aws

开发人员指南

Amazon Keyspaces（Apache Cassandra 兼容）

Amazon Keyspaces（Apache Cassandra 兼容）：开发人员指南

Table of Contents

Amazon Keyspaces（Apache Cassandra 兼容）

Amazon Keyspaces（Apache Cassandra 兼容）是一种可扩展、高可用、托管式的 Apache Cassandra 兼容数据库服务。使用 Amazon Keyspaces，您无需预置、修补或管理服务器，也不需要安装、维护或操作软件。

Amazon Keyspaces 是一项无服务器服务，因此您只需为实际使用的资源付费，并且该服务会根据应用程序流量自动扩展和缩减表。您可以构建每秒可处理数千个请求且吞吐量和存储空间几乎无限的应用程序。

> **Note**
>
> Apache Cassandra 是一个开源宽列数据存储，设计为处理海量数据。有关更多信息，请参阅 Apache Cassandra。

利用 Amazon Keyspaces，可以在 AWS Cloud中轻松迁移、运行和扩展 Cassandra 工作负载。只需在 AWS 管理控制台上单击几下或几行代码，您就可以在 Amazon Keyspaces 中创建密钥空间和表，而无需部署任何基础设施或安装软件。

借助 Amazon Keyspaces，您可以使用与当今相同的 Cassandra 应用程序代码和开发者工具 AWS 来运行现有的 Cassandra 工作负载。

有关可用 AWS 区域 和终端节点的列表，请参阅 Amazon Keyspaces 的服务终端节点。

我们建议您首先阅读以下部分：

主题

- Amazon Keyspaces：工作原理
- Amazon Keyspaces 使用案例
- 什么是 Cassandra 查询语言 (CQL)？

Amazon Keyspaces：工作原理

Amazon Keyspaces 消除了 Cassandra 的管理开销。要了解原因，不妨先从 Cassandra 架构开始，然后将其与 Amazon Keyspaces 进行比较。

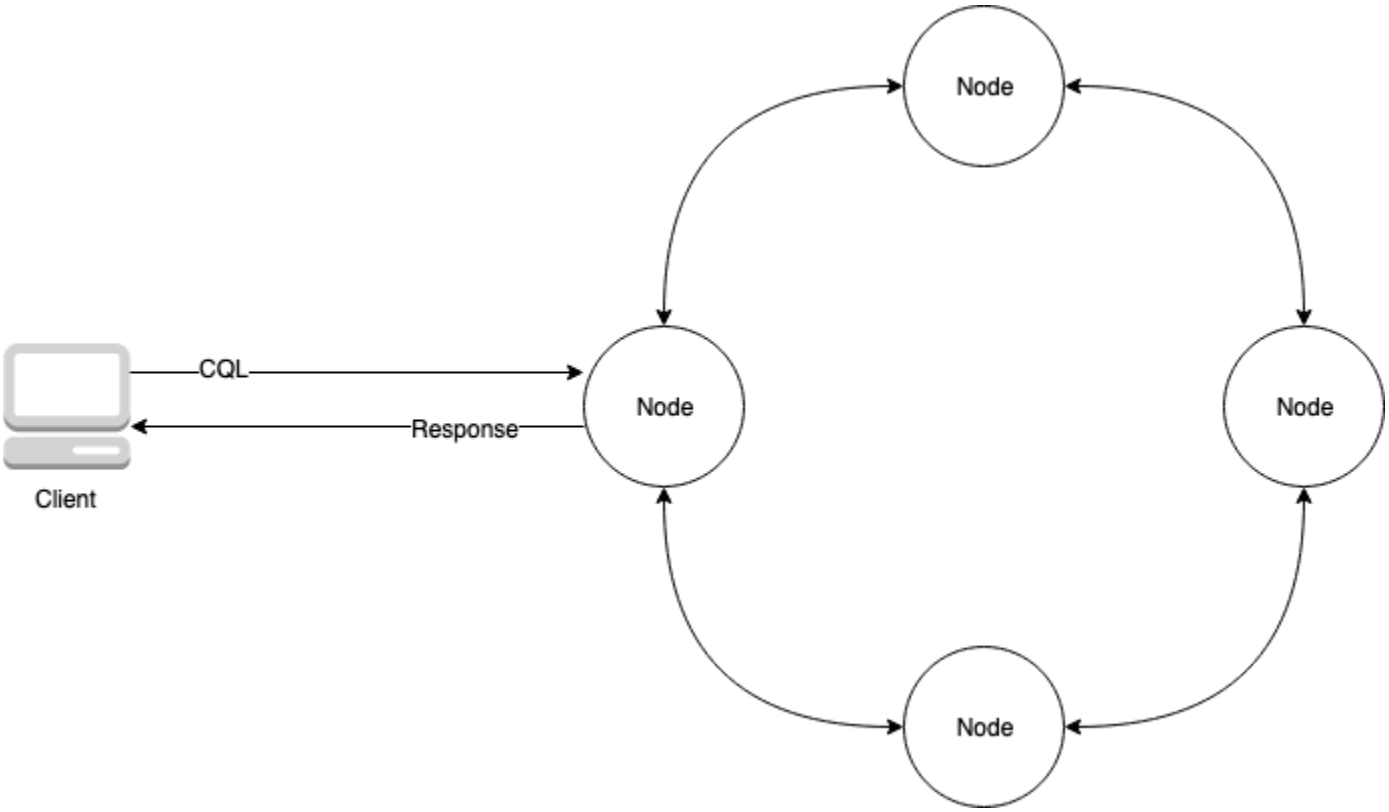
主题

架构简析：Apache Cassandra 与 Amazon Keyspaces

传统的 Apache Cassandra 部署在由一个或多个节点组成的集群中。您负责管理各个节点，并随着集群的扩展添加和删除节点。

客户端程序通过连接到其中一个节点并发出 Cassandra 查询语言 (CQL) 语句，来访问 Cassandra。CQL 类似于 SQL，后者是关系数据库中常用的语言。虽然 Cassandra 并不是关系数据库，不过 CQL 提供了一个熟悉的界面，可用于查询和操作 Cassandra 中的数据。

