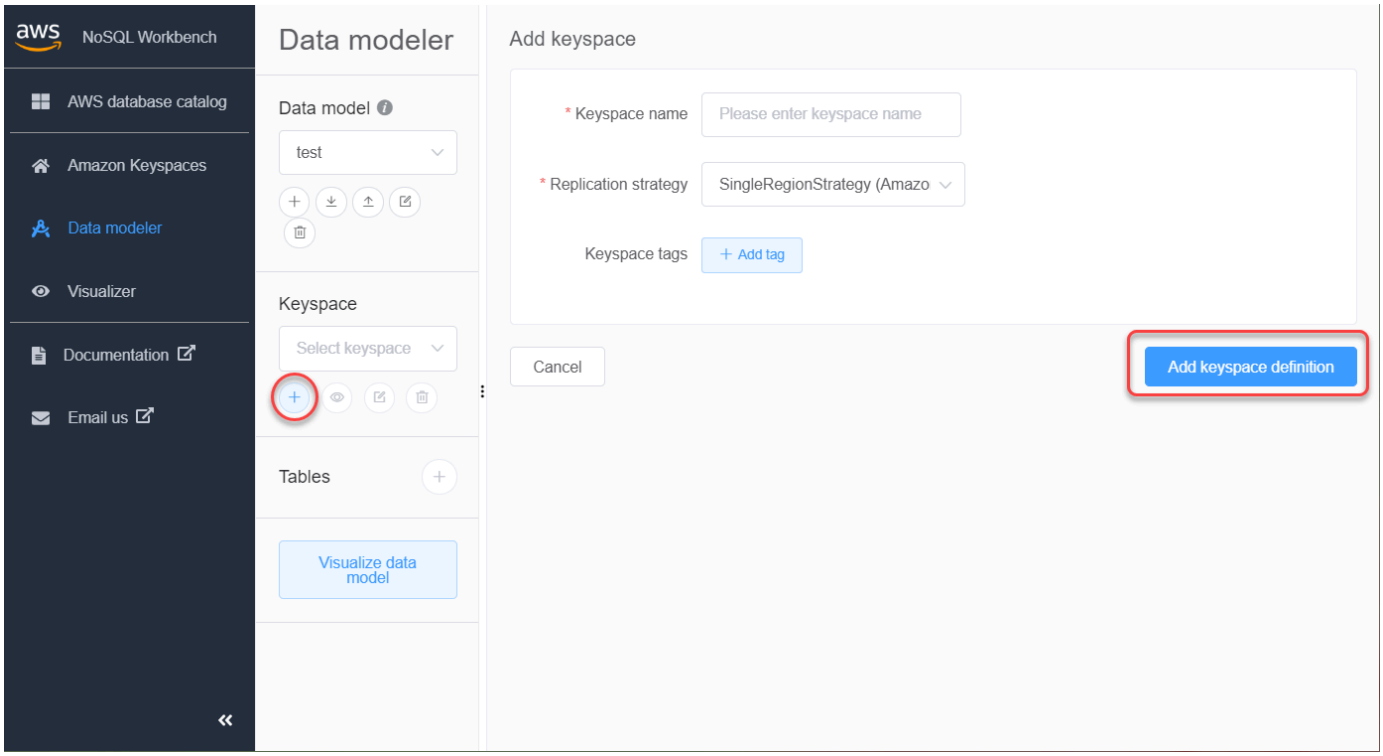
NoSQL Workbench を使用した新規データモデルの構築

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。

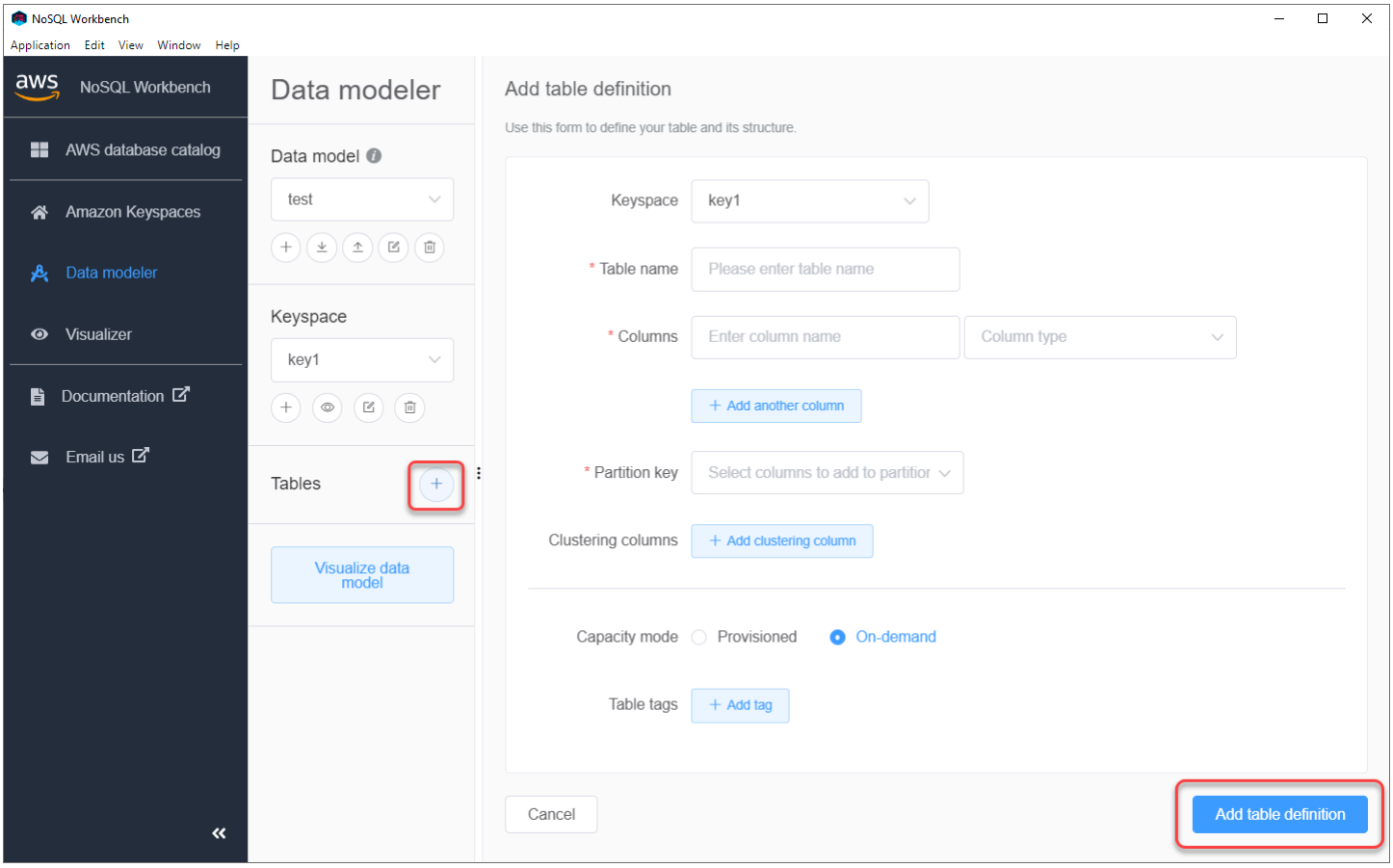
   - [Keyspace name] (キースペース名) — 新しいキースペースの名前を入力します。
   - [Replication strategy] (レプリケーション戦略) - キースペースのレプリケーション戦略を選択します。Amazon Keyspacesでは、[SingleRegionStrategy] を使用して複数のAWS アベイラビリティーゾーンでデータを自動的に 3 回レプリケートします。データモデルを Apache Cassandra クラスターにコミットする予定がある場合は、SimpleStrategy または NetworkTopologyStrategy を選択できます。
   - [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 “タグの使用”」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”」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 “タグの使用”」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 での既存のデータモデルの編集

NoSQL Workbench データモデラーを使用して、Amazon Keyspaces の既存のデータモデルを編集することができます。これらは、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、または、以前に作成したデータモデルです。

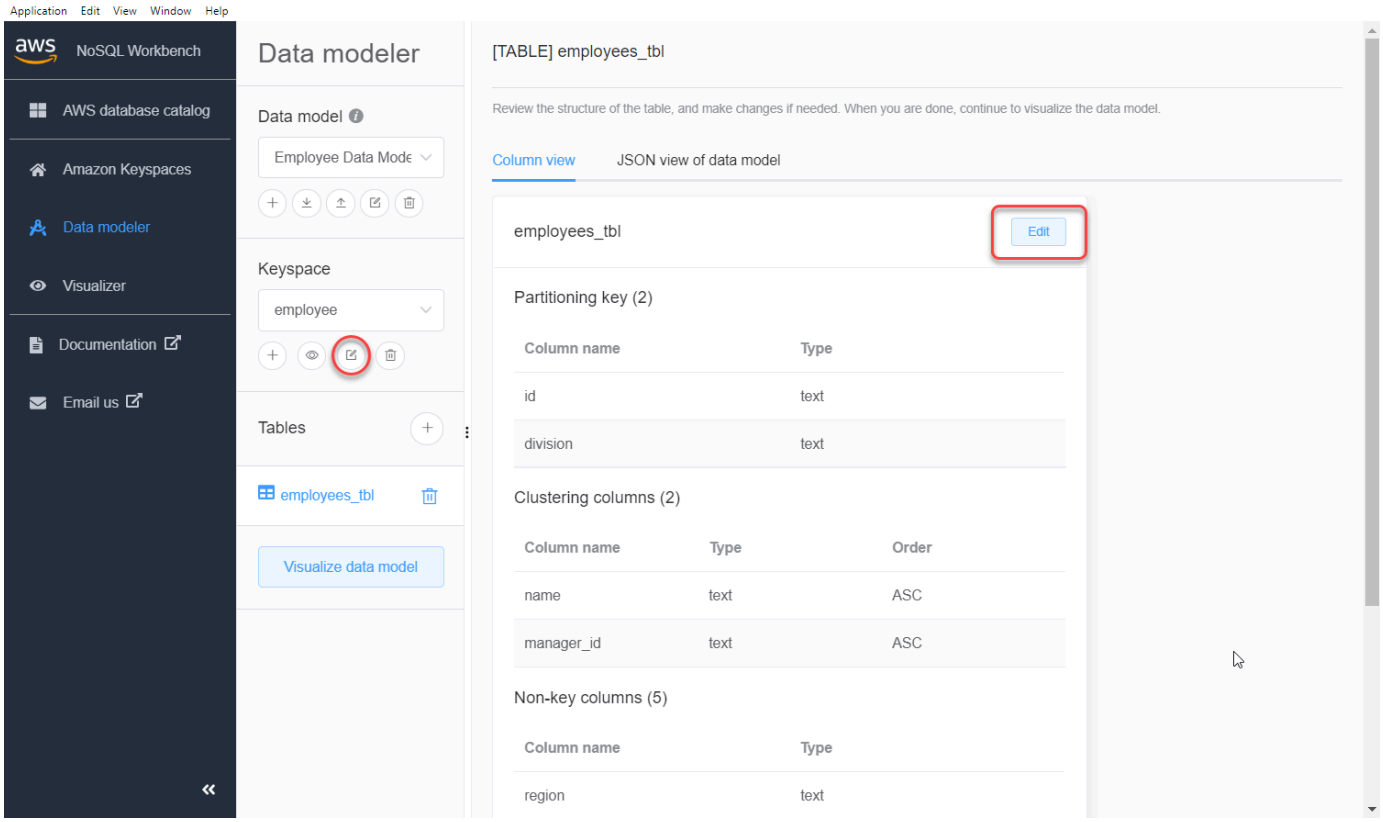
1. キースペースを編集するには、[Keyspace] (キースペース) の下にある編集マークを選択します。

   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では、[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自動的に 3 回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 “タグの使用”」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”」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 “タグの使用”」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。

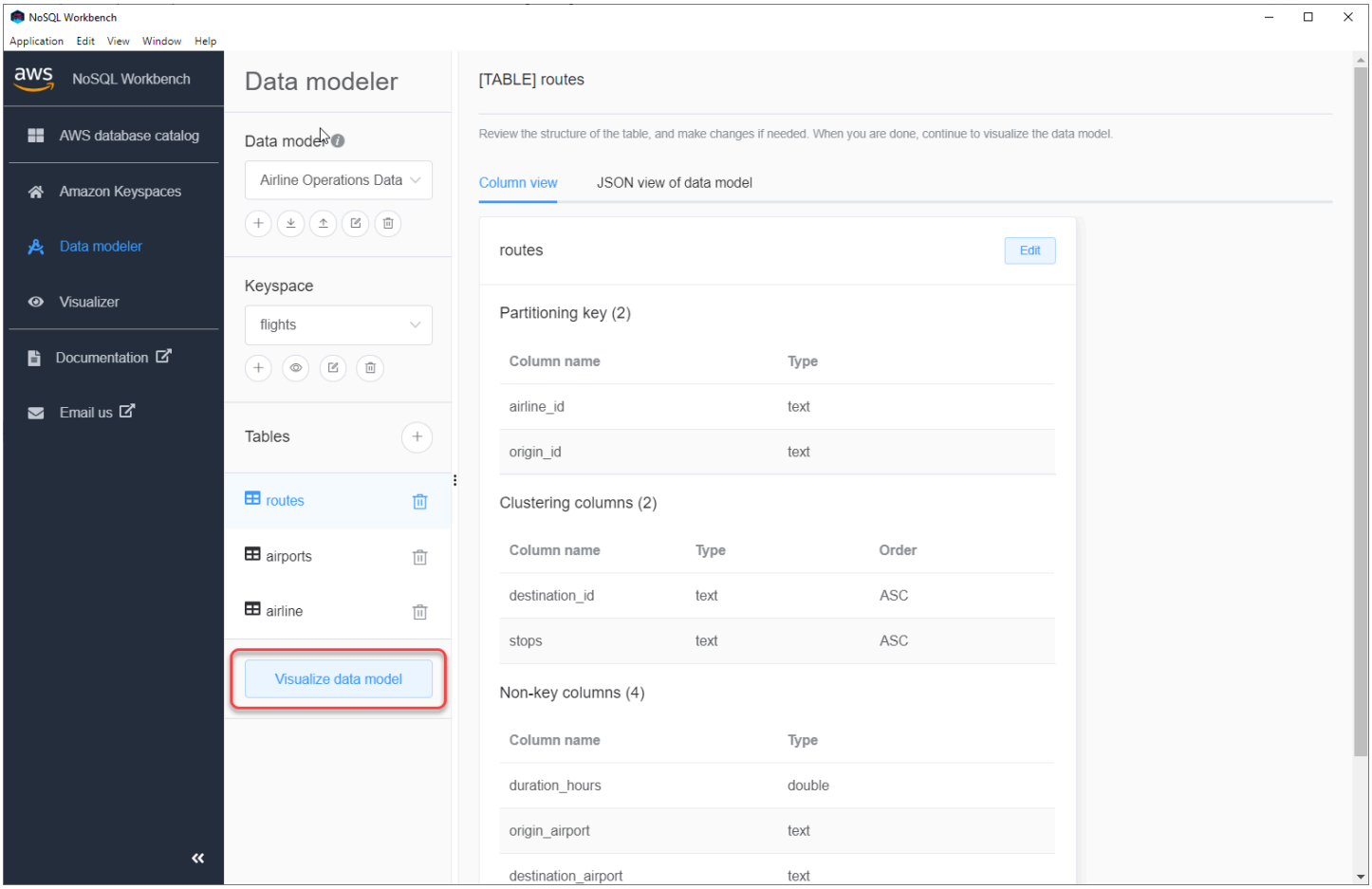
データモデルを可視化する方法

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。

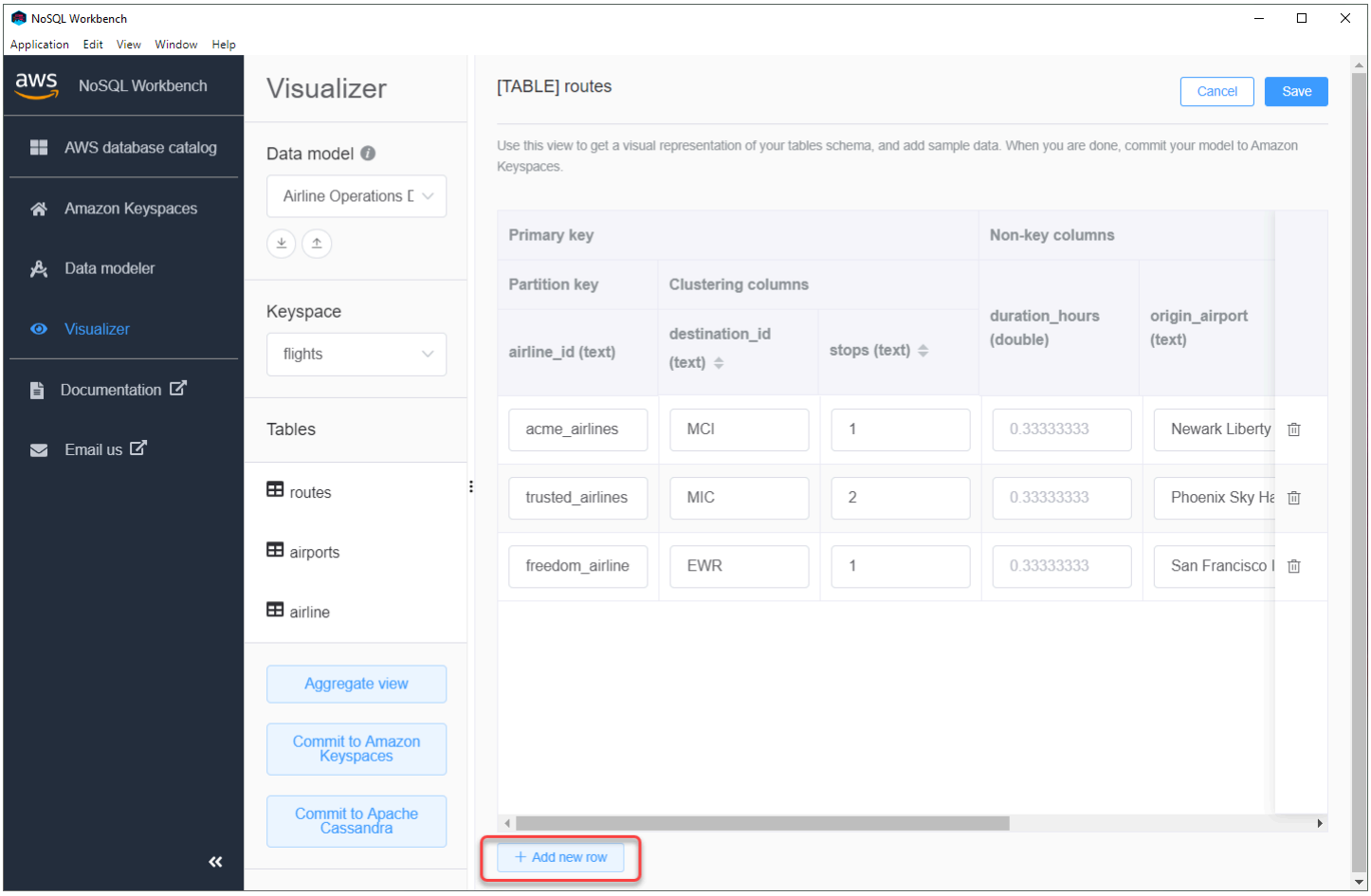
新しいデータモデルの作成後や、既存のデータモデルの編集後に、そのモデルを可視化できます。