下图显示了一个由四个节点构成的简单 Apache Cassandra 集群。



生产 Cassandra 部署可能由数百个节点构成，这些节点在一个或多个物理数据中心内的数百台物理计算机上运行。这会给应用程序开发人员带来运营负担，这些人员除了安装、维护和操作软件之外，还需要预置、修补和管理服务器。

使用 Amazon Keyspaces（Apache Cassandra 兼容），您无需预置、修补或管理服务器，因此可以专注于构建更好的应用程序。Amazon Keyspaces 提供两种读取和写入吞吐能力：按需和预置。您可以根据工作负载的可预测性和可变性选择表的吞吐容量模式，以优化读取和写入价格。

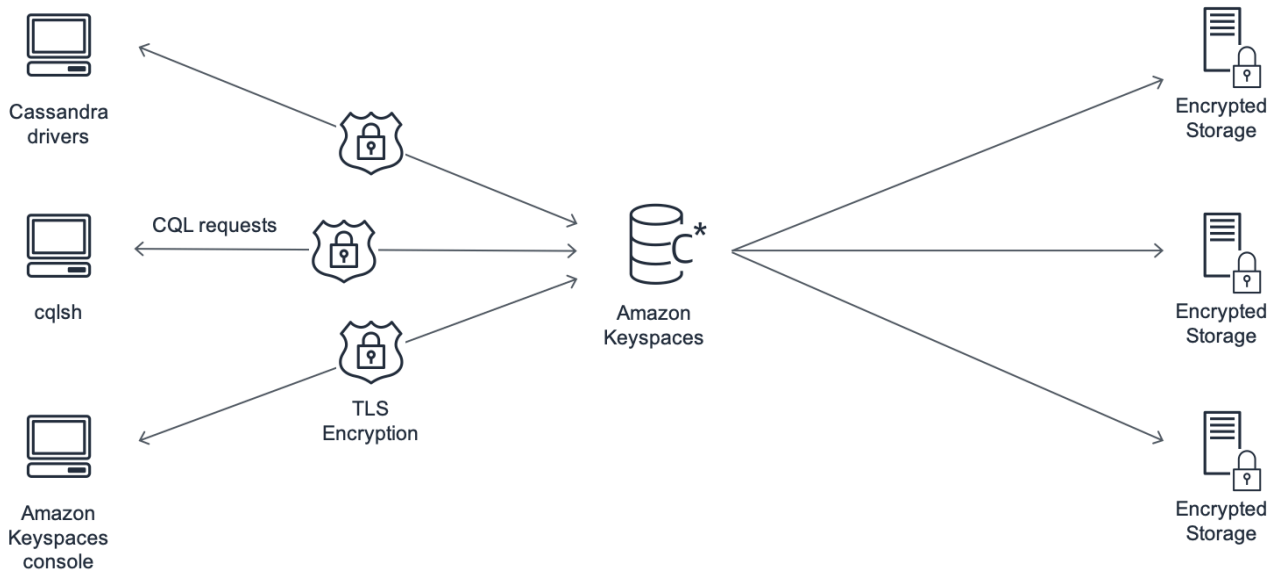
借助按需模式，您只需为您的应用程序实际执行的读取和写入付费。您无需事先指定表的吞吐能力。Amazon Keyspaces 几乎能在应用流量上升或下降时立即满足您的需求，因此对于流量不可预测的应用程序来说，它是一个不错的选择。

如果您的应用程序流量可预测，并且可以提前预测表的容量需求，那么预置容量模式可以帮助您优化吞吐量的价格。使用预置容量模式时，请指定您的应用程序预计每秒需要执行的读取和写入次数。您可以通过启用自动扩展来自动增加和减少表的预置容量。

在以下情况下，您可以每天更改一次表的容量模式：您需要了解有关工作负载的流量模式的更多信息，或如果您预计流量会突增（例如，您预计会发生导致大量表流量的重大事件）。有关预置读取和写入容量的更多信息，请参阅the section called “读/写容量模式”。

Amazon Keyspaces（Apache Cassandra 兼容）将数据的三个副本存储在多个可用区中，以实现持久性和高可用性。此外，您可以从专为满足大多数安全敏感型组织的要求而打造的数据中心和网络架构中受益。在您创建新的 Amazon Keyspaces 表时，静态加密将自动启用，并且所有客户端连接都需要传输层安全性协议 (TLS)。其他 AWS 安全功能包括监控AWS Identity and Access Management、和虚拟私有云 (VPC) 端点。有关所有可用安全功能的概述，请参阅安全性。

下图显示了 Amazon Keyspaces 的架构。



客户端程序通过连接到预先确定的端点（主机名和端口号）并发出 CQL 语句来访问 Amazon Keyspaces。有关可用端点的列表，请参阅the section called “服务端点”。

Cassandra 数据模型

业务案例数据的建模方式对于从 Amazon Keyspaces 实现最佳性能至关重要。较差的数据模型会显著地降低性能。

尽管 CQL 与 SQL 类似，但 Cassandra 后端与关系数据库后端存在明显的不同，必须采取不同的方法。下面是需要考虑的一些更重要的问题：

存储

可以在表中直观地显示您的 Cassandra 数据，其中每行表示一条记录，每列表示该记录中的一个字段。

表设计：查询优先

CQL 中没有 `JOIN`。因此，您应该根据数据的性质，以及需要如何访问数据来满足业务使用案例的要求，设计您的表。这可能会导致重复数据的逆规范化。您应该专门为特定的访问模式设计每个表。

分区

您的数据存储在磁盘分区中。存储数据的分区数以及数据在分区之间的分布方式由您的分区键 决定。分区键的定义方式会对查询的性能产生重大影响。有关最佳实践，请参阅the section called “分区键设计”。

主键

在 Cassandra 中，数据以键/值对的形式存储。为此，每个 Cassandra 表必须有一个主键，这是表中每一行的键。主键是一个必需的分区键与多个可选的聚类列的组合。组成主键的数据在表中的所有记录之间必须是唯一的。

- 分区键 - 主键的分区键部分是必需的，可确定数据存储在集群的哪个分区中。分区键可以是单个列，也可以是由两列或多个列组成的复合值。如果单列分区键会导致单个分区，或者导致极少数分区具有大部分数据，从而承担大部分磁盘 I/O 操作，则应使用复合分区键。
- 聚类列 - 主键的可选聚类列部分决定如何在每个分区中聚类和排序数据。如果您在主键中包含聚类列，聚类列可以有一个或多个列。如果聚类列中有多个列，则排序顺序由各列在聚类列中从左到右的列出顺序决定。

有关 NoSQL 设计和 Amazon Keyspaces 的更多信息，请参阅。the section called “NoSQL 设计”有关 Amazon Keyspaces 和数据建模的更多信息，请参阅。the section called “数据建模”

通过应用程序访问 Amazon Keyspaces

Amazon Keyspaces（Apache Cassandra 兼容）实施 Apache Cassandra 查询语言 (CQL) API，因此您可以使用您已在使用的 CQL 和 Cassandra 驱动程序。要更新您的应用程序，只需更新您的 Cassandra 驱动程序或 `cqlsh` 配置以指向 Amazon Keyspaces 服务端点即可。有关所需凭证的更多信息，请参阅the section called “用于 AWS 身份验证的 IAM 证书”。

> **Note**
>
> 为了帮助您入门，您可以在上的 Amazon Keyspaces end-to-end 代码示例存储库中找到使用各种 Cassandra 客户端驱动程序连接亚马逊密钥空间的代码示例。GitHub

考虑以下 Python 程序，该程序连接到 Cassandra 集群并查询表。

```
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)
```

要对 Amazon Keyspaces 运行同一个程序，您需要：

- 添加集群终端节点和端口：例如，可以将主机替换为服务终端节点，例如端口号为 9142 的 `cassandra.us-east-2.amazonaws.com`。
- 添加 TLS/SSL 配置：有关使用 Cassandra 客户端 Python 驱动程序添加 TLS/SSL 配置以连接 Amazon Keyspaces 的更多信息，请参阅使用 Cassandra Python 客户端驱动程序以编程方式访问 Amazon Keyspaces。

Amazon Keyspaces 使用案例

下面列举您可以使用 Amazon Keyspaces 的一些方法：

- 构建需要低延迟的应用程序 — 为需要延 single-digit-millisecond迟的应用程序高速处理数据，例如工业设备维护、交易监控、车队管理和路线优化。
- 使用开源技术构建应用程序 — 使用适用于各种编程语言的开源 Cassandra API 和驱动程序构建应用程序，例如 Java、Python、Ruby、Microsoft .NET、Node.js、PHP、C++、Perl 和 Go。 AWS 有关代码示例，请参阅 库和工具。
- 将您的 Cassandra 工作负载移动到云 – 自行管理 Cassandra 表非常耗时且成本高昂。借助 Amazon Keyspaces，您 AWS Cloud 无需管理基础设施即可在中设置、保护和扩展 Cassandra 表。有关更多信息，请参阅 无服务器资源管理。

什么是 Cassandra 查询语言 (CQL)？

Cassandra 查询语言 (CQL) 是与 Apache Cassandra 通信的主要语言。Amazon Keyspaces（Apache Cassandra 兼容）与 CQL 3.x API 兼容（向后兼容 2.x 版）。

要运行 CQL 查询，可以执行以下操作之一：

- 在 AWS Management Console上使用 CQL 编辑器。
- 使用 AWS CloudShell 和 cql sh-expansion。
- 在 `cqlsh` 客户端上运行查询。
- 使用 Apache 2.0 许可的 Cassandra 客户端驱动程序通过编程方式运行查询。

此外，您还可以使用 AWS 软件开发工具包访问 Amazon Keyspaces 和。 AWS Command Line Interface

有关使用这些方法访问 Amazon Keyspaces 的更多信息，请参阅访问 Amazon Keyspaces（Apache Cassandra 兼容）。

有关 CQL 的更多信息，请参阅Amazon Keyspaces（Apache Cassandra 兼容）的 CQL 语言参考。

Amazon Keyspaces（Apache Cassandra 兼容）与 Apache Cassandra 相比如何？

要建立与 Amazon Keyspaces 的连接，您可以使用公共AWS 服务终端节点，也可以使用亚马逊虚拟私有云中的接口 VPC 终端节点 (AWS PrivateLink) 使用私有终端节点。根据使用的终端节点，Amazon Keyspaces 可以通过以下方式之一显示给客户端。

AWS 服务端点连接

这是通过任何公共端点建立的连接。在本例中，Amazon Keyspaces 在客户端上显示为九节点 Apache Cassandra 3.11.2 集群。

接口 VPC 终端节点连接

这是使用接口 VPC 终端节点建立的私有连接。在本例中，Amazon Keyspaces 在客户端上显示为三节点 Apache Cassandra 3.11.2 集群。

无论连接类型和客户端可见的节点数量如何，Amazon Keyspaces 几乎可以提供无限的吞吐量和存储空间。为此，Amazon Keyspaces 会将节点映射到负载均衡器，这些负载均衡器会将您的查询路由到众多底层存储分区之一。有关连接的更多信息，请参阅the section called “它们如何运作”。

Amazon Keyspaces 将数据存储在分区中。分区是为表分配的存储空间，由固态硬盘 (SSD) 支持。AWS 区域 为了实现持久性和高可用性，Amazon Keyspaces 会自动在中的多个可用区域中复制您的数据。随着吞吐量或存储需求的增长，Amazon Keyspaces 会为您处理分区管理，并自动配置所需的额外分区。有关更多信息，请参阅 the section called “存储”。

Amazon Keyspaces 支持所有常用的 Cassandra 数据面板操作，例如创建键空间和表、读取数据和写入数据。Amazon Key spaces 是无服务器的，因此您无需配置、修补或管理服务器。您也不必安装、维护或操作软件。因此，在 Amazon Keyspaces 中，您无需使用 Cassandra 控制平面 API 操作来管理集群和节点设置。

Amazon Keyspaces 会自动配置重复系数和一致性级别等设置，为您提供高可用性、耐久性和性能。single-digit-millisecond 为了获得更高的弹性和低延迟的本地读取，Amazon Keyspaces 提供了多区域复制。

主题

- 功能差异：Amazon Keyspaces 与 Apache Cassandra

- Amazon Keyspaces 中支持的 Cassandra API、操作、函数和数据类型
- Amazon Keyspaces 中支持的 Apache Cassandra 一致性级别