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。

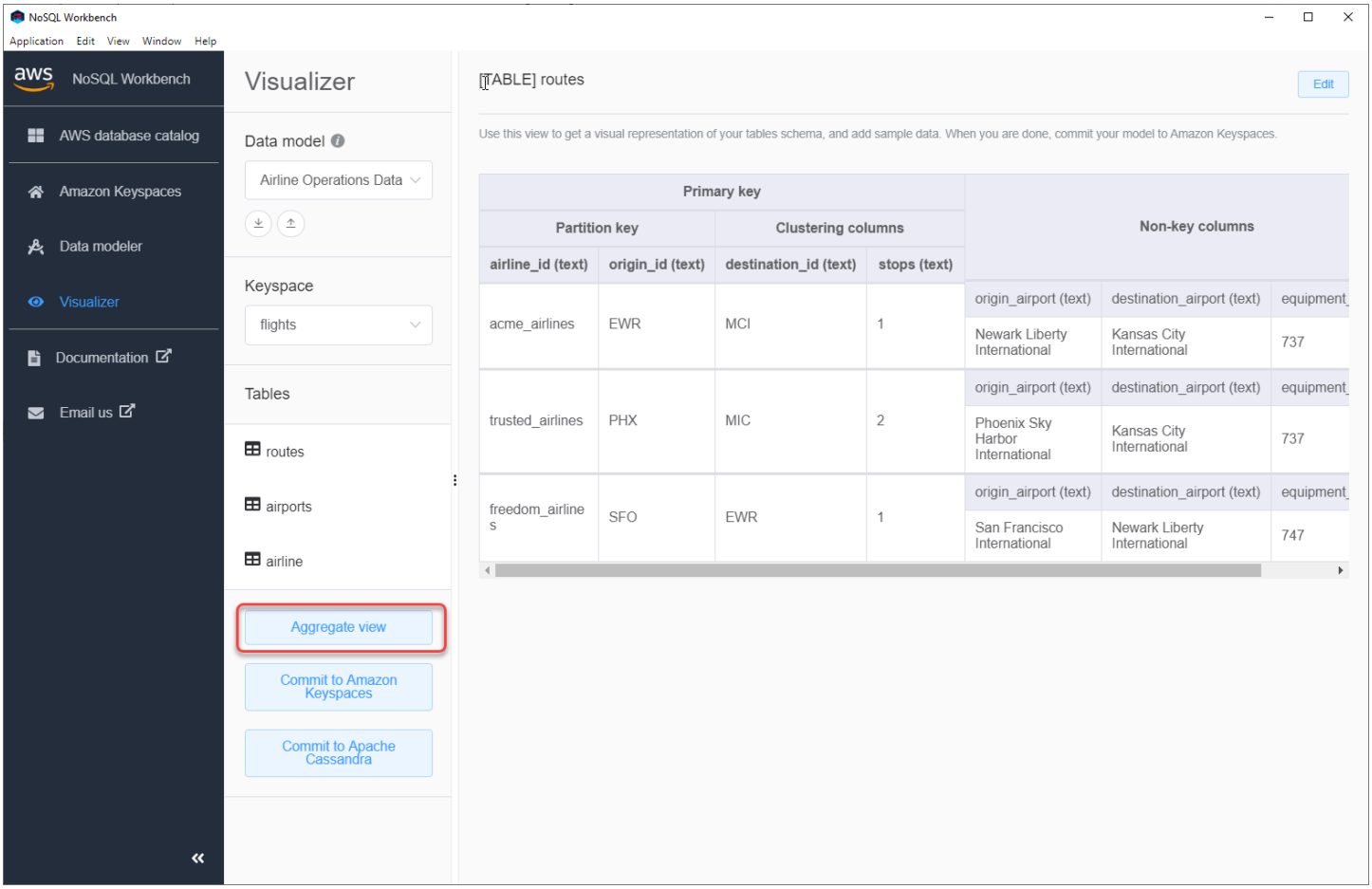


これにより、NoSQL Workbench のデータビジュアライザーに移動します。データビジュアライザーでは、テーブルのスキーマの視覚的表現と、サンプルデータの追加を行います。テーブルにサンプルデータを追加するには、モデルからテーブルを選択し、[Edit] (編集) を選択します。新しいデータ行を追加するには、画面の下部にある [Add new row] (新しい行の追加) を選択します。以上が完了したら、[Save] (保存) を選択します。



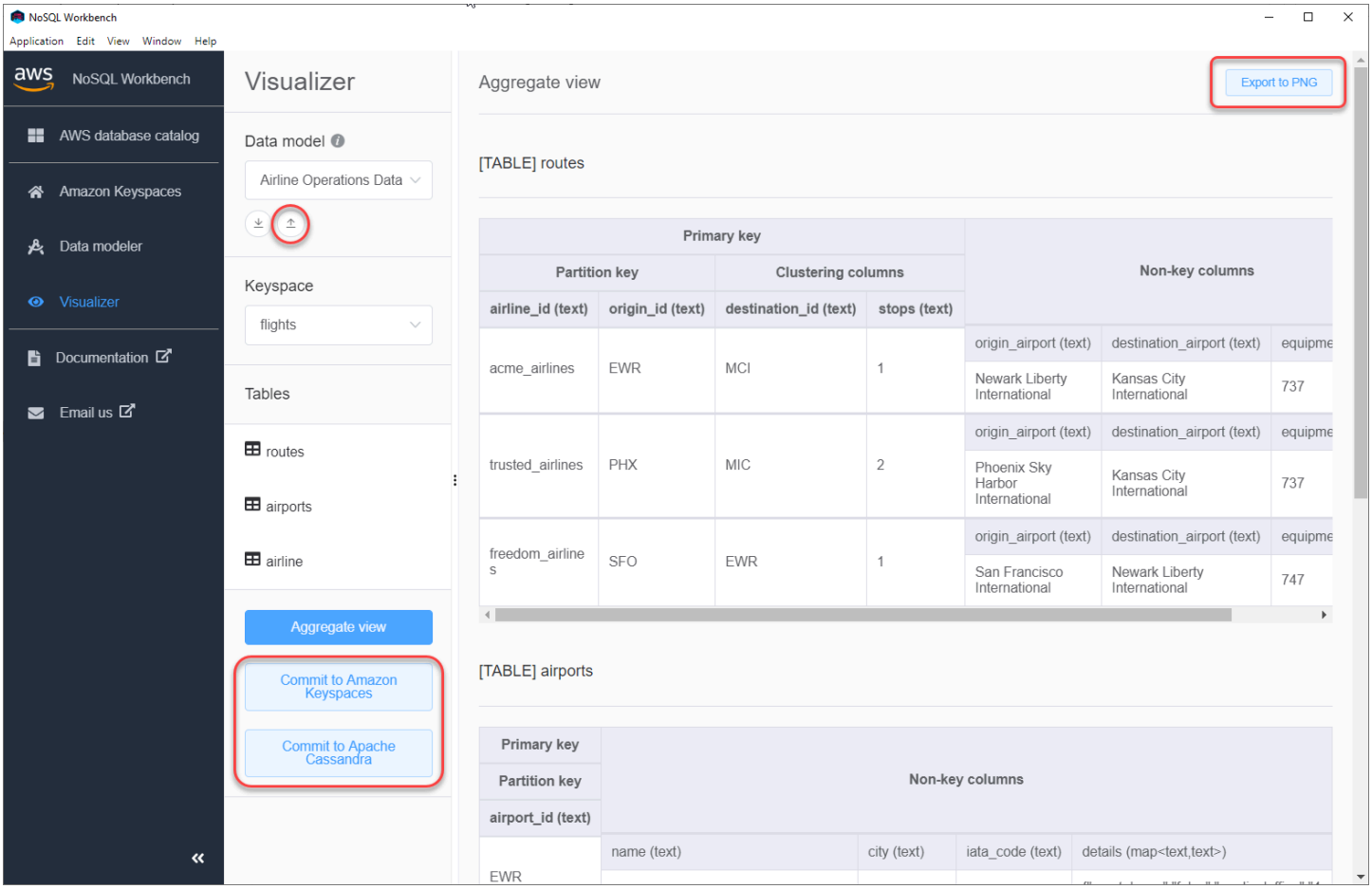
集約ビュー

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。



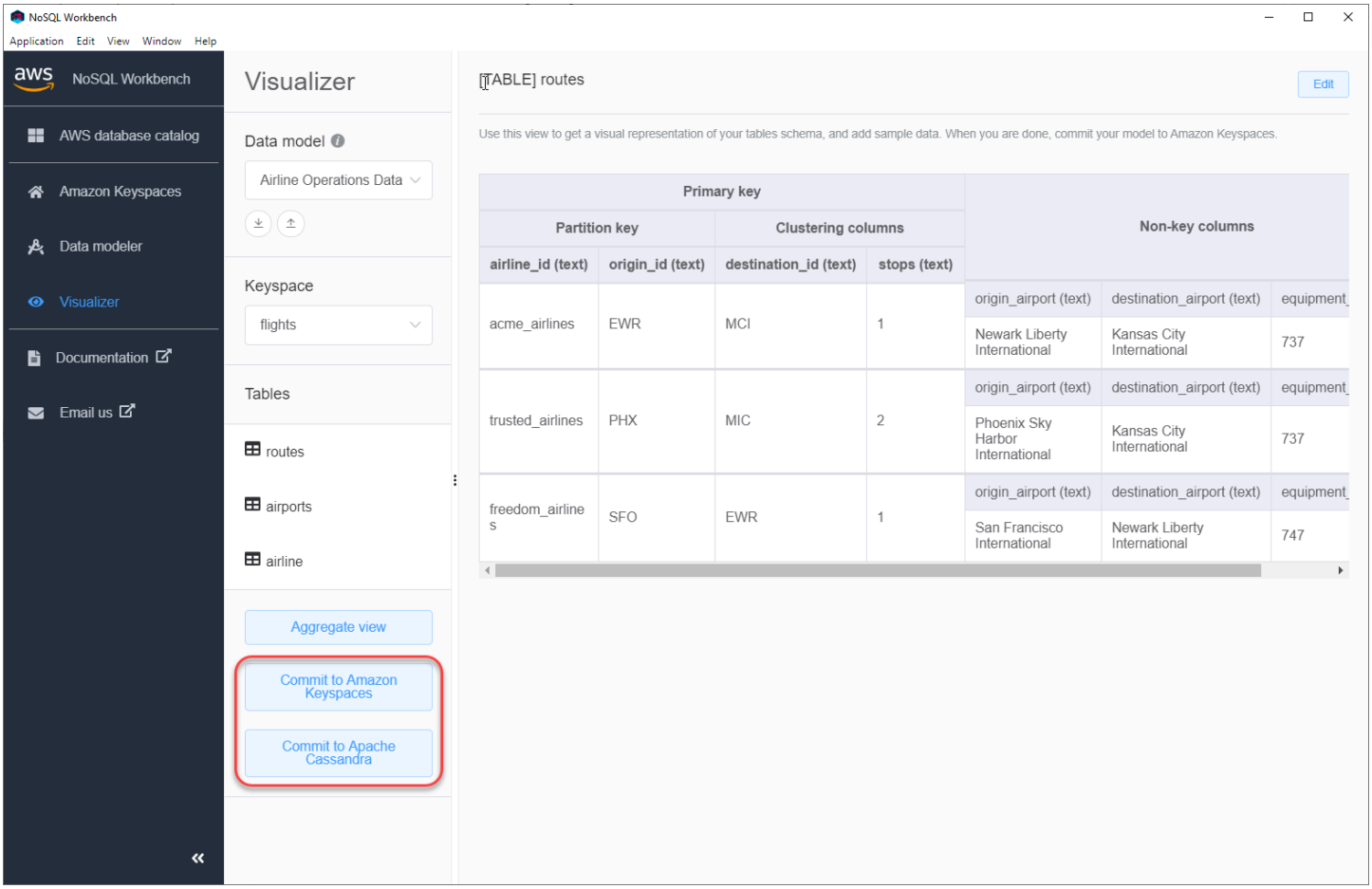
以下は変更をコミットするためのオプションです。

- 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。

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による Amazon Keyspaces への接続
- AWS Identity and Access Management (IAM) 認証情報での Amazon Keyspaces への接続
- 保存済み接続の使用
- Apache Cassandra へのコミット

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 -0
```

> **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 -0
```


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 “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による接続”」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 認証情報での接続”」を参照してください。

サービス固有の認証情報による Amazon Keyspaces への接続

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。

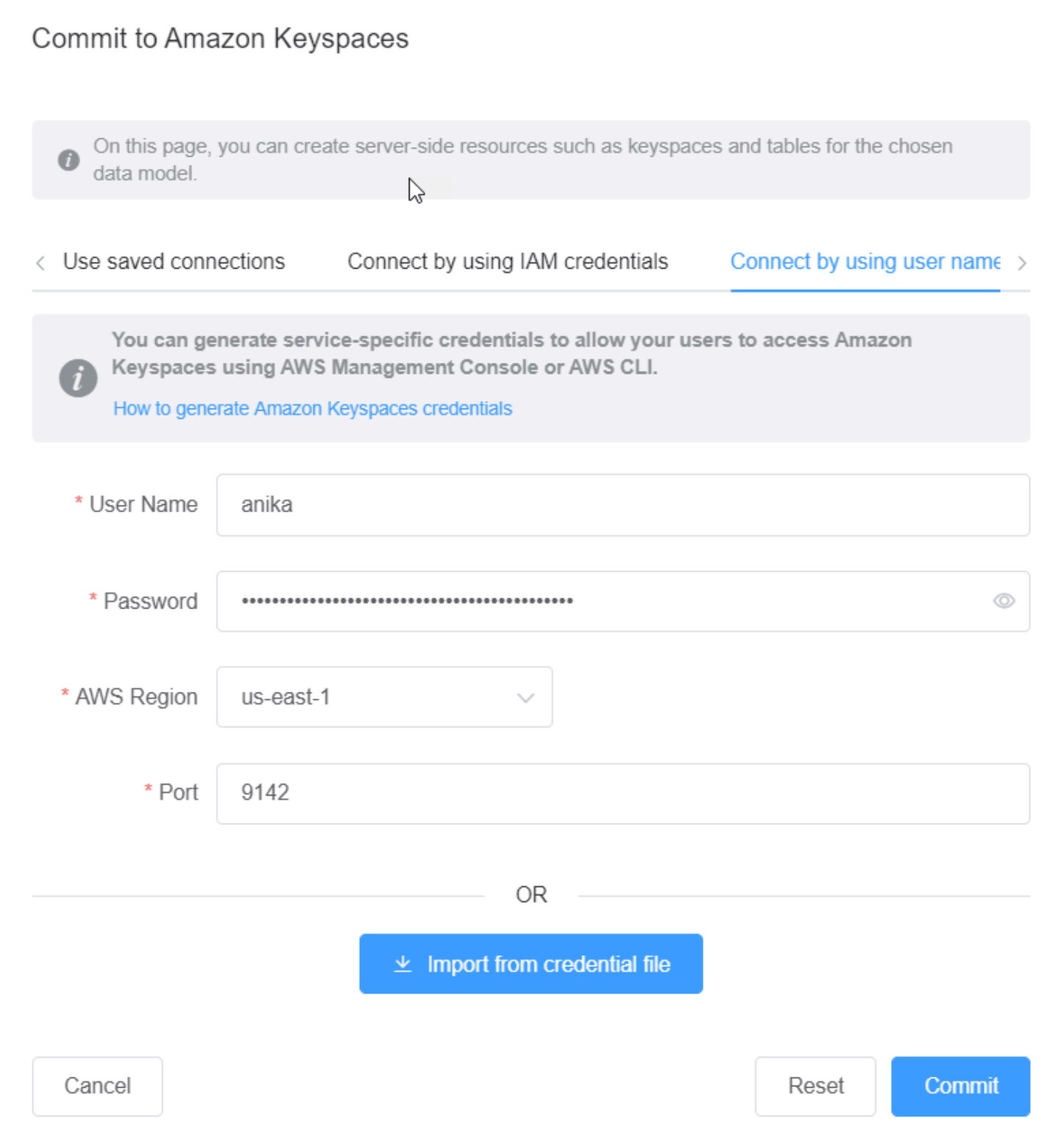
   - 開始する前に、the section called “サービス固有の認証情報” に記載されているプロセスを使用して、サービス固有の認証情報を作成しておく必要があります。

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) 認証情報での Amazon Keyspaces への接続

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認証情報の代替など、AWS セキュリティ認証情報については、「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。AWS アカウント へのサインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS にサインインする方法」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 リソースを使用した一時的なセキュリティ認証情報の使用」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。IAM アクセスキーの詳細については、「IAM ユーザーガイド」の「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。