功能差异：Amazon Keyspaces 与 Apache Cassandra

Apache Cassandra 与 Amazon Keyspaces 之间的功能差异如下。

主题

- Apache Cassandra API、操作和数据类型
- 异步创建与删除键空间和表
- 身份验证和授权
- 批处理
- 集群配置
- 连接
- IN 关键字
- CQL 查询吞吐量调整
- FROZEN 集合
- 轻量级事务
- 负载均衡
- 分页
- 分区器
- 预准备语句
- 范围删除
- 系统表
- 时间戳

Apache Cassandra API、操作和数据类型

Amazon Keyspaces 支持所有常用的 Cassandra 数据面板操作，例如创建键空间和表、读取数据和写入数据。要查看当前支持的项，请参阅Amazon Keyspaces 中支持的 Cassandra API、操作、函数和数据类型。

异步创建与删除键空间和表

Amazon Keyspaces 以异步方式执行数据定义语言 (DDL) 操作，例如创建和删除键空间和表。要了解如何监控资源的创建状态，请参阅the section called “创建键空间”和the section called “创建表”。有关 CQL 语言参考中的 DDL 语句列表，请参阅the section called “DDL 语句”。

身份验证和授权

Amazon Keyspaces（适用于 Apache Cassandra）使用 AWS Identity and Access Management (IAM) 进行用户身份验证和授权，并支持与 Apache Cassandra 等效的授权策略。因此，Amazon Keyspaces 不支持 Apache Cassandra 的安全配置命令。

批处理

Amazon Keyspaces 支持未记录的批处理命令，批处理中最多包含 30 条命令。批处理中仅允许无条件的 INSERT、UPDATE 或 DELETE 命令。不支持记录的批处理。

集群配置

Amazon Keyspaces 是无服务器的，因此没有要配置的集群、主机或 Java 虚拟机 (JVM)。Cassandra 的压缩、缓存、垃圾回收和 bloom 筛选设置不适用于 Amazon Keyspaces，这些设置在指定后将被忽略。

连接

您可以使用现有的 Cassandra 驱动程序与 Amazon Keyspaces 通信，但需要对驱动程序进行不同的配置。Amazon Keyspace 支持每个 TCP 连接每秒最多 3000 次 CQL 查询，但对驱动程序可建立的连接数没有限制。

大多数开源 Cassandra 驱动程序都会建立一个连接到 Cassandra 的连接池，并在该连接池上对查询进行负载均衡。Amazon Keyspaces 向驱动程序公开 9 个对等 IP 地址，而大多数驱动程序的默认行为是与每个对等 IP 地址建立单一连接。因此，使用默认设置的驱动程序的最大 CQL 查询吞吐量将是每秒 27000 次 CQL 查询。

要增大此数字，我们建议增加驱动程序在其连接池中维护的每个 IP 地址的连接数。例如，如果将每个 IP 地址的最大连接数设置为 2，则将使驱动程序的最大吞吐量增加一倍，达到每秒 54000 次 CQL 查询。

作为最佳实践，我们建议将驱动程序配置为每个连接每秒使用 500 个 CQL 查询，以减少开销并改善分配。在这种情况下，每秒 18000 次 CQL 查询需要 36 个连接。配置驱动程序，在 9 个端点上设置 4

个连接，这样可以实现 36 个连接，每秒执行 500 个请求。有关连接最佳实践的更多信息，请参阅the section called “连接”。

与 VPC 端点连接时，可用的端点可能较少。这意味着您必须增加驱动程序配置中的连接数量。有关 VPC 连接最佳实践的更多信息，请参阅the section called “VPC 端点连接”。

IN 关键字

Amazon Keyspaces 支持 SELECT 语句中的 IN 关键字。UPDATE 和 DELETE 不支持 IN。在 SELECT 语句中使用 IN 关键字时，查询结果将按 SELECT 语句中的键的显示顺序返回。在 Cassandra 中，结果按字典顺序排列。

使用 ORDER BY 时，不支持在禁用分页的情况下进行完全重新排序，结果排列在一个页面内。IN 关键字不支持切片查询。IN 关键字不支持 TOKENS。Amazon Keyspaces 通过创建子查询来处理使用 IN 关键字的查询。对于每秒每个 TCP 连接 3000 次 CQL 查询这一限制，每个子查询计为一个连接。有关更多信息，请参阅 the section called “IN SELECT 语句”。

CQL 查询吞吐量调整

Amazon Keyspace 支持每个 TCP 连接每秒最多 3000 次 CQL 查询，但对驱动程序可建立的连接数没有限制。

大多数开源 Cassandra 驱动程序都会建立一个连接到 Cassandra 的连接池，并在该连接池上对查询进行负载均衡。Amazon Keyspaces 向驱动程序公开 9 个对等 IP 地址，而大多数驱动程序的默认行为是与每个对等 IP 地址建立单一连接。因此，使用默认设置的驱动程序的最大 CQL 查询吞吐量将是每秒 27000 次 CQL 查询。

要增大此数字，我们建议增加驱动程序在其连接池中维护的每个 IP 地址的连接数。例如，如果将每个 IP 地址的最大连接数设置为 2，则将使驱动程序的最大吞吐量增加一倍，达到每秒 54000 次 CQL 查询。

FROZEN 集合

Cassandra 中的 FROZEN 关键字会将集合数据类型的多个组件序列化为单个不可变值，该值被视为一个 BLOB。INSERT 和 UPDATE 语句会覆盖整个集合。

默认情况下，Amazon Keyspaces 支持嵌套最多五个级别的冻结集合。有关更多信息，请参阅 the section called “Amazon Keyspaces 服务限额”。

Amazon Keyspaces 不支持在条件性 UPDATE 或 SELECT 语句中使用整个冻结集合的不等式比较。在 Amazon Keyspaces 中，集合和冻结集合的行为是相同的。

当您使用带有客户端时间戳的冻结集合时，如果写入操作的时间戳与某个未过期或未被删除的现有列的时间戳相同，则 Amazon Keyspaces 不会进行比较。相反，它会让服务器确定最新的写入者，并以最新的写入者为准。

有关冻结集合的更多信息，请参阅the section called “集合类型”。

轻量级事务

Amazon Keyspaces（Apache Cassandra 兼容）完全支持在 INSERT、UPDATE 和 DELETE 命令上比较和设置功能，这在 Apache Cassandra 中称为轻量级事务 (LWT)。作为一种无服务器产品，Amazon Keyspaces（Apache Cassandra 兼容）可以提供任何规模下（包括适用于轻量级事务）的一致性能。利用 Amazon Keyspaces，使用轻量级事务不会造成性能损失。

负载均衡

`system.peers` 表条目对应着 Amazon Keyspaces 负载均衡器。要获得最佳结果，我们建议使用轮询负载均衡策略并调整每个 IP 的连接数以满足应用程序需求。

分页

Amazon Keyspaces 根据自己为处理请求而读取的行数（而不是结果中返回的行数）对结果进行分页。因此，某些页面包含的行数可能少于您针对筛选的查询在 PAGE SIZE 中指定的行数。此外，Amazon Keyspaces 会在读取 1 MB 的数据后自动对结果进行分页，从而为客户提供稳定的毫秒级读取性能。有关更多信息，请参阅 the section called “对结果进行分页”。

分区器

Amazon Keyspaces 中的默认分区器是与 Cassandra 兼容的 `Murmur3Partitioner`。此外，您可以选择使用 Amazon Keyspaces `DefaultPartitioner` 或与 Cassandra 兼容的 `RandomPartitioner`。

使用 Amazon Keyspaces，您可以安全地更改账户的分区器，无需重新加载 Amazon Keyspaces 数据。配置更改完成后（大约需要 10 分钟），客户端就会在下次连接时自动看到新的分区器设置。有关更多信息，请参阅 the section called “使用分区程序”。

预准备语句

Amazon Keyspaces 支持使用预准备语句执行数据操作语言 (DML) 操作，例如读取和写入数据。Amazon Keyspaces 目前不支持使用预准备语句执行数据定义语言 (DDL) 操作，例如如创建表和键空间）。DDL 操作必须在预准备语句范围之外执行。

范围删除

Amazon Keyspaces 支持删除一个范围内的行。范围是分区内的一组连续的行。您可以使用 WHERE 子句在 DELETE 操作中指定一个范围。您可以将范围指定为整个分区。

此外，您可以使用关系运算符（例如“>”、“<”），或者通过包含分区键并省略一个或多个集群列，将范围指定为一个分区内的连续行的子集。借助 Amazon Keyspaces，您通过一次操作最多可以删除一个范围内的 1000 行。此外，范围删除是原子性的，但不是孤立的。

系统表

Amazon Keyspaces 会填充 Apache 2.0 开源 Cassandra 驱动程序所需的系统表。对客户端可见的系统表包含经过身份验证的用户所特有的信息。系统表完全由 Amazon Keyspaces 控制，并且是只读的。

用户需要对系统表具有只读访问权限，您可以使用 IAM 访问策略对其进行控制。有关更多信息，请参阅 the section called “使用策略管理访问”。根据您是使用 AWS SDK 还是通过 Cassandra 驱动程序和开发者工具调用 Cassandra 查询语言 (CQL) API，您必须以不同的方式为系统表定义基于标签的访问控制策略。有关针对系统表实现基于标签的访问控制的更多信息，请参阅the section called “ 基于标签的 Amazon Keyspaces 资源访问”。

如果您使用 Amazon VPC 端点访问 Amazon Keyspaces，则会在 `system.peers` 表中看到 Amazon Keyspaces 有权查看的每个 Amazon VPC 端点的条目。因此，您的 Cassandra 驱动程序可能会发出关于 `system.peers` 表中的控制节点本身的警告消息。您可以放心地忽略这一警告。

时间戳

在 Amazon Keyspaces 中，与 Apache Cassandra 中的默认时间戳兼容的单元格级时间戳是一项可选功能。

只有在为表开启客户端时间戳时，`USING TIMESTAMP` 子句和 `WRITETIME` 功能才可以使用。要了解有关 Amazon Keyspaces 中的客户端时间戳的更多信息，请参阅客户端时间戳。

Amazon Keyspaces 中支持的 Cassandra API、操作、函数和数据类型

Amazon Keyspaces（Apache Cassandra 兼容）与 Cassandra Query Language (CQL) 3.11 API（与版本 2.x 向后兼容）兼容。

Amazon Keyspaces 支持所有常用的 Cassandra 数据面板操作，例如创建键空间和表、读取数据和写入数据。

以下部分列出了支持的功能。

主题

Cassandra API 支持

API 操作	支持
CREATE KEYSPACE	是
ALTER KEYSPACE	是
DROP KEYSPACE	是
CREATE TABLE	是
ALTER TABLE	是
DROP TABLE	是
CREATE INDEX	否
DROP INDEX	否
UNLOGGED BATCH	是
LOGGED BATCH	否
SELECT	是
INSERT	是

API 操作	支持
DELETE	是
UPDATE	是
USE	是
CREATE TYPE	否
ALTER TYPE	否
DROP TYPE	否
CREATE TRIGGER	否
DROP TRIGGER	否
CREATE FUNCTION	否
DROP FUNCTION	否
CREATE AGGREGATE	否
DROP AGGREGATE	否
CREATE MATERIALIZED VIEW	否
ALTER MATERIALIZED VIEW	否
DROP MATERIALIZED VIEW	否
TRUNCATE	不可以

Cassandra 控制面板 API 支持

由于 Amazon Keyspaces 是托管的，因此不需要用来管理集群和节点设置的 Cassandra 控制面板 API。因此，以下 Cassandra 功能不适用。

功能	Reason
持久写入切换	所有写入都是持久性的
读取修复设置	不适用
GC 宽限期秒数	不适用
Bloom 筛选条件设置	不适用
压缩设置	不适用
Compression settings (压缩设置)	不适用
缓存设置	不适用
安全设置	替换为 IAM

Cassandra 数据面板 API 支持

功能	支持
针对 SELECT 和 INSERT 语句的 JSON 支持	可以
静态列	可以
生存时间 (TTL)	可以

Cassandra 函数支持

有关受支持的函数的更多信息，请参阅 the section called “内置函数”。

功能	支持
`Aggregate` 函数	不可以
`Blob` 转换	是

功能	支持
Cast	可以
Datetime 函数	可以
Timeconversion 函数	可以
TimeUuid 函数	是
Token	是
User defined functions (UDF)	否
Uuid	可以