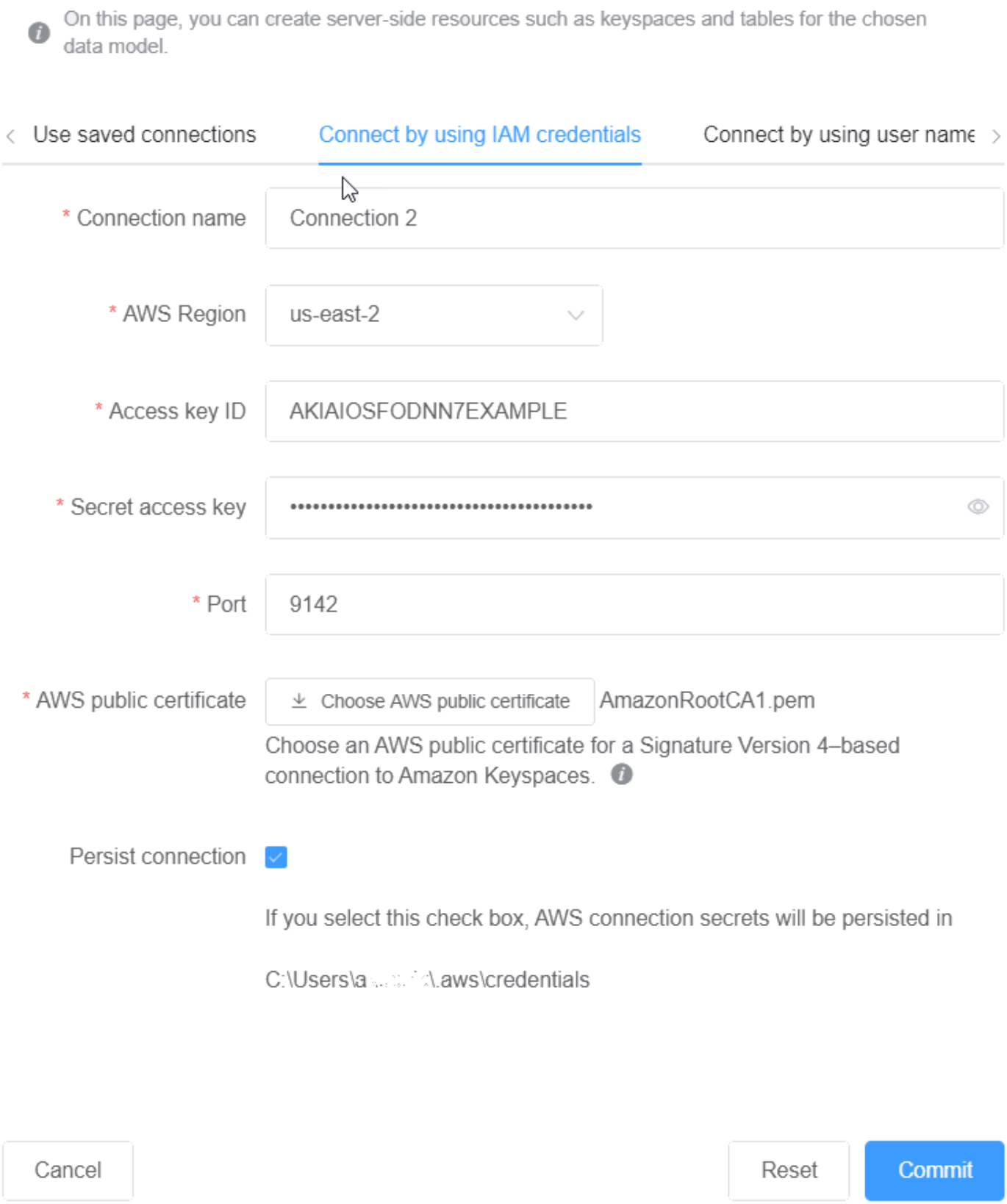
     詳細については、「Managing access keys for IAM users」(IAM ユーザーのアクセスキーの管理) を参照してください。

   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。

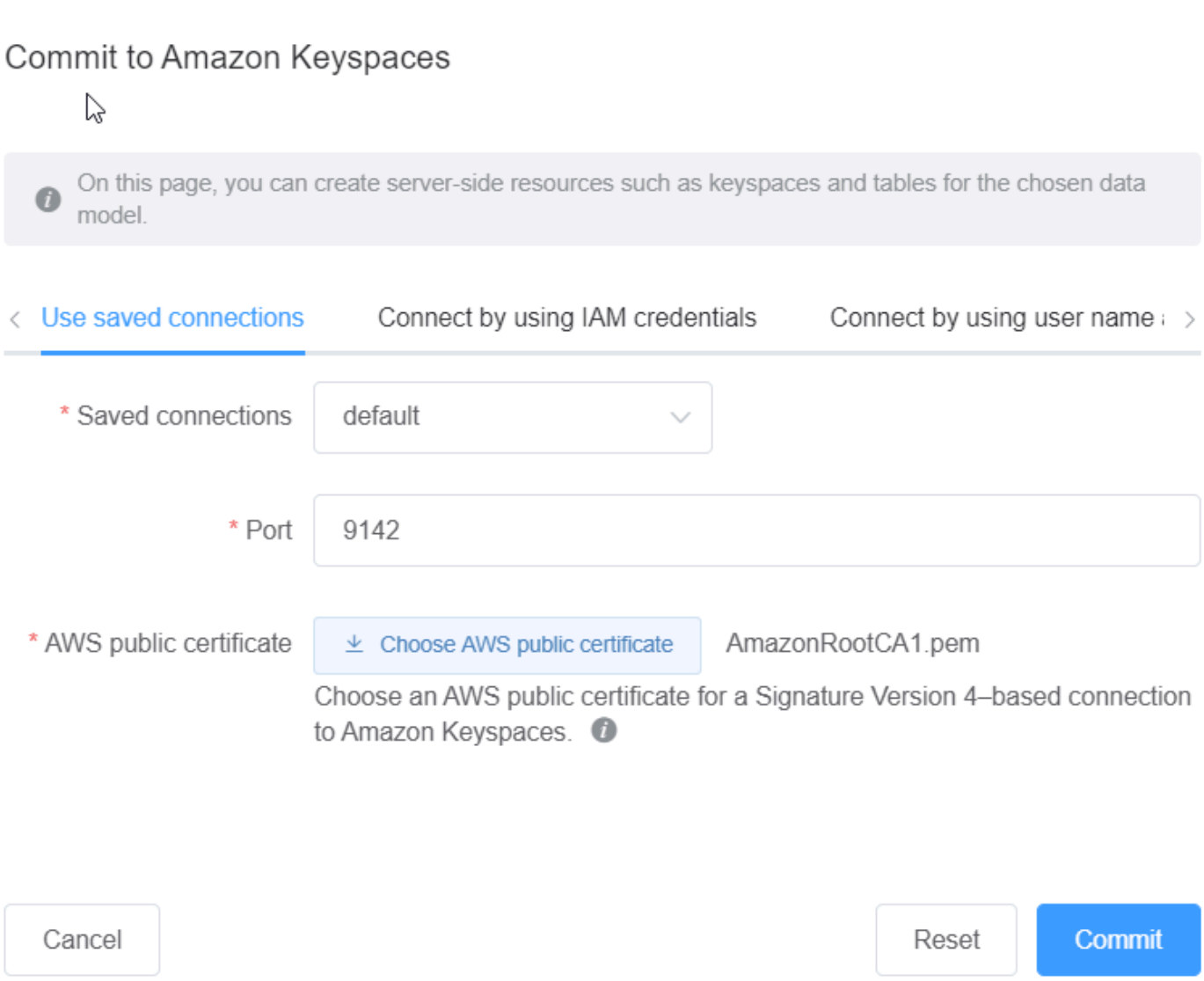
- [Secret access key] (シークレットアクセスキー) – シークレットアクセスキーを入力します。
- ポート — Amazon Keyspaces はポート 9142 を使用します。
- [AWS public certificate] (パブリック証明書) – 最初のステップでダウンロードした AWS 証明書を指します。
- [Persist connection] (永続的接続) — AWS 接続シークレットをローカルに保存する場合はこのチェックボックスを選択します。

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

保存済み接続の使用

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。

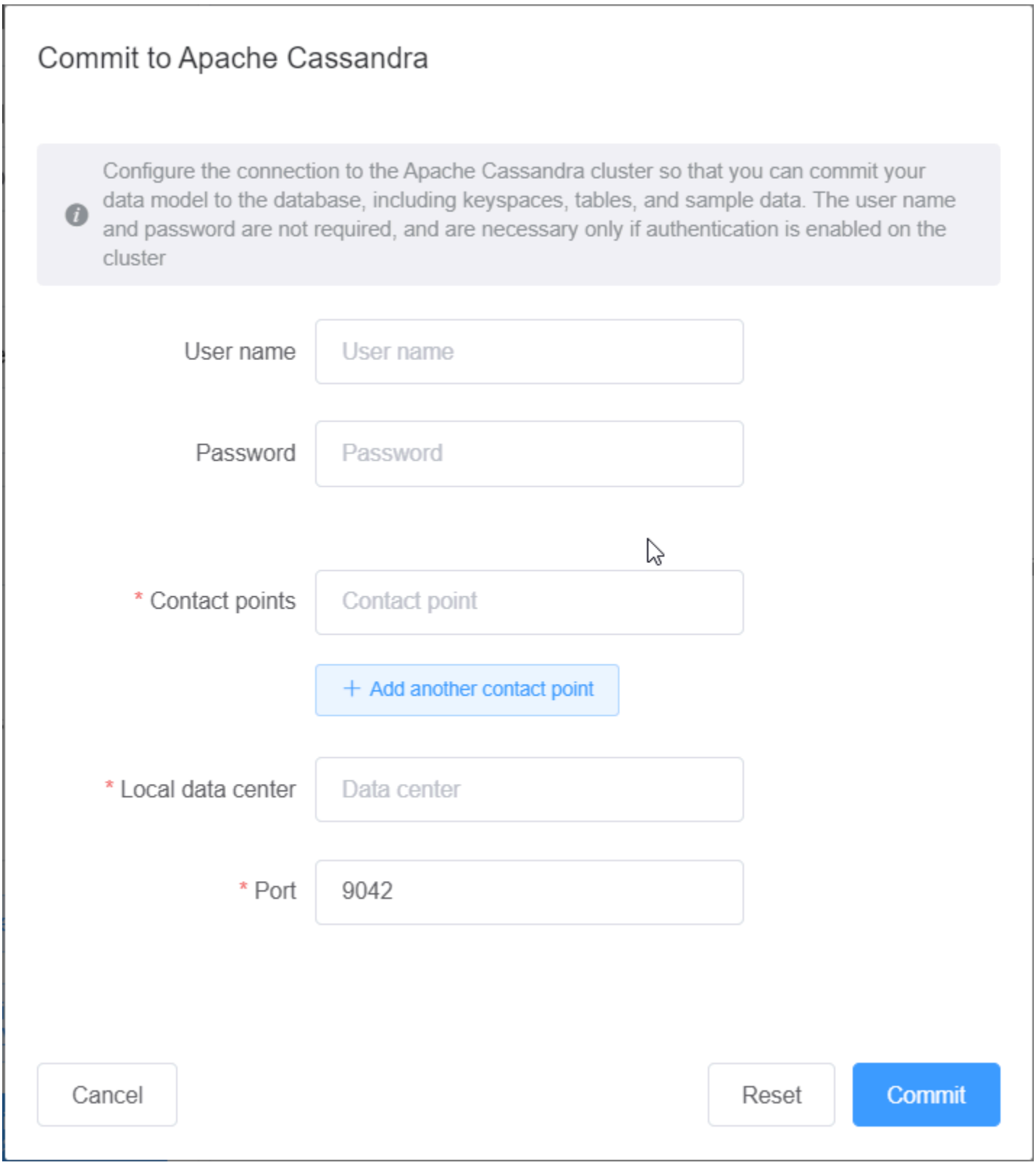


Apache Cassandra へのコミット

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

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。	

Amazon Keyspaces (Apache Cassandra 向け) のマルチリージョンレプリケーション

Amazon Keyspaces マルチリージョンレプリケーションを使用すると、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。グローバルアプリケーションの低レイテンシーのローカル読み取りおよび書き込みのメリットを享受しながら、リージョンの縮小による可用性と耐障害性の両方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ます。データは通常 1 秒以内にリージョン全体に伝達されます。また、マルチリージョンレプリケーションを使用すると、競合の解決や、データ分散問題の解決というやっかいな作業が不要になるため、アプリケーションに集中できます。

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間で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、選択した最大 6 つの異なる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 AWS リージョン 。

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

マルチリージョンレプリケーションには、次の利点があります。

- 1 桁ミリ秒のレイテンシーでのグローバル読み取りおよび書き込み – Amazon Keyspaces では、レプリケーションはアクティブ/アクティブです。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。

- ビジネス継続性と単一リージョンの低下からの保護の向上 — 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、単一の でのパフォーマンス低下から回復できます。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戻ると、Amazon Keyspaces は自動的に不足している変更を同期し、アプリケーションに影響を与えることなく復旧できるようにします。

- 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では、リージョン間のデータの高速ストレージベースの物理レプリケーションを使用します。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。つまり、高書き込みスループットのユースケースやスループットの急増やバーストを伴うユースケースに、アプリケーションに影響を与えることなく対応できます。

- 整合性と競合の解決 — 任意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空間では、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のいずれかを使用できます。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”」を参照してください。

オンデマンドモードでは、1 行あたり最大 1 KB のデータを書き込むために 1.25 書き込みリクエストユニット (WRUs) が課金されます。マルチリージョンキー空間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、リージョンが 2 つのマルチリージョンキースペースに 3 KB のデータの行を書き込むには、3 * 1.25 * 2 = 7.5 WRU の 7.5 WRU 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。

プロビジョンドモードでは、1 行あたり最大 1 KB のデータを書き込むために 1.25 書き込みキャパシティーユニット (WCUs) が課金されます。マルチリージョンキー空間の各リージョンでの書き込みに対して課金されます。例えば、2 つのリージョンを持つマルチリージョンキー空間に 1 秒あたり 3 KB のデータ行を書き込むには、7.5 WCUsが必要です。3 * 1.25 * 2 = 7.5 WCUs 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働き

このセクションでは、Amazon Keyspaces マルチリージョンのレプリケーションの働き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

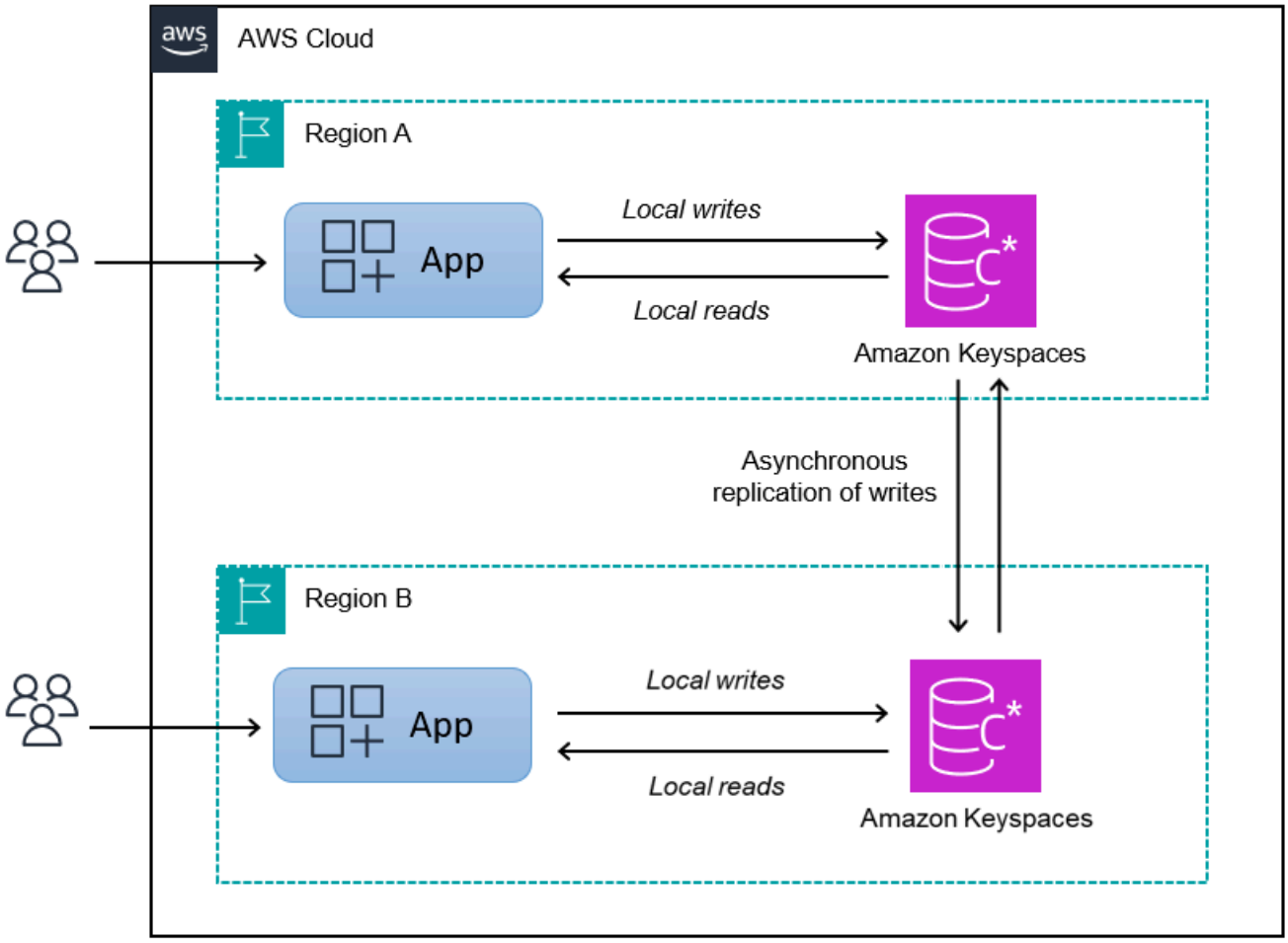
Amazon Keyspaces でのマルチリージョンレプリケーションの働き

Amazon Keyspaces のマルチリージョンレプリケーションは、独立した地理的に分散した AWS リージョンにデータを分散させるデータ復元アーキテクチャを実装しています。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空間を作成すると、データの複製先になる追加リージョンを最大 5 つ選択できます。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。



マルチリージョンのレプリケーション競合の解決

Amazon Keyspaces のマルチリージョンレプリケーションは完全マネージド型であるため、データ同期の問題を解消するために定期的に実行する修復操作などの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 することにより、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、「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の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にわたって非同期的にレプリケートされます。めったにありませんが、1 つのリージョンで機能低下や障害が発生しても、マルチリージョンレプリケーションがあればアプリケーションにほとんどまたはまったく影響を与えることなくディザスターから回復できます。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で、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では、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で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では、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。

マルチリージョンのキー空間とテーブルの作成に必要な IAM 権限

マルチリージョンのキー空間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以下の要素が必要です。

- `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空間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空間とテーブルに対するその他の IAM 権限については、『サービス認証リファレンス』の「Amazon Keyspaces (Apache Cassandra 向け) のアクション、リソース、および条件キー (Apache Cassandra用) 」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と point-in-time リカバリとの統合 (PITR)

P oint-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。

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。

マルチリージョンの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、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 AWS リージョン 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 メトリクスのモニタリング方法の詳細については、「」を参照してくださいthe section called “によるモニタリング CloudWatch”。

Amazon Keyspaces マルチリージョンレプリケーション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、以下の点を考慮してください。

- 使用可能なパブリック 、中国リージョン、およびデフォルトで無効になっている のうち、最大 AWS リージョン 6 つを選択できます。 AWS GovCloud (US) Regions AWS リージョン https://docs.aws.amazon.com/general/latest/gr/rande-manage.html#rande-manage-enable
- キースペースのレプリケーションリージョンは後で追加したり削除したりできないため、慎重に選択してください。
- 後で新しい列を追加することはできないため、マルチリージョンテーブルを作成する前にテーブルスキーマを完成させてください。
- 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、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、以下を参照してください。

  the section called “動作”.
- Amazon Keyspaces の自動スケーリングでプロビジョンドキャパシティ管理を使用する場合は、Amazon Keyspaces 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 がユーザーに代わって呼び出す基盤となる Application Auto Scaling API オペレーションには、マルチリージョン機能はありません。

  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の使用方法”」を参照してください。プロビジョニングされた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見積もる方法の詳細については、「」を参照してくださいthe section called “マルチリージョンテーブル”。
- テーブルに有効期限 (TTL) が必要かどうかを判断します。後でオンにすることはできません。詳細については、「有効期限 (TTL) を使用してデータを期限切れにする」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`整合性をサポートします。 `LOCAL_QUORUM` 整合性では、クライアントに成功を返す前に、ローカルリージョンの 2 つのレプリカに行の更新が永続的に保持される必要があります。その後、レプリケートされたリージョン (またはリージョン) への書き込みの伝播は非同期的に実行されます。

  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 を使用して既存のタグを変更するとALTER TABLE、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。

- Amazon CloudWatch は、レプリケートされた各リージョンのReplicationLatencyメトリクスを提供します。このメトリクスは、到着した行を追跡し、到着時間と最初の書き込み時間を比較し、平均を計算することで計算されます。タイミングは、ソースリージョン CloudWatch の 内に保存されます。詳細については、「the section called “によるモニタリング CloudWatch”」を参照してください。

  平均タイミングと最大タイミングを表示して、平均レプリケーションラグと最悪のレプリケーションラグを判断すると便利です。このレイテンシーには SLA はありません。

- オンデマンドモードでマルチリージョンテーブルを使用する場合、テーブルレプリカに新しいトラフィックピークが発生した場合、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ことがあります。Amazon Keyspaces が単一リージョンのオンデマンドテーブルの容量を受信したアプリケーショントラフィックに自動的に適応させるのと同様に、Amazon Keyspaces はマルチリージョンのオンデマンドテーブルレプリカの容量を受信したトラフィックに自動的に適応させます。Amazon Keyspaces はトラフィック量の増加に応じて自動的により多くの容量を割り当てるため、レプリケーションレイテンシーの増加は一時的なものです。すべてのレプリカがトラフィックボリュームに適応すると、レプリケーションのレイテンシーは通常に戻ります。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。
- プロビジョニングモードでマルチリージョンテーブルを使用する場合、アプリケーションがプロビジョニングされたスループットキャパシティを超えると、容量不足エラーが発生し、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常に十分な読み取りおよび書き込み容量を確保するために、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ことで、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のに役立ちます。詳細については、「the section called “マルチリージョンテーブルでのauto スケーリングの仕組み”」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使用方法

Amazon Keyspaces (Apache Cassandra 向け) コンソール、Cassandra クエリ言語 (CQL)、 AWS SDK、 AWS Command Line Interface および () を使用して、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AWS CLI。

このセクションでは、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両方を使用して AWS CLI、 コンソール、CQL、および でマルチリージョンキースペースとテーブルを作成する方法の例を示します。マルチリージョンキー空間で作成されたすべてのテーブルは、キー空間からマルチリージョン設定を自動的に継承します。

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 Auto Scaling 設定 AWS CLI を管理する方法の例も示します。一般的な Auto Scaling 設定オプションとその仕組みの詳細については、「」を参照してくださいthe section called “auto スケーリングによるスループット容量の管理”。

マルチリージョンテーブルに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。これは、基盤となる Application Auto Scaling API オペレーションがリージョン対応ではないためです。

プロビジョニングされた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見積もる方法の詳細については、「」を参照してくださいthe section called “マルチリージョンテーブル”。

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。

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”。

トピック

- コンソールを使用したマルチリージョンテーブルの作成と管理
- CQL を使用したマルチリージョンテーブルの作成と管理
- を使用したマルチリージョンテーブル AWS CLI の作成と管理

コンソールを使用したマルチリージョンテーブルの作成と管理

このセクションでは、Amazon Keyspaces (Apache Cassandra 向け) コンソールを使用して、オンデマンドおよびプロビジョンドキャパシティモードでマルチリージョンキースペースとテーブルを作成する方法の例を示します。マルチリージョンキースペースで作成したすべてのテーブルは、キースペースからマルチリージョン設定を自動的に継承します。

CQL の例については、「」を参照してくださいthe section called “CQL の使用”。 AWS CLI 例については、「」を参照してくださいthe section called “の使用 AWS CLI”。

トピック

- マルチリージョンキー空間の作成 (コンソール)
- デフォルト設定でマルチリージョンテーブルを作成する (コンソール)
- Auto Scaling を有効にして、プロビジョンドモードでマルチリージョンテーブルを作成する (コンソール)
- 既存のマルチリージョンテーブルの Auto Scaling の有効化 (コンソール)
- マルチリージョンテーブルの Auto Scaling をオフにする (コンソール)
- コンソールでの Amazon Keyspaces Auto Scaling アクティビティの表示

マルチリージョンキー空間の作成 (コンソール)

Amazon Keyspaces コンソール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には、次の手順に従います。

マルチリージョンキー空間 (コンソール) を作成するには (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 (キー空間)] を選択し、次に [Create keyspace (キー空間の作成)] を選択します。
3. [Keyspace name (キー空間名)] でキー空間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、リストに表示されているリージョンを最大 5 つまで追加できます。
5. 終了するには、[ロールを作成] を選択します。

> **Note**
>
> マルチリージョンキー空間を作成すると、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。

デフォルト設定でマルチリージョンテーブルを作成する (コンソール)

Amazon Keyspaces コンソールでマルチリージョンテーブルは、次のステップで作成します。

マルチリージョンテーブルを作成するには (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。プロビジョニングモードの設定の詳細については、「」を参照してくださいthe section called “Auto Scaling を有効にして、プロビジョンドモードでマルチリージョンテーブルを作成する (コンソール) ”。
   - 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。

Auto Scaling を有効にして、プロビジョンドモードでマルチリージョンテーブルを作成する (コンソール)

> **Note**
>
> Amazon Keyspaces のオートスケーリングでは、ユーザーに代わってオート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 “サービスリンクロールの使用”」を参照してください。

自動スケーリングを有効にして新しいマルチリージョン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. 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、キー空間を選択し、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、オプションの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 (オートスケーリング)] が選択されているか確認します。

    テーブルがレプリケートされるすべての AWS リージョン に対して、同じ読み込みキャパシティーユニットを設定することもできます。または、チェックボックスをオフにして、各リージョンの読み込み容量を異なる方法で設定できます。

    各リージョンを異なる方法で設定することを選択した場合は、各テーブルレプリカの最小読み込み容量ユニットと最大読み込み容量ユニット、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。

- テーブルの読み込み容量を手動でプロビジョニングする場合は、スケールの自動チェックボックスをオフにします。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込みキャパシティを確保するために、プロビジョニングされたマルチリージョンテーブルには Amazon Keyspaces のオートスケーリングをお勧めします。

> **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. 「容量の書き込み」セクションで、スケーリングが自動的に選択されていることを確認します。次に、テーブルのキャパシティーユニットを設定します。書き込みキャパシティーユニットは、リージョン間で書き込みイベント AWS リージョン をレプリケートするのに十分なキャパシティーを確保するために、すべての で同期されたままになります。

- 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、自動的にスケーリングを解除します。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、プロビジョニングされたマルチリージョンテーブルには Amazon Keyspaces のオートスケーリングをお勧めします。

12. [Create table (テーブルの作成)] を選択します。指定したオート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。

既存のマルチリージョンテーブルの Auto Scaling の有効化 (コンソール)

Amazon Keyspaces コンソールを使用して、プロビジョニングモードでマルチリージョンテーブルの Auto Scaling を有効にするには、次の手順に従います。

> **Note**
>
> Amazon Keyspaces のオートスケーリングでは、ユーザーに代わってオート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 “サービスリンクロールの使用”」を参照してください。

既存のマルチリージョンテーブルに対して Amazon Keyspaces オートスケーリングを有効にするには

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. 使用するテーブルを選択し、キャパシティ タブに移動します。
3. 「容量設定」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. 自動的にスケーリング を選択し、Auto Scaling を有効にして、プロビジョンドモードでマルチリージョンテーブルを作成する (コンソール) 「」のステップ 9 を参照して読み取りおよび書き込み容量を編集します。
6. オートスケーリング設定が定義されたら、[Save (保存)] を選択します。

マルチリージョンテーブルの Auto Scaling をオフにする (コンソール)

Amazon Keyspaces コンソールを使用して、プロビジョンドモードでマルチリージョンテーブルの Auto Scaling をオフにするには、次の手順に従います。

既存のマルチリージョンテーブルの Amazon Keyspaces オートスケーリングを無効にするには

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. 使用するテーブルを選択し、キャパシティタブを選択します。
3. 「容量設定」セクションで、「編集」を選択します。
4. Amazon Keyspaces のオートスケーリングを無効にするには、「スケールの自動スケーリング」チェックボックスをオフにします。自動スケーリングを無効にすると、Application Auto Scaling でテーブルがスケーラブルターゲットとして登録解除されます。Application Auto Scaling が Amazon Keyspaces テーブルにアクセスするために使用するサービスにリンクされたロールを削除するには、「」のステップに従いますthe section called “Amazon Keyspaces のサービスリンクロールの削除”。

   > **Note**
   >
   > Application Auto Scaling が使用するサービスにリンクされたロールを削除するには、すべての でアカウント内のすべてのテーブルのオートスケーリングを無効にする必要があります AWS リージョン。

5. オートスケーリング設定が定義されたら、[Save (保存)] を選択します。

コンソールでの Amazon Keyspaces Auto Scaling アクティビティの表示

Amazon Keyspaces オートスケーリングがリソースをどのように使用するかをモニタリングするには、Amazon を使用します。これにより CloudWatch、使用状況とパフォーマンスに関するメトリクスが生成されます。Application Auto Scaling ユーザーガイドの手順に従ってダッシュボードを作成します CloudWatch。

CQL を使用したマルチリージョンテーブルの作成と管理

Cassandra クエリ言語 (CQL) を使用して、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、CQL を使用してマルチリージョンテーブルを作成および管理する方法の例を示します。マルチリージョンキースペースで作成したすべてのテーブルは、キースペースからマルチリージョン設定を自動的に継承します。CQL の詳細については、「Amazon Keyspaces CQL 言語リファレンス」を参照してください。

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

- マルチリージョンキー空間の作成 (CQL)
- デフォルト設定でマルチリージョンテーブルを作成する (CQL)
- プロビジョンドキャパシティモードと Auto Scaling (CQL) を使用したマルチリージョンテーブルの作成
- マルチリージョンテーブル (CQL) 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の更新
- マルチリージョンテーブル (CQL) のプロビジョンドキャパシティとオートスケーリング設定の表示
- マルチリージョンテーブル (CQL) の Auto Scaling をオフにする
- 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 (CQL)

マルチリージョンキー空間の作成 (CQL)

マルチリージョンキー空間を作成するには、 を使用してキー空間をレプリケート AWS リージョンする NetworkTopologyStrategy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。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' };
```

キー空間内のすべてのテーブルは、キー空間と同じレプリケーション戦略を使用します。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間で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。

> **Note**
>
> マルチリージョンキー空間を作成すると、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。

CQL ステートメントを使用してキーsystem_multiregion_info空間のtablesテーブルをクエリし、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で一覧表示できます。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';
```

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
```

デフォルト設定でマルチリージョンテーブルを作成する (CQL)

デフォルト設定でマルチリージョンテーブルを作成するには、次の例を使用できます。

```
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'
```

```
 }
};
```

プロビジョンドキャパシティモードと Auto Scaling (CQL) を使用したマルチリージョンテーブルの作成

Auto Scaling を使用してプロビジョンドモードでマルチリージョンテーブルを作成するには、まずCUSTOM_PROPERTIESテーブルの を定義してキャパシティモードを指定する必要があります。プロビジョンドキャパシティモードを指定した後、 を使用してテーブルの Auto Scaling 設定を構成できますAUTOSCALING_SETTINGS。

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、およびオプション設定の詳細については、「」を参照してくださいthe section called “CQL を使用して自動スケーリングで新しいテーブルを作成する”。

マルチリージョンテーブルを作成するときに、テーブルのレプリカごとに異なる読み込み容量と読み込み自動スケーリング設定を指定することもできます。指定した設定は、指定した のテーブルの一般的な設定を上書きします AWS リージョン。ただし、書き込み容量はすべてのレプリカ間で同期されたままになり、すべてのリージョンに書き込みをレプリケートするのに十分な容量が確保されます。

特定のリージョンのテーブルレプリカの読み取り容量を定義するには、テーブルの の一部として以下のパラメータを設定できますreplica_updates。

- リージョン
- プロビジョニングされた読み込みキャパシティーユニット (オプション)
- 読み込み容量の Auto Scaling 設定 (オプション)

次の例は、プロビジョニングモードのマルチリージョンテーブルの CREATE TABLEステートメントを示しています。書き込みキャパシティーと読み取りキャパシティーの自動スケーリングの一般的な設定は同じです。ただし、読み取り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ーをスケールアップまたはスケールダウンする前に、60 秒の追加クールダウン期間を指定します。さらに、米国東部 (バージニア北部) リージョンの読み込みキャパシティーの自動スケーリング設定は、他のレプリカの設定よりも高くなります。また、ターゲット値は 50% ではなく 70% に設定されます。

```
CREATE TABLE mykeyspace.mytable(pk int, ck int, PRIMARY KEY (pk, ck))
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
```

```
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 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
                    }
                }
            }
        }
    }
};
```

マルチリージョンテーブル (CQL) 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の更新

を使用してALTER TABLE、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設定を更新できます。現在オンデマンドキャパシティモードになっているテーブルを更新する場合は、capacity_mode が必要です。テーブルがすで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は、このフィールドを省略できます。

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、およびオプション設定の詳細については、「」を参照してくださいthe section called “CQL を使用して自動スケーリングで新しいテーブルを作成する”。

同じステートメントで、テーブルの replica_updatesプロパティを更新することで、特定のリージョンのテーブルレプリカの読み取り容量と自動スケーリング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
                    }
                }
            }
        }
    }
};
```

マルチリージョンテーブル (CQL) のプロビジョンドキャパシティとオートスケーリング設定の表示

マルチリージョンテーブルの Auto Scaling 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}}}
```

マルチリージョンテーブル (CQL) の Auto Scaling をオフにする

を使用してALTER TABLE、既存のテーブルの Auto Scaling をオフにすることができます。個々のテーブルレプリカの Auto Scaling はオフにできないことに注意してください。

次の例では、テーブルの読み込みキャパシティーの Auto Scaling がオフになっています。

```
ALTER TABLE mykeyspace.mytable
WITH AUTOSCALING_SETTINGS = {
    'provisioned_read_capacity_autoscaling_update': {
        'autoscaling_disabled': true
    }
};
```

> **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オートスケーリングを無効にする必要があります。

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 (CQL)

マルチリージョンテーブルの Auto Scaling をオフにする必要がある場合は、 ALTER TABLEを使用してレプリカテーブルの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
         }
    }
};
```

> **Note**
>
> プロビジョニングされた容量を使用するマルチリージョンテーブルには、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”」を参照してください。

を使用したマルチリージョンテーブル AWS CLI の作成と管理

AWS Command Line Interface（ AWS CLI) を使用して、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、 を使用してマルチリージョンテーブルを作成および管理する方法の例を示します AWS CLI。マルチリージョンキースペースで作成したすべてのテーブルは、キースペースからマルチリージョン設定を自動的に継承します。

このトピックで説明する Amazon Keyspaces AWS CLI コマンドの詳細については、AWS CLI「Amazon Keyspaces の コマンドリファレンス」を参照してください。

トピック

- 新しいマルチリージョンキー空間の作成 (CLI)
- デフォルト設定で新しいマルチリージョンテーブルを作成する (CLI)
- Auto Scaling によるプロビジョニングモードでの新しいマルチリージョンテーブルの作成 (CLI)
- 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 Auto Scaling 設定の更新 (CLI)
- 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オートスケーリング設定の表示 (CLI)

- 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 (CLI)
- 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 (CLI)

新しいマルチリージョンキー空間の作成 (CLI)

マルチリージョンキー空間は、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
```

> **Note**
>
> マルチリージョンキー空間を作成すると、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。

デフォルト設定で新しいマルチリージョンテーブルを作成する (CLI)

デフォルト設定でマルチリージョンテーブルを作成するには、スキーマを指定するだけで済みます。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}'
```

コマンドの出力は次のとおりです。

```
{
    "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/
table/mytable"
}
```

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すべてのデフォルト設定が表示されます。

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

Auto Scaling によるプロビジョニングモードでの新しいマルチリージョンテーブルの作成 (CLI)

Auto Scaling 設定を使用してプロビジョニングモードでマルチリージョンテーブルを作成するには、を使用できます AWS CLI。マルチリージョンの自動スケーリング設定を構成するには、Amazon Keyspaces CLI `create-table` コマンドを使用する必要があります。これは、Amazon Keyspaces がユーザーに代わって自動スケーリングを実行するために使用するサービスである Application Auto Scaling が、複数のリージョンをサポートしていないためです。

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、およびオプション設定の詳細については、「」を参照してくださいthe section called “を使用して自動スケーリングで新しいテーブルを作成する AWS CLI”。

Auto Scaling 設定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場合、テーブルがレプリケートされるすべての に対して有効な AWS リージョン テーブルの一般設定を指定できます。その後、各レプリカの読み取り容量設定と読み取り自動スケーリング設定を上書きできます。ただし、書き込み容量はすべてのレプリカ間で同期されたままになり、すべてのリージョンに書き込みをレプリケートするのに十分な容量が確保されます。

特定のリージョンのテーブルレプリカの読み取り容量を定義するには、テーブルの の一部として以下のパラメータを設定できます`replicaSpecifications`。

- リージョン
- プロビジョニングされた読み込みキャパシティーユニット (オプション)
- 読み込み容量の Auto Scaling 設定 (オプション)

複雑な Auto Scaling 設定とテーブルレプリカのさまざまな設定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を作成する場合は、JSON ファイルからテーブルの Auto Scaling 設定とレプリカ設定をロードすると便利です。

次のコード例を使用するには、auto-scaling.zip からサンプル JSON ファイルをダウンロードし、`auto-scaling.json`と を抽出します`replication.json`。ファイルへのパスを書き留めます。

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
```

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 Auto Scaling 設定の更新 (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、 コマンドを使用します 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 CLI コマンドを使用する必要があります。これは、Amazon Keyspaces がユーザーに代わってテーブル容量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である Application Auto Scaling が、複数のをサポートしていないためです AWS リージョン。

マルチリージョンテーブルのプロビジョニングモードまたは自動スケーリング設定を更新すると、テーブルの各レプリカの読み込み容量設定と読み込み自動スケーリング設定を更新できます。

ただし、書き込み容量はすべてのレプリカ間で同期されたままになり、すべてのリージョンに書き込みをレプリケートするのに十分な容量が確保されます。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 の以下のオプションパラメータのいずれかを変更できますreplicaSpecifications。

- プロビジョニングされた読み込みキャパシティーユニット (オプション)
- 読み込み容量の Auto Scaling 設定 (オプション)

複雑な Auto Scaling 設定とテーブルレプリカのさまざまな設定を使用してマルチリージョンテーブルを更新する場合は、JSON ファイルからテーブルの Auto Scaling 設定とレプリカ設定をロードすると便利です。

次のコード例を使用するには、auto-scaling.zip からサンプル JSON ファイルをダウンロードし、auto-scaling.jsonと を抽出しますreplication.json。ファイルへのパスを書き留めます。

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
```