Cassandra 数据类型支持

数据类型	支持	备注
ascii	是	
bigint	是	
blob	是	
boolean	是	
counter	是	
date	是	
decimal	是	
double	是	
float	是	
frozen	是	

数据类型	支持	备注
inet	是	
int	是	
list	是	
map	是	
set	是	
smallint	是	
text	是	
time	是	
timestamp	是	
timeuuid	是	
tinyint	是	
tuple	是	
user-defined types (UDT)	不可以	要使用协议缓冲区重构 UDT，请参阅 Amazon Keyspaces Protocol Buffers。
uuid	是	
varchar	是	
varint	可以	

Amazon Keyspaces 中支持的 Apache Cassandra 一致性级别

此部分中的主题介绍了 Amazon Keyspaces（Apache Cassandra 兼容）中的读取和写入操作支持哪些 Apache Cassandra 一致性级别。

主题

写入一致性级别

Amazon Keyspaces 跨多个可用区重复所有写入操作三次以实现持久性和高可用性。在使用 LOCAL_QUORUM 一致性级别确认写入之前，将持久存储写入。就每个 1 KB 的写入而言，对于使用预置容量模式的表，将向您收取 1 个写入容量单位 (WCU) 的费用，对于使用按需模式的表，将收取 1 个写入请求单位 (WRU) 的费用。

您可以使用 cqlsh 并用以下代码将当前会话中所有查询的一致性设置为 LOCAL_QUORUM。

```
CONSISTENCY LOCAL_QUORUM;
```

要以编程方式配置一致性级别，您可以使用相应的 Cassandra 客户端驱动程序设置一致性级别。例如，4.x 版 Java 驱动程序允许您在 app config 文件中设置一致性级别，如下所示。

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

如果您使用的是 3.x 版本的 Java Cassandra 驱动程序，则可以通过添加 .withQueryOptions(new QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM) 来指定会话的一致性级别，如以下代码示例所示。

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

要为特定的写入操作配置一致性级别，可以在使用 Java 驱动程序时使用 setConsistencyLevel 参数调用 QueryBuilder.insertInto 时定义一致性。

读取一致性级别

Amazon Keyspaces 支持三个读取一致性级别：ONE、LOCAL_ONE和 LOCAL_QUORUM。在 LOCAL_QUORUM 读取期间，Amazon Keyspaces 将返回一个响应来反映来自所有先前成功的写入操作的最近更新。通过使用一致性级别，ONE 或 LOCAL_ONE 可以提高读取请求的性能和可用性，但响应可能无法反映最近完成的写入操作的结果。

就每个使用 ONE 或 LOCAL_ONE 一致性的 4 KB 读取操作而言，对于使用预置容量模式的表，将向您收取 0.5 个读取容量单位 (RCU) 的费用，对于使用按需模式的表，将向您收取 0.5 个读取请求单位 (RRU) 的费用。就每个使用 LOCAL_QUORUM 一致性的 4 KB 读取操作而言，对于使用预置容量模式的表，将向您收取 1 个读取容量单位 (RCU) 的费用，对于使用按需模式的表，将向您收取 1 个读取请求单位 (RRU) 的费用。

根据每个表的读取一致性和读取容量吞吐量模式对每个 4 KB 的读取操作进行计费

一致性级别	已预置	按需
ONE	0.5 个 RCU	0.5 个 RRU
LOCAL_ONE	0.5 个 RCU	0.5 个 RRU
LOCAL_QUORUM	1 个 RCU	1 个 RRU

要为读取操作指定不同的一致性，请在使用 Java 驱动程序时使用 `setConsistencyLevel` 参数调用 `QueryBuilder.select`。

不受支持的一致性级别

以下一致性级别不受 Amazon Keyspaces 的支持，并且将导致异常。

不受支持的一致性级别

Apache Cassandra	Amazon Keyspaces
EACH_QUORUM	不支持
QUORUM	不支持
ALL	不支持

Apache Cassandra	Amazon Keyspaces
TWO	不支持
THREE	不支持
ANY	不支持
SERIAL	不支持
LOCAL_SERIAL	不支持

访问 Amazon Keyspaces（Apache Cassandra 兼容）

您可以使用控制台、通过运行`cqlsh`客户端 AWS CloudShell、软件开发工具包或使用 Apache 2.0 许可的 Cassandra 驱动程序以编程方式访问 Amazon AWS Keyspaces。Amazon Keyspaces 支持与 Apache Cassandra 3.11.2 兼容的驱动程序和客户端。在访问亚马逊密钥空间之前，您必须完成设置，AWS Identity and Access Management 然后向亚马逊密钥空间授予 IAM 身份访问权限。

设置 AWS Identity and Access Management

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请 为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建管理用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。

2. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建管理用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，为管理用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

作为管理用户登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

设置 Amazon Keyspaces

使用 IAM 管理对 Amazon Keyspaces 资源的访问。使用 IAM，您可以将策略附加到 IAM 用户、角色和联合身份，以授予对 Amazon Keyspaces 中特定资源的读写权限。

要开始向 IAM 身份授予权限，您可以使用其中一个适用于 Amazon Keys AWS paces 的托管策略：

- AmazonKeyspacesFullAccess— 此政策授予访问亚马逊密钥空间中所有资源的权限，并授予对所有功能的完全访问权限。
- AmazonKeyspacesReadOnlyAccess_v2 — 此策略向 Amazon Keyspaces 授予只读权限。

有关托管策略中定义的操作的详细说明，请参阅the section called “AWS 托管式策略”。

要限制 IAM 身份可以执行的操作范围或限制该身份可以访问的资源，您可以创建一个使用AmazonKeyspacesFullAccess托管策略作为模板的自定义策略，并删除您不需要的所有权限。您

还可以限制对特定密钥空间或表的访问权限。有关如何限制操作或限制对 Amazon Keyspaces 中特定资源的访问的更多信息，请参阅。the section called “Amazon Keyspaces 如何与 IAM 配合使用”