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オートスケーリング設定の表示 (CLI)

マルチリージョンテーブルの Auto Scaling 設定を表示するには、 get-table-auto-scaling-settingsオペレーションを使用できます。次の CLI コマンドはその一例です。

```
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
                    }
                }
            }
        }
```

```
        }
    ]
}
```

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 (CLI)

コマンドを使用して AWS CLI update-table、既存のテーブルの Auto Scaling をオフにすることができます。個々のテーブルレプリカの Auto Scaling はオフにできないことに注意してください。

次の例では、テーブルの読み込みキャパシティーの Auto Scaling がオフになっています。

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable
          \ --auto-scaling-specification
 readCapacityAutoScaling={autoScalingDisabled=true}
```

> **Note**
>
> Application Auto Scaling が使用するサービスにリンクされたロールを削除するには、すべての でアカウント内のすべてのテーブルのオートスケーリングを無効にする必要がありますAWS リージョン。

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 (CLI)

マルチリージョンテーブルの Auto Scaling をオフ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
```

> **Note**
>
> プロビジョニングされた容量を使用するマルチリージョンテーブルには、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”」を参照してください。

Amazon Keyspaces (Apache Cassandra 向け) のポイントインタイムリカバリ

ポイントインタイムリカバリ (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。

Amazon Keyspaces テーブルは、コンソール、AWS SDK と AWS Command Line Interface (AWS CLI)、または Cassandra クエリ言語 (CQL) で特定時点 (ポイントインタイム) の状態に復元できます。詳細については、「Amazon Keyspaces テーブルのポイントインタイムリカバリ」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 での point-in-time リカバリの仕組み
- Amazon Keyspaces テーブルのポイントインタイムリカバリ

Amazon Keyspaces での point-in-time リカバリの仕組み

このセクションでは、Amazon Keyspaces point-in-time リカバリ (PITR) 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

point-in-time リカバリ (PITR) の有効化

コンソールを使用して PITR を有効にすることも、プログラムで有効にすることも可能です。

コンソールを使用した PITR の有効化

新しいテーブルの PITR 設定は、[Customized settings (カスタマイズされた設定)] オプションで管理できます。デフォルトでは、コンソールを通じて作成された新しいテーブルに対して PITR が有効になっています。

既存のテーブルに対して PITR を有効にするには、次の手順を実行します。

1. AWS Management Console にサインインし、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. point-in-time 「リカバリ設定の編集」セクションで、「P oint-in-time リカバリの有効化」を選択します。

テーブルの PITR は、以下のステップでいつでも無効にできます。

1. AWS Management Console にサインインし、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. point-in-time 復旧設定の編集セクションで、P oint-in-time 復旧を有効にするチェックボックスをオフにします。

> **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。

コンソールを使用したテーブル復元方法については、「the section called “テーブルのポイントインタイムリカバリ (コンソール)”」を参照してください。

AWS CLI を使用した PITRの有効化

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。

AWS CLI を使用して新規のテーブルを作成する場合、新規のテーブルを作成する時点で PITR を明示的に有効にする必要があります。

新規テーブルの作成時に PITR を有効にするには、次の AWS CLI コマンドを例として使用します。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**
>
> point-in-time 復旧値を指定しない場合、 point-in-time復旧はデフォルトで無効になっています。

テーブルの point-in-time 復旧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'
```

AWS CLI を使用して既存のテーブルの PITR を有効にするには、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'
```

既存のテーブルで 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'
```

> **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。

CQL を使用した PITR の有効化

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。

CQL を使用して新規のテーブルを作成する場合、新規のテーブルを作成する時点で PITR を明示的に有効にする必要があります。

新規テーブルの作成時に PITR を有効にするには、次の CQL コマンドを例として使用します。

```
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**
>
> point-in-time リカバリカスタムプロパティが指定されていない場合、 point-in-timeリカバリはデフォルトで無効になっています。

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'}}
```

既存のテーブルで 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'}}
```

> **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。

CQL 言語リファレンスの詳細については、「the section called “CREATE TABLE”」と「the section called “ALTER TABLE”」を参照してください。CQL を使用したテーブル復元方法については、「the section called “CQL によるテーブルのポイントインタイムリカバリ”」を参照してください。

テーブルの復元に必要な許可

テーブルを正常に復元するには、IAM のユーザーまたはロールで次の最小限の許可を取得しておく必要があります。

- `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。

キー空間 mykeyspace 内のテーブルを復元するために必要な最小限の許可をユーザーに付与するポリシーの例を以下に示します。

```
{
   "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 point-in-time リカバリ (PITR)”」を参照してください。

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 Auto Scaling を有効にした状態でプロビジョンドキャパシティモードになっている場合、復元オペレーションはテーブルの Auto Scaling 設定も復元します。これらは、CQL の autoscaling_settingsパラメータまたは CLI 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、「」を参照してくださいthe section called “auto スケーリングによるスループット容量の管理”。

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。

- AWS Identity and Access Management (IAM) ポリシー
- Amazon CloudWatch メトリクスとアラーム
- 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。そのテーブルは、AWS 所有のキー (デフォルト) で暗号化されている場合、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。AWS 所有のキー からカスタマーマネージドキーに変更するには、復元の時点で有効でありアクセスが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 AWS 所有のキー に変更する場合は、AWS 所有のキー でテーブルを復元するためにソーステーブルのカスタマーマネージドキーに Amazon Keyspaces がアクセスできることを確認します。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「Troubleshooting key access」(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。復元操作を正常に行うには、ソーステーブルとターゲットテーブルの両方を同じ AWS リージョン に複製する必要があります。

Amazon Keyspaces は、キースペースの一部であるレプリケートされた各リージョンの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。

マルチリージョンキーのレプリケーションについては、「the section called “仕組み”」を参照してください。

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、AWS CloudTrail を使用してログに記録されるので、継続的な監視と監査が可能です。

- 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 テーブルのポイントインタイムリカバリ

Amazon Keyspaces (Apache Cassandra 向け) ポイントインタイムリカバリ (PITR) を使用すれば、過去 35 日間の任意の時点に復元することができます。このチュートリアルの第 1 部では、Amazon Keyspaces コンソール、AWS Command Line Interface (AWS CLI)、および Cassandra クエリ言語 (CQL) を使用してテーブルを特定時点の状態に復元する方法を説明します。第 2 部では、AWS CLI と CQL を使用して削除したテーブルを復元する方法を示します。

トピック

開始する前に

ユーザーによる Amazon Keyspaces テーブルの復元を可能にする適切な許可をまだ設定していない場合は、設定する必要があります。AWS Identity and Access Management (IAM) では、AWS 管理ポリシー `AmazonKeyspacesFullAccess` には Amazon Keyspaces テーブルの復元許可が含まれています。必要最小限の許可をポリシーに実装するステップの詳細については、「the section called “復元許可”」を参照してください。

テーブルのポイントインタイムリカバリ (コンソール)

Amazon Keyspaces コンソールを使用して `mytable` という既存のテーブルのポイントインタイムリカバリを実行する方法を次の例に示します。

> **Note**
>
> この手順は、ポイントインタイムリカバリを有効にしていることを前提としています。`mytable` テーブルに対して PITR を有効にするには、the section called “ コンソールを使用する場合” のステップに従います。

1. AWS Management Console にサインインし、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、[Tables] (テーブル) を選択します。
3. テーブルのリストで、[mytable] テーブルを選択します。
4. mytable テーブルの [Backups] (バックアップ) タブの [Point-in-time recovery] (ポイントインタイムリカバリ) セクションで、[Restore] (復元) を選択します。
5. 新しいテーブルの名前に **mytable_restored** と入力します。
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。復元プロセスが終了すると、mytable_restored テーブルのステータスは [Active] (アクティブ) に変わります。

   > **Important**
   >
   > 復元中は、復元を目的とした、IAM エンティティ (例: ユーザー、グループ、ロール) を付与する AWS Identity and Access Management (IAM) ポリシーの変更や削除を行わないでください。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テーブルの書き込み許可を削除したとします。この場合、基本となる RestoreTableToPointInTime オペレーションを使用しても、復元されたデータをテーブルに書き込むことはできません。
   > 復元オペレーション完了後は、アクセス権限の変更または削除のみ行うことができます。

AWS CLI によるテーブルのポイントインタイムリカバリ

AWS CLI を使用して、myTable という既存のテーブルをポイントインタイムに復元する方法を次の手順に示します。

1. 最初のステップでは、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": ""
    }
}
```

アクティブなテーブルを 1 秒間隔で earliestRestorableTimestamp 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。

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刻に戻すには、restore-timestamp 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'"
```

Important

復元中は、復元を目的とした、IAM エンティティ (例: ユーザー、グループ、ロール) を付与する AWS Identity and Access Management (IAM) ポリシーの変更や削除を行わないでください。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テーブルの書き込み許可を削除したとします。この場合、基本となる RestoreTableToPointInTime オペレーションを使用しても、復元されたデータをテーブルに書き込むことはできません。
復元オペレーション完了後は、アクセス権限の変更または削除のみ行うことができます。

CQL によるテーブルのポイントインタイムリカバリ

CQL を使用して、mytable という名前の既存のテーブルを特定の時点の状態に戻す方法を次の手順に示します。

Note

この手順は、ポイントインタイムリカバリを有効にしていることを前提としています。テーブルで PITR を有効にするには、「the section called “CQL”」のステップを実行します。

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。

   mytable の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、以下のように 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. restore_timestamp により ISO 8601 形式で指定された特定の時点の状態にテーブルを復元します。この場合は、mytable テーブルが現在の時刻に復元されます。WITH restore_timestamp = ... 句は省略できます。この句がない場合は、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;
```

特定の時点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。

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。復元プロセスが終了すると、`mytable_restored` テーブルのステータスは [Active (アクティブ)] に変わります。

> **Important**
>
> 復元中は、復元を目的とした、IAM エンティティ (例: ユーザー、グループ、ロール) を付与する AWS Identity and Access Management (IAM) ポリシーの変更や削除を行わないでください。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テーブルの書き込み許可を削除したとします。この場合、基本となる `RestoreTableToPointInTime` オペレーションを使用しても、復元されたデータをテーブルに書き込むことはできません。
> 復元オペレーション完了後は、アクセス権限の変更または削除のみ行うことができます。

AWS CLI による削除済みテーブルの復元

AWS CLI を使用して、`myTable` という既存のテーブルを削除時点の状態に復元する方法を次の手順に示します。

> **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。

1. 前のチュートリアルで作成したテーブルを削除します。

```
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"
}
```

CQL による削除済みテーブルの復元

CQL を使用して、mytable という既存のテーブルを削除時点の状態に復元する方法を次の手順に示します。

> **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。

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;
```

Amazon Keyspaces の有効期限 (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。詳細については、「the section called “使用方法”」を参照してください。

コンソールまたは Cassandra クエリ言語 (CQL) を使用して、新規および既存のテーブルのデフォルト TTL 設定の構築、変更、無効化を行うことができます。デフォルトの TTL が設定されているテーブルでは、Cassandra クエリ言語 (CQL) 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 値を行および列に適用することができます。詳細については、「the section called “有効期限 (TTL) の使い方”」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

- 仕組み: Amazon Keyspaces の有効期限 (TTL)
- 有効期限 (TTL) の使い方

仕組み: Amazon Keyspaces の有効期限 (TTL)

Amazon Keyspaces の有効期限 (TTL) は完全に管理されています。コンパクション戦略などといった低レベルのシステム設定を管理する必要はありません。Amazon Keyspaces では、データは指定した時点で期限切れになり、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、期限切れのデータが自動的に (通常は 10 日以内に) 削除されます。

期限切れのデータは削除対象としてマークされ、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。期限切れのデータが含まれてい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、期限切れのデータは、ストレージから削除されるまで、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) にカウントされ続けます。

トピック

- テーブルのデフォルト TTL 値の設定
- 行と列のカスタム TTL 値の設定
- テーブルでの TTL の有効化
- Amazon Keyspaces の 有効期限 (TTL) および AWS サービスとの統合

テーブルのデフォルト TTL 値の設定

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。既存のテーブルを編集し、テーブルに挿入される新しい行に対してデフォルト TTL 値の設定や変更を行うこともできます。テーブルのデフォルト TTL 値を変更しても、テーブル内の既存のデータの TTL 値は変更されません。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。

Amazon Keyspaces では、データが更新されるたびに新しい TTL タイムスタンプが計算されます。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。AWS Management Console または CQL を使用してテーブルのデフォルト TTL 値の設定、変更、無効化を行う方法については、「the section called “有効期限 (TTL) の使い方”」を参照してください。

行と列のカスタム TTL 値の設定

> **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 “カスタムプロパティを使用して既存のテーブルの有効期限 (TTL) を有効にする方法”」を参照してください。

テーブルのデフォルト TTL 値を上書きする場合や、個々の行の有効期限を設定する場合には、次の CQL データ操作言語 (DML) ステートメントを使用できます。

- INSERT — TTL 値セットが含まれている新しいデータ行を挿入する場合に使用します。
- UPDATE — 新しい TTL 値が含まれている既存のデータ行に変更を加える場合に使用します。

行の TTL 値の設定は、テーブルのデフォルト TTL 設定よりも優先されます。

CQL 構文と例については、「the section called “INSERT を使用して CQL でカスタムの有効期限 (TTL) 設定を編集するには”」を参照してください。

個々の列の TTL 値のオーバーライドまたは設定を行うには、次の CQL DML ステートメントを使用して、既存行内の列のサブセットの TTL 設定を更新します。

- UPDATE — データの列を更新するために使用します。

列の TTL 値の設定は、テーブルのデフォルト TTL 設定と行のカスタム TTL 設定よりも優先されます。CQL 構文と例については、「the section called “UPDATE を使用して CQL でカスタムの有効期限 (TTL) 設定を編集するには”」を参照してください。

テーブルでの TTL の有効化

CREATE TABLE または ALTER TABLE ステートメントのいずれかの default_time_to_live 値として 0 より大きい値を指定すると、TTL はテーブルに対して自動的に有効化されます。テーブルに対して default_time_to_live を指定せず、INSERT または UPDATE オペレーションを使用し

て行または列に対してカスタムの TTL 値を指定したい場合は、まず、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。

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 “によるモニタリング CloudWatch”」を参照してください。

AWS CloudFormation を使用する場合、Amazon Keyspaces テーブルを作成するときに 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。

有効期限 (TTL) の使い方

Amazon Keyspaces (Apache Cassandra 向け) コンソールまたは CQL を使用して、有効期限 (TTL、Time to Live) 設定を有効化、更新、および無効化することができます。

トピック

- デフォルトの有効期限 (TTL) 設定を有効にして新しいテーブルを作成するには (コンソール)
- 既存のテーブルでデフォルト有効期限 (TTL) 設定を更新するには (コンソール)
- 既存のテーブルのデフォルト有効期限 (TTL) 設定を無効にするには (コンソール)
- CQL を使用してデフォルトの有効期限 (TTL) 設定で新しいテーブルを作成するには

- ALTER TABLE を使用して CQL でデフォルトの有効期限 (TTL) 設定を編集するには
- カスタムプロパティを使用して新しいテーブルの有効期限 (TTL) を有効にする方法
- カスタムプロパティを使用して既存のテーブルの有効期限 (TTL) を有効にする方法
- INSERT を使用して CQL でカスタムの有効期限 (TTL) 設定を編集するには
- UPDATE を使用して CQL でカスタムの有効期限 (TTL) 設定を編集するには

デフォルトの有効期限 (TTL) 設定を有効にして新しいテーブルを作成するには (コンソール)

次の手順に従って、Amazon Keyspaces コンソールを使用し、有効期限設定が有効になっている新規テーブルを作成します。

1. AWS Management Console にサインインし、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. [Create table (テーブルを作成)] 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。

> **Note**
>
> CQL エディタのデータ操作言語 (DML) を使用して、特定の行または列に対して、そのテーブルのデフォルト TTL 設定をオーバーライドすることができます。

既存のテーブルでデフォルト有効期限 (TTL) 設定を更新するには (コンソール)

次のステップに従い、Amazon Keyspaces コンソールを使用して、既存のテーブルの有効期限 (TTL) 設定を更新します。

1. AWS Management Console にサインインし、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. [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。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。

既存のテーブルのデフォルト有効期限 (TTL) 設定を無効にするには (コンソール)

次のステップに従い、Amazon Keyspaces の AWS Management Console を使用して既存のテーブルの有効期限 (TTL) 設定を無効にします。

1. AWS Management Console にサインインし、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. [Default TTL Period] (デフォルトの TTL 期間) を選択して、値を 0 (ゼロ) に設定します。これにより、その後のデータではテーブルの TTL がデフォルトで無効になります。既存の行の TTL 値は変更されません。
5. TTL 設定が定義されたら、[Save changes] (変更を保存) を選択します。

CQL を使用してデフォルトの有効期限 (TTL) 設定で新しいテーブルを作成するには

デフォルトの TTL 値が 3,024,000 秒 (35 日間) に設定されている新しいテーブルを作成する場合は TTL を有効にします。

```
CREATE TABLE my_table (
                userid uuid,
                time timeuuid,
                subject text,
                body text,
                user inet,
                PRIMARY KEY (userid, time)
                ) WITH default_time_to_live = 3024000;
```

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。

```
describe my_table;
```

ALTER TABLE を使用して CQL でデフォルトの有効期限 (TTL) 設定を編集するには

既存のテーブルの TTL 設定を 2,592,000 秒 (30 日間) に更新します。

```
ALTER TABLE my_table WITH default_time_to_live = 2592000;
```

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。

```
describe my_table;
```

カスタムプロパティを使用して新しいテーブルの有効期限 (TTL) を有効にする方法

テーブル全体の TTL デフォルト設定を有効にしないで行と列に適用できる有効期限 (TTL) カスタム設定を有効にするには、次の CQL ステートメントを使用します。

```
CREATE TABLE my_keyspace.my_table (id int primary key) WITH CUSTOM_PROPERTIES={'ttl':
{'status': 'enabled'}};
```

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。

カスタムプロパティを使用して既存のテーブルの有効期限 (TTL) を有効にする方法

テーブル全体の TTL デフォルト設定を有効にしないで行と列に適用できる有効期限 (TTL) カスタム設定を有効にするには、次の CQL ステートメントを使用します。

```
ALTER TABLE my_table WITH CUSTOM_PROPERTIES={'ttl':{'status': 'enabled'}};
```

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。

INSERT を使用して CQL でカスタムの有効期限 (TTL) 設定を編集するには

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 を使用して CQL でカスタムの有効期限 (TTL) 設定を編集するには

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
```

Amazon Keyspaces におけるクライアントサイドのタイムスタンプの働き

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。詳細については、「the section called “使用方法”」を参照してください。

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。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がタイムスタンプを提供しない場合、Amazon Keyspaces はセルレベルの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。詳細については、「the section called “クライアント側のタイムスタンプの使用方法”」を参照してください。

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。行サイズの見積もり方法については、「the section called “使用方法”」を参照してください。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。

トピック

- Amazon Keyspaces におけるクライアント側のタイムスタンプの働き
- Amazon Keyspaces におけるクライアント側のタイムスタンプの使用

Amazon Keyspaces におけるクライアント側のタイムスタンプの働き

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。

トピック

- Amazon Keyspaces でのクライアント側のタイムスタンプの働き
- Amazon Keyspaces のクライアント側のタイムスタンプと AWS サービスとの統合

Amazon Keyspaces でのクライアント側のタイムスタンプの働き

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 “行サイズの計算”」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”」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。さらに、タイムスタンプは NULL に設定できないため、CQL ステートメントやクライアントドライバーでクライアント側のタイムスタンプが提供されない場合、Amazon Keyspaces によって生成されたタイムスタンプが自動的に追加されます。

Amazon Keyspaces のクライアント側のタイムスタンプと AWS サービスとの統合

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 “によるモニタリング CloudWatch”」を参照してください。

Amazon Keyspaces におけるクライアント側のタイムスタンプの使用

Amazon Keyspaces (Apache Cassandra 用) コンソール、Cassandra クエリ言語 (CQL)、AWS SDK、および AWS Command Line Interface (AWS CLI) で、クライアント側のタイムスタンプを有効にすることができます。このセクションでは、新規テーブルと既存テーブルでクライアント側のタイムスタンプを有効にする方法と、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の例を示します。API の詳細については、「Amazon ML API リファレンス」を参照してください。

> **Important**
>
> クライアント側のタイムスタンプは無効にできません。クライアント側のタイムスタンプを有効にできるのは 1 回限りの変更です。Amazon Keyspaces には、テーブルを削除せずに無効にするオプションはありません。

トピック

- クライアント側のタイムスタンプを有効にした新しいテーブルの作成 (コンソール)
- 既存のテーブルのクライアント側タイムスタンプを有効にする (コンソール)
- クライアント側のタイムスタンプを有効にした新しいテーブルの作成 (CQL)
- (CQL) ALTER TABLE 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
- クライアント側のタイムスタンプを有効にした新しいテーブルの作成 (CLI)
- 既存のテーブルでクライアント側のタイムスタンプを有効にする (CLI)
- データ操作言語 (DML) ステートメントにおけるクライアント側のタイムスタンプの使用

クライアント側のタイムスタンプを有効にした新しいテーブルの作成 (コンソール)

次の手順に従って、Amazon Keyspaces コンソールでクライアント側のタイムスタンプを有効にして、新規テーブルを作成します。

クライアント側のタイムスタンプ付きの新規テーブルを作成するには (コンソール)

1. AWS Management Console にサインインし、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 (テーブルを作成)] を選択します。
3. [Table details (テーブルの詳細)] セクションの [Create table (テーブルを作成)] 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。

既存のテーブルのクライアント側タイムスタンプを有効にする (コンソール)

Amazon Keyspaces AWS Management Console で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手順は、次のとおりです。

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. AWS Management Console にサインインし、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。

クライアント側のタイムスタンプを有効にした新しいテーブルの作成 (CQL)

新規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するには、次の CQL ステートメントを使用できます。

```
CREATE TABLE my_table (
   userid uuid,
   time timeuuid,
   subject text,
   body text,
   user inet,
   PRIMARY KEY (userid, time)
) WITH CUSTOM_PROPERTIES = {'client_side_timestamps': {'status': 'enabled'}};
```

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 ステートメントを使用して、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'}
```

(CQL) **ALTER TABLE** 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

既存のテーブルのクライアント側のタイムスタンプは、次の CQL ステートメントで有効にできます。

```
ALTER TABLE my_table WITH custom_properties = {'client_side_timestamps': {'status':
 'enabled'}};;
```

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'}
```

クライアント側のタイムスタンプを有効にした新しいテーブルの作成 (CLI)

新規テーブルの作成時に、クライアント側のタイムスタンプは次の CLI ステートメントで有効にできます。

```
./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}]'
```

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": ""
    }
}
```

既存のテーブルでクライアント側のタイムスタンプを有効にする (CLI)

CLI 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は、次のコードで有効にできます。

```
./aws keyspaces update-table \
--keyspace-name my_keyspace \
--table-name my_table \
--client-side-timestamps 'status=ENABLED'
```

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": 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": ""
    }
}
```

データ操作言語 (DML) ステートメントにおけるクライアント側のタイムスタンプの使用

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;
```

AWS CloudFormation を使用した Amazon Keyspaces リソースの作成

Amazon Keyspaces (Apache Cassandra 向け) は AWS CloudFormation と統合されています。これは、リソースとインフラストラクチャの作成と管理の所要時間を短縮できるように AWS リソースをモデル化して設定するためのサービスです。すべての必要な AWS リソース (キースペースやテーブルなど) を記述するテンプレートを作成すれば、AWS CloudFormation がそれらのリソースのプロビジョニングと設定をユーザーに代わって行います。

AWS CloudFormation を使用すると、テンプレートを再利用して Amazon Keyspaces リソースをいつでも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、同じリソースを複数の AWS アカウント やリージョンで何度でもプロビジョニングすることができます。

Amazon Keyspaces および 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には、AWS CloudFormation スタックにプロビジョニングしたいリソースを記述します。JSON や YAML に不慣れな方は、AWS CloudFormation Designer を使えば、AWS CloudFormation テンプレートを使いこなすことができます。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の場合、スキーマ、読み取り/書き込みモード、プロビジョンドスループット設定を指定できます。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、『AWS CloudFormation ユーザーガイド』の「Cassandra resource type referenceCassandra (リソースタイプのリファレンス)」』を参照してください。

AWS CloudFormation の詳細はこちら

AWS CloudFormation の詳細については、以下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) のモニタリング

モニタリングは、その他の AWS ソリューションの信頼性、可用性、およびパフォーマンスの維持における重要な要素です。AWS は、Amazon Keyspaces をモニタリングし、問題が発生した場合には報告を行い、必要に応じて自動アクションを実行するために以下のモニタリングツールを用意しています。

- Amazon Keyspaces には、アカウント内のすべてのテーブルについて集計したレイテンシーとエラーを示す事前設定済みのダッシュボードが AWS Management Console にあります。
- Amazon CloudWatch は、AWS のリソースおよび AWS で実行しているアプリケーションを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、「アマゾンによるアマゾンKeyspaces モニタリング CloudWatch」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、お客様独自のアプリケーション、Software as a Service (SaaS) アプリケーション、AWS のサービスからのリアルタイムデータをストリーム配信し、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

- アマゾンによるアマゾンKeyspaces モニタリング CloudWatch
- を使用した Amazon Keyspaces API コールのログ記録 AWS CloudTrail

アマゾンによるアマゾンKeyspaces モニタリング CloudWatch

Amazon を使用して Amazon Keyspaces をモニタリングできます。Amazon は未加工データを収集し CloudWatch、読み取り可能でほぼリアルタイムのメトリックスに処理します。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 CloudWatch の一般的なメトリックスを表示する設定済みのダッシュボードをすぐに使い始めるには、 AWS CloudFormation から入手できる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

- Amazon Keyspaces CloudWatch を監視するアラームの作成

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. 新しい数式の詳細セクションに、メトリクスの ID を入力し、 CloudWatch メトリクスのピリオド関数 (metric_id/ (PERIOD (metric_id)) で割ります。**`ConsumedReadCapacityUnits`** 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。

目的	関連するメトリクス
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「詳細」セクションに、指標の ID を入力し、**`ConsumedWriteCapacityUnits`** CloudWatch 指標の期間関数 (metric_id/ (PERIOD (metric_id)) で指標を割ります。 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、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。

コンソールを使用してメトリックスを表示するには CloudWatch

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. https://console.aws.amazon.com/cloudwatch/ CloudWatch でコンソールを開きます。

2. 必要に応じて、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。

AWS CLI 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 のメトリクスとディメンション

Amazon Keyspaces が Amazon CloudWatch に送信するメトリックスとディメンションは次のとおりです。

Amazon Keyspaces のメトリクス

Amazon は Amazon Keyspaces メトリクスを 1 CloudWatch 分間隔で集計します。

`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 コンソールから、またはコンソール、、AWS またはすべてのメトリクスの CloudWatch SDK を使用して取得できます。 AWS CLI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。

メトリクス	説明
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。
AccountProvisionedReadCapacityUtilization	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。

メトリクス	説明
AccountProvisionedWriteCapacityUtilization	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。
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。False と評価された条件チェックでは、行のサイズに基づいて書き込みキャパシティーユニットを消費します。詳細については、「the section called “軽量トランザクション”」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は、レプリケートされた各書き込みもこのクォータに含まれます。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が反映され、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されません。 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。
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 は通常、無効なパラメータの組み合わせ、存在しないテーブルへの更新の試み、不正なリクエスト署名など、クライアント側のエラーを示します。 UserErrors AWS リージョン 現在と現在の無効なリクエストの合計を表します 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

メトリクス	説明
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、次の表のいずれかのディメンションに沿って 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 CloudWatch を監視するアラームの作成

Amazon Keyspaces 用の Amazon CloudWatch アラームを作成して、アラームの状態が変化したときに Amazon Simple Notification Service (Amazon SNS) メッセージを送信できます。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 を使用してプロビジョニングモードで Amazon Keyspaces を使用すると、サービスはユーザーに代わって 2 CloudWatch 組のアラームを作成します。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 これらのアラームは、テーブルの実際の使用率が一定期間目標の使用率から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 CloudWatch によって作成されるアラームの詳細については、を参照してくださいthe section called “Amazon Keyspaces オートスケーリングの仕組み”。

アラームは、状態が持続的に変化した場合にのみアクションを呼び出します。 CloudWatch アラームは、単に特定の状態にあるからといってアクションを起動するわけではありません。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「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. CloudTrail captures データ定義言語 (DDL) API コールと Amazon Keyspaces のデータ操作言語 (DML) API コールのユーザー、ロール、または AWS サービスによって実行されたアクションをイベントとして記録するサービスである と統合されています。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。

証跡を作成する場合は、Amazon Keyspaces の CloudTrail イベントなど、Amazon Simple Storage Service (Amazon S3) バケットへのイベントの継続的な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、サポートされている最新のイベントを コンソールの CloudTrailイベント履歴 で表示できます。によって収集された情報を使用して CloudTrail、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。

の詳細については CloudTrail、「 AWS CloudTrail ユーザーガイド」を参照してください。

トピック

- での Amazon Keyspaces ログファイルエントリの設定 CloudTrail
- の Amazon Keyspaces データ定義言語 (DDL) 情報 CloudTrail
- の Amazon Keyspaces データ操作言語 (DML) 情報 CloudTrail
- Amazon Keyspaces ログファイルエントリの理解

での Amazon Keyspaces ログファイルエントリの設定 CloudTrail

ログインした各 Amazon Keyspaces API アクションには、CQL クエリ言語で表されるリクエストパラメータ CloudTrail 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、「イベント履歴 を含む CloudTrail 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。

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 がサポートするサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信
- 複数のアカウントからの CloudTrail ログファイルの受信

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。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) ユーザー認証情報のどちらを使用して行われたか。
- 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 要素」を参照してください。

の Amazon Keyspaces データ定義言語 (DDL) 情報 CloudTrail

CloudTrail アカウントを作成する AWS アカウント と、 で が有効になります。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 CloudTrail イベント履歴 の他の AWS サービスイベントとともに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`
DropKeyspace	DDL	`DROP KEYSPACE`	`DeleteKeyspace`
CreateTable	DDL	`CREATE TABLE`	`CreateTable`
DropTable	DDL	`DROP TABLE`	`DeleteTable`
AlterTable	DDL	`ALTER TABLE`	`UpdateTable` , `TagResource` , `UntagResource`

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 CloudTrail、まず 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 CloudTrail。 CloudTrail コンソールを使用してデータイベントタイプ Cassandra テーブルのアクティビティをログに記録するか、 CLI または API オペレーション`AWS::Cassandra::Table`を使用して `resources.type` 値を

に設定することで、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できます。 AWS CloudTrail 詳細については、「データイベントのログ記録」を参照してください。

次の表は、 CloudTrail の によってログに記録されるデータイベントを示していますCassandra table。

CloudTrail **eventName**	Statement	CQL アクション	AWS SDK アクション
Select	DML	SELECT	GetKeyspace , GetTable, ListKeyspaces , ListTables ListTagsForResource
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

Amazon Keyspaces ログファイルエントリの理解

CloudTrail ログファイルには、1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、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に関する情報が含まれます。 CloudTrail ログファイルはパブリック API コール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。

次の例は、、CreateKeyspace、、DropKeyspaceCreateTableおよび DropTableアクションを示す CloudTrail ログエントリを示しています。

```
{
  "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

AWS では、クラウドのセキュリティが最優先事項です。AWS の顧客は、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から利点を得られます。

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して説明しています。

- クラウドのセキュリティ – AWS は、AWS クラウド 内で AWS の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。また、AWS は、ユーザーが安全に使用できるサービスも提供します。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。
- クラウド内のセキュリティ - お客様の責任は、使用する AWS のサービスに応じて異なります。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 サービスの用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、責任共有モデル、、は、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、 AWS クラウド のすべて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を担います。お客様は、このインフラストラクチャでホストされてい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のよくある質問」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。

データを保護するため、AWS アカウント の認証情報を保護し、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セットアップ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を各ユーザーに付与できます。また、次の方法でデータを保護することをおすすめします。

- 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 が必須です。TLS 1.3 が推奨されます。
- AWS CloudTrail で API とユーザーアクティビティロギングをセットアップします。
- AWS のサービス内でデフォルトである、すべてのセキュリティ管理に加え、AWS の暗号化ソリューションを使用します。
- Amazon Macie などの高度なマネージド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 により AWS にアクセスするときに FIPS 140-2 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) 140-2」を参照してください。

顧客の E メールアドレスなどの機密情報や重要情報は、タグや Name フィールドなどの自由形式のフィールドに入力しないことを強くお勧めします。コンソール、API、AWS CLI、または AWS SDK で Amazon Keyspaces または他の AWS のサービス サービスを使用する場合も同様にこの指示に従ってください。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

トピック

Amazon Keyspaces の保管データ暗号化

Amazon Keyspaces (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 では、ユーザーに意識させることなくテーブルデータの暗号化と復号化が行われます。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 management service concepts (キー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型です。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。ユーザーは、カスタマー管理キーに対する完全なコントロール権限を持ちます (AWS KMS の料金が適用されます)。

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。

AWS 所有のキー のデフォルトオプションを使用した保管データ暗号化に追加料金はかかりません。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。

Amazon Keyspaces の保管データ暗号化は、AWS 中国 (北京) リージョンと AWS 中国 (寧夏) リージョンを含むすべての AWS リージョン 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。

保管データ暗号化には、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の AWS Key Management Service (AWS KMS) が統合されます。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型です。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。ユーザーは、カスタマー管理キーに対する完全なコントロール権限を持ちます (AWS KMS の料金が適用されます)。

AWS KMS キー (KMS キー)

保管データ暗号化では、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。

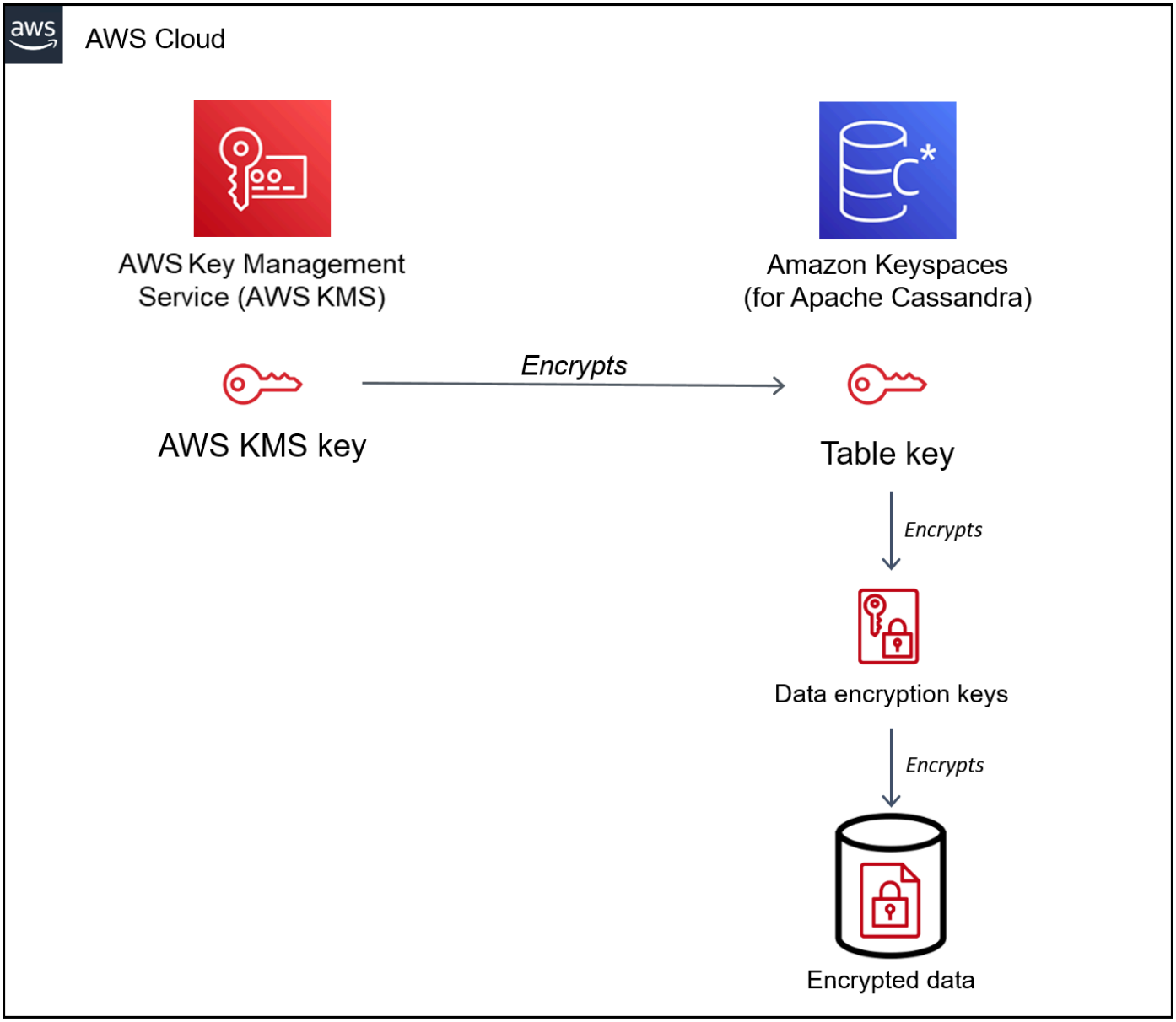
ただし、Amazon Keyspaces テーブルの暗号化には AWS アカウント のカスタマーマネージドキーを使用できます。キースペース内のテーブルごとに異なる KMS キーを選択することができます。テーブルに対して選択した KMS キーは、すべてのメタデータと復元可能なバックアップの暗号化にも使用されます。