要在创建 AWS 账户 并创建授予 IAM 身份访问亚马逊密钥空间的策略之后访问 Amazon Keyspaces，请继续阅读以下部分之一：

- 使用控制台
- 使用 AWS CloudShell
- 以编程方式连接

使用该控制台访问 Amazon Keyspaces

您可以通过以下网址访问 Amazon Keyspaces控制台：https://console.aws.amazon.com/keyspaces/home。有关 AWS Management Console 访问权限的更多信息，请参阅 IAM 用户指南 AWS Management Console中的控制 IAM 用户对的访问权限。

您可以使用控制台在 Amazon Keyspaces 中执行以下操作：

- 创建、删除和管理密钥空间和表。
- 在表格的 “监控” 选项卡上监控重要的表格指标：
  - 计费表大小（字节）
  - 容量指标
- 使用 CQL 编辑器运行查询，例如插入、更新和删除数据。
- 更改账户的分区器配置。
- 在控制面板上查看账户的绩效和错误指标。

要了解如何创建 Amazon Keyspaces 键空间和表以及使用示例应用程序数据进行设置，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）。

AWS CloudShell 用于访问 Amazon Keyspaces

AWS CloudShell 是一个基于浏览器、经过预先验证的 shell，您可以直接从启动。 AWS Management Console您可以使用首选外壳（Bash PowerShell 或 Z shell）对 AWS 服务运行 AWS CLI 命令。要使用 Amazon Keyspaces 使用`cqlsh`，您必须安装。`cqlsh-expansion`有关`cqlsh-expansion`安装说明，请参阅the section called “使用`cqlsh-expansion`”。

您可以AWS CloudShell 从启动 AWS Management Console，用于登录控制台的 AWS 凭据将在新的 shell 会话中自动可用。这种对 AWS CloudShell 用户的预身份验证允许您在cqlsh使用 AWS CLI 或版本 2（预安装在外壳的计算环境中）与 Amazon Keyspaces 等 AWS 服务进行交互时跳过配置凭证。

获取 IAM 权限 AWS CloudShell

使用提供的访问管理资源 AWS Identity and Access Management，管理员可以向 IAM 用户授予权限，使他们能够访问 AWS CloudShell 和使用环境的功能。

管理员向用户授予访问权限的最快方法是通过 AWS 托管策略。AWS 托管式策略 是由 AWS创建和管理的独立策略。以下的 AWS 托管策略 CloudShell 可以附加到 IAM 身份：

- `AWSCloudShellFullAccess`：授予使用权限，并 AWS CloudShell 具有对所有功能的完全访问权限。

如果您想限制 IAM 用户可以执行的操作范围 AWS CloudShell，则可以创建使用`AWSCloudShellFullAccess`托管策略作为模板的自定义策略。有关限制中可供用户执行的操作的更多信息 CloudShell，请参阅AWS CloudShell 用户指南中的使用 IAM 策略管理 AWS CloudShell 访问和使用情况。

> **Note**
>
> 您的 IAM 身份还需要一项策略，该策略允许您调用 Amazon Keyspaces。

您可以使用 AWS 托管策略向您的 Amazon Keyspaces 授予您的 IAM 身份访问权限，也可以从托管策略作为模板开始删除不需要的权限。您还可以限制对特定密钥空间和表的访问权限以创建自定义策略。以下适用于 Amazon Keyspaces 的托管策略可以附加到 IAM 身份：

- AmazonKeyspacesFullAccess— 本政策允许用户使用 Amazon Keyspaces，并拥有对所有功能的完全访问权限。

有关托管策略中定义的操作的详细说明，请参阅the section called “AWS 托管式策略”。

有关如何限制操作或限制对 Amazon Keyspaces 中特定资源的访问的更多信息，请参阅。the section called “Amazon Keyspaces 如何与 IAM 配合使用”

使用与 Amazon Keyspaces 互动 AWS CloudShell

AWS CloudShell 从启动后 AWS Management Console，您可以使用cqlsh或命令行界面立即开始与 Amazon Keyspaces 进行交互。如果您尚未安装cqlsh-expansion，请参阅，了解the section called “使用cqlsh-expansion”详细步骤。

> **Note**
>
> 使用 in 时 AWS CloudShell，您无需cqlsh-expansion在调用之前配置凭据，因为您已经在 shell 中进行了身份验证。

连接到 Amazon Keyspaces 并创建新的密钥空间。然后从系统表中读取以确认密钥空间是使用创建的 AWS CloudShell

1. 从中 AWS Management Console，您可以 CloudShell 通过选择导航栏上的以下可用选项来启动：
   - 选择图 CloudShell 标。
   - 开始在搜索框中输入 “cloudshell”，然后选择相应 CloudShell选项。
2. 您可以使用以下命令建立与 Amazon Keyspaces 的连接。请务必将 *cassandra.us-east-1.amazonaws.com* 替换为您所在地区的正确终端节点。

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

   如果连接成功，您应该会看到类似于以下内容的输出：

   ```
   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. 使用名称mykeyspace创建新的密钥空间。你可以使用以下命令来做到这一点。

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

4. 要确认密钥空间已创建，您可以使用以下命令从系统表中读取。

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

如果调用成功，命令行将显示来自服务的响应，输出与以下类似：

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

(1 rows)
```

以编程方式连接到 Amazon Keyspaces

本主题概述了以编程方式连接 Amazon Keyspaces 所需的步骤。它会指导您创建 IAM 凭证并列出可用的 AWS 服务终端节点。最后一部分介绍了如何使用 cqlsh 连接 Amazon Keyspaces。有关使用不同的 Apache Cassandra 驱动程序连接到 Amazon Keyspaces 的 step-by-step 教程，请参阅。the section called “使用 Cassandra 客户端驱动程序”有关展示如何从 Amazon VPC 终端节点连接到 Amazon Keyspaces 的 step-by-step 教程，请参阅。the section called “连接 VPC 终端节点”

> **Note**
>
> 为了帮助您入门，您可以在上的 Amazon Keyspaces end-to-end 代码示例存储库中找到使用各种 Cassandra 客户端驱动程序连接亚马逊密钥空间的代码示例。GitHub