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したときに、AWS KMS によりデータキーが生成されます。AWS KMS データキーとは、Amazon Keyspaces 内のテーブルキーを指します。

暗号化されたテーブルに最初にアクセスすると、Amazon Keyspaces から、KMS キーを使用してテーブルキーを復号するリクエストが AWS KMS に送信されます。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 により AWS KMS の外部でテーブルキーとデータ暗号化キーの生成、使用、保存が行われます。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 オペレーションごとに AWS KMS が呼び出されることのないように、Amazon Keyspaces では各接続のプレーンテキストテーブルキーがメモリにキャッシュされます。Amazon Keyspaces では、キャッシュしたテーブルキーが 5 分間非アクティブ状態であった後にリクエストを取得すると、AWS KMS に新しいリクエストを送信してテーブルキーを復号します。この呼び出しは、テーブルキーの復号を求める前回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すべてキャプチャします。

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 AWS アカウント に保存されません。これらは、複数の AWS アカウント で使用できるようにするために AWS が所有し管理している KMS キーのコレクションの一部です。AWS サービスでは、データの保護に AWS 所有のキー を使用できます。

AWS 所有のキー は表示、管理、使用することはできず、その使用を監査することもできません。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。

AWS 所有のキー のご利用に関しては、月額料金や使用料金は請求されません。また、アカウントの AWS KMS クォータにも影響しません。

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 のキーで、その作成、所有、管理をユーザーが行います。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 および Amazon CloudWatch Logs を使用して、Amazon Keyspaces がユーザーに代わって AWS KMS 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: AWS CloudTrail を使用してモニタリングを設定する”」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 KMS キーには AWS KMS 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では、処理中に AWS 所有のキー 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 を使用すると、新しいテーブルに対して AWS KMS key を指定し、Amazon Keyspaces の既存のテーブルで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: AWS KMS を使用してカスタマーマネージドキーを作成し、Amazon Keyspaces にアクセス許可を付与する

Amazon Keyspaces テーブルをカスタマーマネージドキーで保護できるようにするには、まず AWS Key Management Service (AWS KMS) でキーを作成し、次に Amazon Keyspaces に対してそのキーの使用を許可します。

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「Creating symmetric encryption KMS keys (対称暗号化 KMS キーの作成)」 のステップに従います。

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 には、特定の AWS リージョン から独立した許可が必要です。
- 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 AWS アカウント の実際のプリンシパルに置き換えます。

```
{
```

```
  "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。権限は、テーブル名と AWS アカウント ID が含まれている Amazon Keyspaces 暗号化コンテキストによる制約を受けます。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. AWS Management Console にサインインし、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、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。

   [Encryption at rest (保管データ暗号化)] セクションの [Choose an AWS KMS key (を選択)] で、オプション [Choose a different KMS key (advanced) (異なる KMS キーの選択 (高度))] を選択し、検索フィールドで、AWS KMS key を選択するか「Amazon リソースネーム (ARN)」と入力します。

   > **Note**
   >
   > 選択したキーにアクセスできない場合、または必要な許可を取得していない場合は、『AWS Key Management Service デベロッパーガイド』の「Troubleshooting key access(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か確認するには、『AWS Key Management Service デベロッパーガイド』の「Troubleshooting key access(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することができます。

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. AWS Management Console にサインインし、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。

   [Choose an AWS KMS key (を選択)] で、オプション [Choose a different KMS key (advanced) (異なる KMS キーの選択 (高度))] を選択し、検索フィールドで、AWS KMS key を選択するか「Amazon Resource Name (ARN)」(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「Troubleshooting key access(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、カスタマーマネージドキーで暗号化されているテーブルに対しては、AWS 所有のキー を選択できます。
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か確認するには、『AWS Key Management Service デベロッパーガイド』の「Troubleshooting key access(キーアクセスのトラブルシューティング)」を参照してください。

AWS 所有のキー を使用して暗号化キーをデフォルトの保管データ暗号化オプションに戻すには、以下の例に示すように、ALTER TABLE ステートメントを使用します。

```
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組み込むと、AWS KMS は暗号化コンテキストを暗号化されたデータに暗号化してバインドします。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 AWS KMS 暗号化オペレーションで同じ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。

Amazon Keyspaces は AWS KMS へのリクエストで 3 つのキーバリューペアがある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: AWS CloudTrail を使用してモニタリングを設定する

カスタマーマネージドキーを使用して Amazon Keyspaces のテーブルを保護する場合、AWS CloudTrail ログを使用して、Amazon Keyspaces がユーザーに代わって AWS KMS に送信したリクエストを追跡することができます。

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。

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。また、GenerateDataKey リクエストを、テーブルの KMS キーを指定する AWS KMS 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化するために、そのテーブルの KMS キーを指定する AWS KMS に対して Amazon Keyspaces から Decrypt リクエストが送信されます。

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している AWS アカウント のプリンシパルです。パラメータには、暗号化されたテーブルキー (暗号化テキストの blob として)、およびテーブルと 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。AWS KMS では、暗号化テキストからカスタマーマネージドキーの ID が取得されます。

```
{
    "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、オンプレミスアプリケーションから Amazon Keyspaces (Apache Cassandra 向け) への接続、および Amazon Keyspaces とその他の AWS リソース (同一 AWS リージョン 内) 間の接続に対する、Amazon Keyspaces による保安方法について説明します。

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 AWS との間には 2 つの接続オプションがあります:

- 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 「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 (for Apache Cassandra) 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 AWS がインフラストラクチャ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、「セキュリティの柱 - AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

AWS が公開している API コールを使用し、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 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、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。パスワードの作成と管理に IAM コンソール、AWS CLI、または AWS API を使用できます。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 “認証用の IAM 認証情報 AWS ”」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を可能にします。インターフェイス VPC エンドポイントには、AWS PrivateLink が搭載されています。これは、VPC 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。AWS PrivateLink では、VPC 内にプライベート IP がある Elastic Network Interface を使用して、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。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 用の

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 AWS のサービス 使用できる です。

トピック

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しておくと、管理者に適切な許可をリクエストするうえで役立ちます。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 の認証情報は、フェデレーティッド 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 AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 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、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」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 Software Development Kit (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に暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストに自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名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。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させる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「AWSでの多要素認証 (MFA) の使用」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 AWS のサービス およびリソースへの完全なアクセス権を持つ 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 内のアイデンティティです。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、ロールでは一時的な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 (ロールではなく) の作成が適している場合」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 内のアイデンティティです。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。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、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「IAM ロールの使用」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詳細については、「IAM ユーザーガイド」の「Creating a role for a third-party Identity Provider」(サードパーティーアイデンティティプロバイダー向けロールの作成) 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 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。

- 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、 (ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。クロスアカウントアクセスの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リソースアクセス」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 でアクションを実行する場合 AWS、ユーザーは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 へのリクエストリクエストリクエスト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 のサービスにアクセス許可を委任するロールの作成」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。 AWS ロールを EC2 インスタンスに割り当て、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。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」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、 AWS ID またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられている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) AWS 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 AWS JSON 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んなリソースにどんな条件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継ぐことができます。

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、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 権限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

ベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」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「マネージドポリシーとインラインポリシーの比較」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、 AWS WAF、および Amazon VPC は、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、一般的ではない追加のポリシータイプをサポートします。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のポリシーによって 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) に対する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、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。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「SCP の仕組み」を参照してください。
- 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。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を把握するには、「IAM ユーザーガイド」の「IAM AWS 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、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 AWS JSON 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んなリソースにどんな条件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権限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。

単一ステートメントに複数アクションを指定するには、次のようにカンマで区切ります:

```
"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 AWS JSON 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どのような条件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、キースペースとテーブルは、IAM 権限の Resource 要素で使用できます。

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}
```

ARN の形式の詳細については、「Amazon リソースネーム (ARNs 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:us-east-1:123456789012:/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:us-east-1:123456789012:/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": "*"
```

接続時にほとんどのドライバーでシステムキースペース/テーブルが読み込まれるため、標準ドライバーで Amazon Keyspaces にプログラムで接続するときは、プリンシパルはシステムテーブルへの SELECT アクセス権限が必要です。例えば、mykeyspace で mytable のために IAM ユーザーに SELECT 権限を与える場合、プリンシパルには mytable と system keyspace の両方を読み取るための権限が必要です。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:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/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 AWS JSON 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んなリソースにどんな条件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することで、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 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、 AWS 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 “タグの使用”」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GetFederationトークン などの AWS STS API オペレーションを呼び出します。

Amazon Keyspaces では、Github リポジトリから入手できる AWS Signature Version 4 (SigV4) 認証プラグインによる一時的な認証情報を、次の言語で使用できます。

- 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。

Amazon Keyspaces にプログラムでアクセスするための認証プラグインを実装する例と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”。

サービスリンクロール

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 “サービスリンクロールの使用”」を参照してください。

サービスロール

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

- ポリシーの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

ポリシーの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を付与するには、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して、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「AWS ジョブ機能の管理ポリシー」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、 などの特定の を介してサービスアクションが使用される場合に 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 policy elements: Condition] (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 ポリシーの検証」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 保護 API アクセスの設定」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 コンソールにアクセスするための特定のアクセス許可は必要ありません。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 – このポリシーは、Amazon Keyspaces への読み取り専用アクセスを許可します。
- AmazonKeyspacesFullAccess – このポリシーは、すべての機能へのフルアクセスを持つ Amazon Keyspaces を使用するアクセス許可を付与します。

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

以下は、Amazon Keyspaces システムテーブルへの読み取り専用 (SELECT) アクセスを許可するポリシーの例です。すべてのサンプルについて、Amazon リソースネーム (ARN) のリージョンとアカウント ID を独自のものに置き換えます。

> **Note**
>
> ほとんどのドライバーで接続時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と接続する場合に、ユーザーは少なくともシステムテーブルへの SELECT アクセス権は持っておく必要があります。

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

```
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

次のサンプルポリシーは、キースペース のユーザーテーブルmytableに読み取り専用アクセスを追加しますmykeyspace。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/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。

> **Note**
>
> システムテーブルは常に読み取り専用です。

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

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:us-east-1:111122223333:/keyspace/mykeyspace/*",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

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 ドライバーおよびデベロッパーツールを介した Cassandra クエリ言語 (CQL) API 呼び出し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:us-east-1:111122223333:/keyspace/mykeyspace/table/*",
            "arn:aws:cassandra:us-east-1:111122223333:/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:us-east-1:111122223333:/keyspace/mykeyspace/
table/*",
          "Condition":{
             "StringEquals":{
                "aws:RequestTag/Owner":"${aws:username}"
            }
         }
      }
   ]
}
```

Amazon Keyspaces の AWS 管理ポリシー

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。AWS マネージドポリシーは、多くの一般的なユースケースで権限を提供できるように設計されているため、ユーザー、グループ、ロールへの権限の割り当てを開始できます。

AWS マネージドポリシーは、ご利用の特定のユースケースに対して最小特権の権限を付与しない場合があることにご注意ください。AWS のすべてのお客様が使用できるようになるのを避けるためです。ユースケース別にカスタマー管理ポリシーを定義することで、権限を絞り込むことをお勧めします。

AWS 管理ポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されている権限を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しい AWS のサービスを起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージド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 管理ポリシー」を参照してください。

AWSマネージドポリシー: AmazonKeyspacesReadOnlyAccess_v2

AmazonKeyspacesReadOnlyAccess_v2 ポリシーは IAM ID 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。
- Application Auto Scaling — アプリケーションオートスケーリング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 — 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、各自のアカウントで作成して管理している AWS KMS キーを表示して、Amazon Keyspaces に割り当てられているキーが有効な対称暗号化キーであるかを確認できる場合に必須です。
- 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が、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することができます。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 ポリシーは IAM ID 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。
- 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 — 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、各自のアカウントで作成して管理している AWS KMS キーを表示して、Amazon Keyspaces に割り当てられているキーが有効な対称暗号化キーであるかを確認できる場合に必須です。

JSON ポリシーをフォーマットで確認するには、「AmazonKeyspacesReadOnlyAccess」を参照してください。

AWS 管理ポリシー: AmazonKeyspacesFullAccess

AmazonKeyspacesFullAccess ポリシーは IAM ID 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 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。
  - `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、あなたに代わって 自動スケーリングアクションを実行するためのサービスにリンクされたロールを作成します。
  - `Amazon Keyspaces Multi-Region Replication` — 管理者がマルチリージョンキー空間を作成すると、サービスにリンクされたロールが自動的に作成され、選択された AWS リージョン に対するデータレプリケーションがあなたに代わって行われます。

  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 “サービスリンクロールの使用”」を参照してください。
- `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、自分のアカウントで作成し、管理している AWS KMS キーを表示して、Amazon Keyspaces に割り当てられている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。

Amazon Keyspaces での AWS 管理ポリシーに関する更新

Amazon Keyspaces の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。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
AmazonKeyspacesFullAccess — 既存のポリシーに対する更新	Amazon Keyspaces では、クライアントがインターフェイス 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ための新しい読み取り専用権限を追加しました。 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 10 月 3 日
AmazonKeyspacesReadOnlyAccess_v2 – 新しいポリシー	Amazon Keyspaces では、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ための読み取り専用権	2023 年 9 月 12 日

変更	説明	日付
	限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。	
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

変更	説明	日付
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 のメトリクスとディメンション”」を参照してください。	2022 年 7 月 7 日
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 のメトリクスとディメンション”」を参照してください。	2022 年 7 月 7 日

変更	説明	日付
AmazonKeyspacesReadOnlyAccess — 既存のポリシーに対する更新	Amazon Keyspaces で、Amazon Keyspaces の保管データ暗号化が設定されている AWS KMS キーをユーザーが表示できる新しい権限を追加しました。 Amazon Keyspaces の保管データ暗号化は、保管中のデータの暗号化に使用する暗号化キーの保護および管理のために、AWS KMS に統合されます。Amazon Keyspaces に対して設定されている AWS KMS キーを表示するために、読み取り専用権限を追加しました。	2021 年 6 月 1 日
AmazonKeyspacesFullAccess — 既存のポリシーに対する更新	Amazon Keyspaces で、Amazon Keyspaces の保管データ暗号化が設定されている AWS KMS キーをユーザーが表示できる新しい権限を追加しました。 Amazon Keyspaces の保管データ暗号化は、保管中のデータの暗号化に使用する暗号化キーの保護および管理のために、AWS KMS に統合されます。Amazon Keyspaces に対して設定されている AWS KMS キーを表示するために、読み取り専用権限を追加しました。	2021 年 6 月 1 日

変更	説明	日付
Amazon Keyspaces で変更の追跡が開始されました。	Amazon Keyspaces で AWS 管理ポリシーの変更の追跡が開始されました。	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 point-in-time リカバリ (PITR) を使用してテーブルを復元できない
- iam を実行する権限がありません。PassRole
- 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。

がアクションを実行する権限がないと AWS Management Console 通知した場合は、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 point-in-time リカバリ (PITR) を使用してテーブルを復元できない

Amazon Keyspaces テーブルを point-in-time 復旧 (PITR) で復元しようとしていて、復元プロセスが開始されても正常に完了しない場合は、復元プロセスに必要なアクセス許可がすべて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 “復元許可”」を参照してください。

iam を実行する権限がありません。PassRole

`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、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

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。サインイン認証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

Amazon Keyspaces へのアクセスを他のユーザーに許可するには、アクセスを必要とする人またはアプリケーション用に 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。ユーザーまたはアプリケーションは、そのエンティティの認証情報を使用して AWSにアクセスします。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。

すぐに開始するには、『IAM ユーザーガイド』の「Creating your first IAM delegated user and group (IAM が委任した最初のユーザーおよびグループの作成)」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認証されたユーザーへのアクセスの提供 (ID フェデレーション) を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いについては、IAM ユーザーガイドの「IAM でのクロスアカウントリソースアクセス」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 向け) では 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。サービスリンクロールは Amazon Keyspaces によって事前に定義されており、ユーザーに代わってサービスにより他の AWS サービスが呼び出されるようにするために必要となるすべての権限が、サービスロールに含まれています。

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 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 向け) では 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。サービスリンクロールは Amazon Keyspaces によって事前に定義されており、ユーザーに代わってサービスにより他の AWS サービスが呼び出されるようにするために必要となるすべての権限が、サービスロールに含まれています。

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。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連動する AWS のサービス」を参照し、[Service-linked roles(サービスにリンクされたロール)] の列内で [Yes (はい)] と

表記されたサービスを確認してください。そのサービスに関するサービスリンクロールのドキュメントを表示するには、リンクが設定されている [Yes (はい)] を選択します。

Amazon Keyspaces のサービスリンクロール権限

Amazon Keyspaces は、 という名前のサービスにリンクされたロールを使用してAWSServiceRoleForApplicationAutoScaling_CassandraTable、Application Auto Scaling が CloudWatch ユーザーに代わって Amazon Keyspaces と Amazon を呼び出すことを許可します。

AWSServiceRoleForApplicationAutoScaling_CassandraTable 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。

- `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: `arn:*:cassandra:*:*:/keyspace/system/table/*` 上で `cassandra:Select`
- 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。AWS Management Console、CQL、、AWS CLIまたは AWS API を使用してテーブルで Amazon Keyspaces Auto Scaling 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。テーブルに対して Amazon Keyspaces Auto Scaling を有効にすると、Application Auto Scaling はサービスにリンクされたロールを再度作成します。

> **Important**
>
> このサービスリンクロールは、このロール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、「AWS アカウント に新しいロールが表示される」を参照してください。

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 自動アプリケーションスケーリングを有効にすると、Application Auto Scaling によってサービスリンクロールが再び作成されます。

Amazon Keyspaces のサービスリンクロールの編集

Amazon Keyspaces では、 AWSServiceRoleForApplicationAutoScaling_CassandraTable 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を作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、『IAM ユーザーガイド』の「サービスにリンクされたロールの編集」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が必要な機能またはサービスが不要になった場合には、その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、サービスリンクロールを手動で削除できるようにするには、すべての AWS リージョン のアカウント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 テーブルのオートスケーリングを無効にするには、「Amazon Keyspaces の自動スケーリング設定の変更または無効化」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、、または AWS API を使用してAWS CLI、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「Deleting a Service-Linked Role」(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。

> **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 向け) では 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。サービスリンクロールは Amazon Keyspaces によって事前に定義されており、ユーザーに代わってサービスにより他の AWS サービスが呼び出されるようにするために必要となるすべての権限が、サービスロールに含まれています。

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。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連動する AWS のサービス」を参照し、[Service-linked roles(サービスにリンクされたロール)] の列内で [Yes (はい)] と表記されたサービスを確認してください。そのサービスに関するサービスリンクロールのドキュメントを表示するには、リンクが設定されている [Yes (はい)] を選択します。

Amazon Keyspaces のサービスリンクロール権限

Amazon Keyspaces は、 という名前のサービスにリンクされたロールを使用してAWSServiceRoleForAmazonKeyspacesReplication、Amazon Keyspaces が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に書き込みをレプリケートできるようにします。

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`

という名前のロールのアクセス許可ポリシー KeyspacesReplicationServiceRolePolicy は、Amazon Keyspaces が以下のアクションを実行することを許可します。

- 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`

Amazon Keyspaces のサービスにリンクされたロールは、ポリシーで指定された Amazon リソースネーム (ARN) の「Action:」アクセス許可 AWSServiceRoleForAmazonKeyspacesReplication を提供しますが、Amazon Keyspaces はアカウントの ARN を提供します。

ユーザー、グループ、ロールなどがサービスにリンクされたロールを作成、編集、削除できるようにするには、アクセス権限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「サービスリンクロールのアクセス許可」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 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を作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、『IAM ユーザーガイド』の「サービスにリンクされたロールの編集」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が必要な機能またはサービスが不要になった場合には、その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、サービにリンクされたロールを手動で削除できるようにするには、まず、すべての AWS リージョン のアカウントのすべてのマルチリージョンキースペースを削除にする必要があります。

サービスリンクロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除するには、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。

で使用されている Amazon Keyspaces リソースを削除するには AWSServiceRoleForAmazonKeyspacesReplication（ コンソール ）

1. AWS Management Console にサインインし、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
- AWS CLI の delete-keyspace操作。
- Amazon Keyspaces API の DeleteKeyspaceオペレーション。

サービスリンクロールの手動による削除

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
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 (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、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照

し、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「コンプライアンスプログラム」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、「 でのレポートのダウンロード AWS Artifact」の」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 を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、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に重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- 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。このガイドでは、ガイダンスを保護し AWS のサービス 、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にマッピングするためのベストプラクティスをまとめています。
- 「 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、リソース設定が社内プラクティス、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リストについては、「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 – これにより AWS アカウント、疑わしい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監視することで、、、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 は、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への対応に役立ちます。
- 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。AWS リージョン には、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立・隔離されたアベイラビリティーゾーンがあります。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に比べて、可用性、耐障害性、および拡張性に優れています。

Amazon Keyspaces では、耐久性と高可用性を確保するために、同一 AWS リージョン 内の複数の AWS アベイラビリティーゾーンにおいて、データが自動的に 3 回レプリケートされます。

AWS リージョン と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 では、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に対応できるように複数の機能を提供しています。

マルチリージョンレプリケーション

データまたはアプリケーションを遠方でレプリケートする必要がある場合は、Amazon Keyspaces でマルチリージョンレプリケーションを使用します。Amazon Keyspaces ステーブルは、選択した最大 6 つ AWS リージョン の異なるテーブルに複製できます。詳細については、「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として、Amazon Keyspaces (Apache Cassandra 用) 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 AWS がインフラストラクチャ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、「セキュリティの柱 - AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

AWS が公開している API コールを使用し、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 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、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。パスワードの作成と管理に IAM コンソール、AWS CLI、または AWS API を使用できます。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 “認証用の IAM 認証情報 AWS ”」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 を使用しています。これは、Amazon VPC で Elastic Network Interface とプライベート IP を使用して AWS のサービス間のプライベート通信を可能にする AWS のテクノロジーです。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 AWS とサービス間のプライベート通信を可能にするサービスによって駆動されます。 AWS

AWS PrivateLink これを可能にするため、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、ネットワークトラフィックが Amazon ネットワークから流出することはありません。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 point-in-time 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 step-by-step への接続を設定するチュートリアルについては、を参照してください。the section called “VPC エンドポイントとの接続”VPC AWS アカウント 内の異なる Amazon Keyspaces リソースのクロスアカウントアクセスをアプリケーションから分離して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”

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場所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 point-in-time 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条件キーを使用すると、 AWS いずれかの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送信したときにアクセスできます。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。

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) やデータベースへのパッチ適用、ファイアウォール設定、災害対策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に相当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顧客の環境に必ずしも適切または十分でない可能性があるので、処方箋ではなく、あくまで有用な検討事項とお考えください。

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

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。

Amazon Keyspaces のデフォルトでは、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。

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、有効な AWS 認証情報が AWS API リクエストに含まれている必要があります。ア

プリケーションまたは EC2 インスタンスに AWS 認証情報を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、次の操作を実行できます。

- 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。

AWS CloudTrail 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する

保管データ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、このキーの使用状況が AWS CloudTrail に記録されます。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を追跡し、オペレーション上の問題をトラブルシューティングするサポートになります。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、統合された 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 SDK、および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って、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 アドレスなどについての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 呼び出しの口グ記録」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 オペレーションのログ記録) 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、「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 のイベントの継続記録のために証跡を作成します。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 ログで収集したイベントデータをより詳細に分析し、それに基づく対応するためにその他の AWS の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

AWS のリソースにメタデータをタグ付け形式で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 (Apache Cassandra 向け) エンドポイントに接続した後、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。

トピック

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

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。

データ型	説明
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
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 は、デフォルトで最大5レベルのフリーズコレクション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。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タイプ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
```

静的列の使用の詳細については、「」を参照してくださいthe section called “静的列”。

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。
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

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
			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。
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。

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
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。
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。
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。

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
`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
- CREATE TABLE
- ALTER TABLE
- RESTORE TABLE
- DROP TABLE

トピック

- Keyspaces
- テーブル

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

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 つから最大 6 つの を指定します 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空間を作成するには、NetworkTopologyStrategy を指定して、少なくとも 2 つ、最大で 6 つまで含める必要があります。 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 を使用して、キー空間でのタグの追加や削除を行います。

[Syntax (構文)]

```
alter_keyspace_statement ::=
    ALTER KEYSPACE keyspace_name
    [[ADD | DROP] TAGS
```

Where:

- *keyspace_name* は変更するキー空間の名前です。

- 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。

例

キー空間を次のように変更します。

```
ALTER KEYSPACE "myGSGKeyspace" ADD TAGS {'key1':'val1', 'key2':'val2'};
```

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 "myGSGKeyspace";
```

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”」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 を設定できますAUTOSCALING_SETTINGS。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、「」を参照してくださいthe section called “CQL の使用”。

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 ]
                              | options
                              | 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。
- *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*— この種の列 collection (LIST、SET、またはMAP など) 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。
  - *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。

- 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 point-in-time 復元を有効または無効にするかどうかを指定します。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: に固有のマルチリージョンテーブルの設定を指定します AWS リージョン。マルチリージョンテーブルの場合、テーブルの読み込み容量は ごとに異なる方法で設定できます AWS リージョン。これを行うには、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 “プロビジョンドキャパシティモードと Auto Scaling (CQL) を使用した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 – 次の設定を持つテーブルレプリカ 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、プロビジョニングモードのテーブルに対する以下のオプション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 “CQL を使用して自動スケーリングで新しい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 – 書き込み容量の自動スケーリングを有効にするには、値を に設定しますfalse。デフォルトは true です。(オプション)
    - minimum_units – テーブルが常にサポートする準備が整っている必要がある書き込みスループットの最小レベル。値は、1 からアカウントの 1 秒あたりの最大スループットクォータ (デフォルトでは 40,000) の間でなければなりません。

- maximum_units – テーブルが常にサポートする準備が整っている必要がある書き込みスループットの最大レベル。値は、1 からアカウントの 1 秒あたりの最大スループットクォータ (デフォルトでは 40,000) の間でなければなりません。
- scaling_policy — Amazon Keyspaces はターゲット追跡ポリシーをサポートしています。自動スケーリングターゲットは、テーブルのプロビジョニングされた書き込み容量です。
  - target_tracking_scaling_policy_configuration – ターゲット追跡ポリシーを定義するには、ターゲット値を定義する必要があります。ターゲットの追跡期間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。 Auto Scaling
    - target_value – テーブルのターゲット使用率。Amazon Keyspaces の自動スケーリングにより、プロビジョンドキャパシティに対する消費キャパシティの比率がこの値またはその近くにとどまるようになります。target_value をパーセンテージとして定義します。20 ~ 90 の倍数。(必須)
    - scale_in_cooldown – 別のスケールインアクティビティが開始される前にテーブルが安定する、スケーリングアクティビティ間の秒単位のクールダウン期間。値が指定されていない場合、デフォルトは 0 です。(オプション)
    - scale_out_cooldown – 別のスケールアウトアクティビティが開始される前にテーブルが安定する、スケーリングアクティビティ間の秒単位のクールダウン期間。値が指定されていない場合、デフォルトは 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booleanを指定する 。このパラメータはデフォルトで無効になっています。をオンにするにはscale-in、boolean値を に設定しますFALSE。つまり、ユーザーに代わってテーブルの容量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 - 読み取り容量の自動スケーリングを有効にするには、値を に設定しますfalse。デフォルトは true です。(オプション)
  - minimum_units – テーブルが常にサポートする準備が整っている必要があるスループットの最小レベル。値は、1 からアカウントの 1 秒あたりの最大スループットクォータ (デフォルトでは 40,000) の間でなければなりません。
  - maximum_units – テーブルが常にサポートする準備が整っている必要があるスループットの最大レベル。値は、1 からアカウントの 1 秒あたりの最大スループットクォータ (デフォルトでは 40,000) の間でなければなりません。

- scaling_policy — Amazon Keyspaces はターゲット追跡ポリシーをサポートしています。自動スケーリングターゲットは、テーブルのプロビジョニングされた読み込み容量です。
  - target_tracking_scaling_policy_configuration – ターゲット追跡ポリシーを定義するには、ターゲット値を定義する必要があります。ターゲットの追跡期間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。 Auto Scaling
    - target_value – テーブルのターゲット使用率。Amazon Keyspaces の自動スケーリングにより、プロビジョンドキャパシティに対する消費キャパシティの比率がこの値またはその近くにとどまるようになります。target_value をパーセンテージとして定義します。20 ~ 90 の倍数。(必須)
    - scale_in_cooldown – 別のスケールインアクティビティが開始される前にテーブルが安定するスケーリングアクティビティ間の秒単位のクールダウン期間。値が指定されていない場合、デフォルトは 0 です。(オプション)
    - scale_out_cooldown – 別のスケールアウトアクティビティが開始される前にテーブルを安定させる、スケーリングアクティビティ間の秒単位のクールダウン期間。値が指定されていない場合、デフォルトは 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booleanを指定する 。このパラメータはデフォルトで無効になっています。をオンにするにはscale-in、boolean値を に設定しますFALSE。つまり、ユーザーに代わってテーブルの容量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、テーブルの読み込み容量は ごとに異なる方法で設定できます AWS リージョン。これを行うには、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 “プロビジョンドキャパシティモードと Auto Scaling (CQL) を使用した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 – 次の設定を持つ AWS リージョン テーブルレプリカの 。
    - provisioned_read_capacity_autoscaling_update
      - autoscaling_disabled - テーブルの読み込み容量の自動スケーリングを有効にするには、値を に設定しますfalse。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、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` – テーブルが常にサポートする準備が整っている必要がある読み取りスループットの最小レベル。値は、1 からアカウントの 1 秒あたりの最大スループットクォータ (デフォルトでは 40,000) の間でなければなりません。
- `maximum_units` – テーブルが常にサポートする準備が整っている必要がある読み取りスループットの最大レベル。値は、1 からアカウントの 1 秒あたりの最大スループットクォータ (デフォルトでは 40,000) の間でなければなりません。
- `scaling_policy` — Amazon Keyspaces はターゲット追跡ポリシーをサポートしています。自動スケーリングターゲットは、テーブルのプロビジョニングされた読み込み容量です。
  - `target_tracking_scaling_policy_configuration` – ターゲット追跡ポリシーを定義するには、ターゲット値を定義する必要があります。ターゲットの追跡期間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。 Auto Scaling
    - `target_value` – テーブルのターゲット使用率。Amazon Keyspaces の自動スケーリングにより、消費された読み込み容量とプロビジョニングされた読み込み容量の比率がこの値またはその近くにとどまります。`target_value` をパーセンテージとして定義します。20 ~ 90 の倍数。(必須)
    - `scale_in_cooldown` – 別のスケールインアクティビティが開始される前にテーブルが安定するスケーリングアクティビティ間の秒単位のクールダウン期間。値が指定されていない場合、デフォルトは 0 です。(オプション)
    - `scale_out_cooldown` – 別のスケールアウトアクティビティが開始される前にテーブルを安定させる、スケーリングアクティビティ間の秒単位のクールダウン期間。値が指定されていない場合、デフォルトは 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`boolean`を指定する 。このパラメータはデフォルトで無効になっています。をオンにするには`scale-in`、`boolean`値を に設定しますFALSE。つまり、ユーザーに代わってテーブルの読み取り容量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”」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));
```

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。
  - *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。
    - 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 point-in-time 復元を有効または無効にするかどうかを指定します。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 AWS リージョン 特定の設定を指定します。マルチリージョンテーブルの場合、テーブルの読み込み容量は ごとに異なる方法で設定できます AWS リージョン。これを行うには、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 (CQL) 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の更新”」を参照してください。
  - region – 次の設定を持つ AWS リージョン テーブルレプリカの 。
    - 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、プロビジョニングされたテーブルのオプションの自動スケーリング設定が含まれています。構文と詳細な説明については、「」を参照してくださいthe section called “CREATE TABLE”。例については、「the section called “CQL を使用して既存のテーブルで自動スケーリングを有効にする”」を参照してください。

- 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>>>);
```

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 point-in-time 復元を有効にするには、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 point-in-time リカバリを有効にする必要があります。詳細については、「ポイントインタイムリカバリ」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 がソーステーブルの 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 point-in-time 復元を有効または無効にするかどうかを指定します。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 AWS リージョン 特定の設定を指定します。マルチリージョンテーブルの場合、テーブルの読み込み容量は ごとに異なる方法で設定できます AWS リージョン。これを行うには、次のパラメータを設定します。
  - `region` – 次の設定を持つ AWS リージョン テーブルレプリカの 。
    - `read_capacity_units`

- *`AUTOSCALING_SETTINGS`* には、プロビジョニングされたテーブルのオプションの自動スケーリング設定が含まれています。構文と説明の詳細については、「」を参照してくださいthe section called “CREATE TABLE”。
- *`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 "myGSGKeyspace".employees_tbl;
```

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

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 Statement” 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 “結果の順序付け”」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、「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。

機能	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。

機能	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

機能	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

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 Support。

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 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
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

クォータ	説明	Amazon Keyspaces デフォルト
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 “行サイズの計算”」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
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

クォータ	説明	Amazon Keyspaces デフォルト
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 AWS Support。	5
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
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総読み込みキャパシティーユニット (RCUsの最大数。これは、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。	80,000
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総書き込みキャパシティユニット (WCU) の最大数。これは、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。	80,000

クォータ	説明	Amazon Keyspaces デフォルト
アカウントあたり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	リージョンあたりのアカウントのスケーラブルターゲットの最大数。Amazon Keyspaces テーブルは、読み込みキャパシティーに対して Auto Scaling 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、書き込みキャパシティーに対して Auto Scaling 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、Amazon Keyspaces のスケーラブルターゲットを選択することで、Application Auto Scaling の Service Quotas コンソールで調整できます。Auto Scaling	1,500
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

クォータ	説明	Amazon Keyspaces デフォルト
P oint-in-time Recovery (PITR) を使用した最大同時テーブル復元数	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
point-in-time 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

コンソールReadCapacityUnitsまたは ALTER TABLEステートメントを使用してWriteCapacityUnits、必要な回数だけ または を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。

プロビジョニングされた容量を追加する場合、アカウントごと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された容量を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。サービスクォータの引き上げをリクエストするには、「」を参照してくださいAWS Support。

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。

- ドキュメントの最新更新日:2024 年 2 月 7 日

変更	説明	日付
Amazon エラスティックKubernetesサービスからアマゾンKeyspaces にConnect	これで、 step-by-step 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用の Amazon Keyspaces auto スケーリング API	Amazon Keyspaces はCQL、プロビジョニングされたキャパシティモードでauto AWS スケーリングを設定するためのAPIサポートを提供するようになりました。	2024 年 1 月 23 日
Amazon Keyspaces のプロビジョニングされたテーブルのマルチリージョンレプリケーションサポート	Amazon Keyspaces は、マルチリージョンテーブルのプロビジョニングされたキャパシティモードをサポートするようになりました。	2024 年 1 月 23 日
ログに含まれる Amazon Keyspaces DML アクティビティ CloudTrail	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 は、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 のマルチリージョンレプリケーションによって、耐障害性、安定性、耐障害性が向上し、グローバルに分散したアプリケーションを維持しやすくなりました。	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	step-by-step このチュートリアルでは、VPC から Amazon Keyspaces に接続する方法を学びます。	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 が利用可能になりました AWS GovCloud (US)	Amazon Keyspaces がで利用できるようになり、AWS GovCloud (US) Region FedRAMP の高いコンプライアンスの対象となっています。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 Keyspaces テーブルストレージコストを Amazon でモニタリングしましょう CloudWatch	Amazon Keyspaces は、`BillableTableSizeInBytes` CloudWatch メトリックスを使用してテーブルストレージコストを経時的に監視および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。	バルクローダー (DSBulk) を使用して Apache Cassandra から Amazon Keyspaces tep-by-step にデータを移行するためのチュートリアル。 DataStax	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、カスタマーマネージドキーをサポートするようにしました。 AWS KMS	Amazon Keyspaces のIAM管理ポリシーには、 AWS KMS に保存されている利用可能なカスタマー管理キーを一覧表示および表示する権限が含まれるようになりました。 AWS KMS	2021 年 6 月 1 日
Amazon Keyspaces AWS KMS はカスタマーマネージドキーをサポートします。	Amazon Keyspaces を使用すると、 AWS KMS AWS KMS 保管中の暗号化のために保存されている顧客管理キーを管理できます。	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 point-in-time リカバリ (PITR)	Amazon Keyspaces point-in-time にリカバリ (PITR) 機能が加わりました。これにより、テーブルデータの継続的なバックアップが可能になり、テーブルが誤って書き込みまたは削除されるのを防ぐことができます。	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) を使用して Amazon Keyspaces にアクセス許可を管理し、セキュリティポリシーを実装する方法、および DataStax Java Driver for Cassandra の認証プラグインを使用して IAM ロールとフェデレーテッドアイデンティティを使用して Amazon Keyspaces にプログラムでアクセスする方法についての情報を追加しました。	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。