Amazon Keyspaces 支持与 Apache Cassandra 3.11.2 兼容的驱动程序和客户端。它假设您已经完成了中的 AWS 设置说明访问 Amazon Keyspaces。

如果您已有 AWS 账户，请参阅以下主题，了解如何使用 cqlsh 以编程方式访问 Amazon Keyspaces：

主题

- 创建凭证以通过编程方式访问 Amazon Keyspaces
- Amazon Keyspaces 的服务端点
- 使用 cqlsh 连接 Amazon Keyspaces
- 使用 AWS CLI
- 使用 API
- 将 Amazon Keyspaces 与 SDK 配合使用 AWS

- 使用 Cassandra 客户端驱动程序以编程方式访问 Amazon Keyspaces
- 教程：从亚马逊 Elastic Kubernetes Service 连接到亚马逊密钥空间 Amazon Kubernetes Service

创建凭证以通过编程方式访问 Amazon Keyspaces

要为用户和应用程序提供凭证，以通过编程方式访问 Amazon Keyspaces 资源，您可以执行以下任一操作：

- 创建与 Cassandra 用于身份验证和访问管理的传统用户名和密码相似的特定服务凭证。 AWS 特定于服务的凭证与特定 AWS Identity and Access Management (IAM) 用户相关联，并且只能用于为其创建的服务。有关详细信息，请参阅《IAM 用户指南》中的结合使用 IAM 与 Amazon Keyspaces（Apache Cassandra 兼容）。

  > **Warning**
  >
  > IAM 用户拥有长期证书，这会带来安全风险。为帮助减轻这种风险，我们建议仅向这些用户提供执行任务所需的权限，并在不再需要这些用户时将其移除。

- 为了增强安全性，我们建议创建用于所有 AWS 服务的 IAM 身份并使用临时证书。借助适用于 Cassandra 客户端驱动程序的 Amazon Keyspaces SigV4 身份验证插件，您可以使用 IAM 访问密钥而不是用户名和密码来验证对 Amazon Keyspaces 的调用。要进一步了解 Amazon Keyspaces SigV4 插件如何支持 IAM 用户、角色和联合身份在 Amazon Keyspaces API 请求中进行身份验证，请参阅 AWS 签名版本 4 流程 (SigV4)。

  您可以从以下位置下载 SigV4 插件。

  - 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。

  有关如何使用 SigV4 身份验证插件建立连接的代码示例，请参阅the section called “使用 Cassandra 客户端驱动程序”。

主题

- 生成服务特定凭证
- 如何为 Amazon Keyspaces 创建和配置 AWS 证书

生成服务特定凭证

服务特定凭证与 Cassandra 用于身份验证和访问管理的传统用户名和密码类似。使用服务特定凭证，IAM 用户可以访问特定的 AWS 服务。这些长期凭证不能用于访问其他 AWS 服务。它们与特定的 IAM 用户关联，不能由其他 IAM 用户使用。

> **Important**
>
> 服务专用证书是与特定 IAM 用户关联的长期证书，只能用于为其创建的服务。要授予 IAM 角色或联合身份使用临时证书访问您的所有 AWS 资源的权限，您应使用AWS 适用于 A mazon Keyspaces 的 Sigv4 身份验证插件进行身份验证。

使用以下过程之一生成特定于服务的凭证。

使用控制台生成服务特定凭证

使用控制台生成服务特定凭证

1. 登录 AWS Management Console 并打开 AWS Identity and Access Management 控制台，网址为https://console.aws.amazon.com/iam/home。
2. 在导航窗格中，选择用户，然后选择之前创建的具有 Amazon Keyspaces 权限（已附加策略）的用户。
3. 选择 Security Credentials (安全凭证)。在 Amazon Keyspaces 的凭证下，选择生成凭证以生成服务特定凭证。

   您的服务特定凭证现在可用。这是您下载或查看密码的唯一机会。以后您无法恢复它。不过，您可以随时重置密码。将用户和密码保存在安全的位置，因为随后您需要使用它们。

使用生成特定于服务的凭证 AWS CLI

要生成特定于服务的凭证，请使用 AWS CLI

在生成特定于服务的凭证之前，您需要下载、安装和配置 AWS Command Line Interface (AWS CLI)：

1. AWS CLI 在 http://aws.amazon.com/cli 下载。

> **Note**
>
> 它们可以在 Windows、macOS 或 Linux 上 AWS CLI 运行。

2. 按照AWS Command Line Interface 用户指南中有关安装 AWS CLI 和配置 AWS CLI 的说明进行操作。
3. 使用运行以下命令为用户alice生成特定于服务的证书，以便她可以访问 Amazon Keyspaces。AWS CLI

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

输出如下所示。

```
{
    "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"
    }
}
```

在输出中，记下 ServiceUserName 和 ServicePassword 的值。将这些值保存在安全位置，因为稍后将需要它们。

> **Important**
>
> 这是您唯一能够看到 ServicePassword 的机会。

如何为 Amazon Keyspaces 创建和配置 AWS 证书

要使用 AWS CLI、软件开发工具包或 Cassandra 客户端驱动程序和 Sigv4 插件以编程方式访问 Amazon AWS Keyspaces，您需要一个具有访问密钥的 IAM 用户或角色。当您 AWS 以编程方式使用时，您需要提供 AWS 访问密钥，以便 AWS 可以在编程调用中验证您的身份。您的访问密钥由访问密钥 ID（例如，AKIAIOSFODNN7EXAMPLE）和秘密访问密钥（例如 wjalrxutnfemi/k7mdeng/ CYEX bPxRfi AMPLEKEY）组成。AKIAIOSFODNN7EXAMPLE 本主题将引导您完成此过程中的必要步骤。

安全最佳实践建议您创建权限有限的 IAM 用户，改为将 IAM 角色与执行特定任务所需的权限相关联。然后，IAM 用户可以临时担任 IAM 角色来执行所需任务。例如，您账户中使用 Amazon Keyspaces 控制台的 IAM 用户可以在控制台中切换到角色以临时使用该角色的权限。该用户放弃自己的原始权限，采用分配给该角色的权限。用户退出角色时，将恢复其原始权限。用户用于担任该角色的凭证是临时的。相反，IAM 用户拥有长期证书，如果他们没有担任角色，而是直接分配给他们的权限，则会带来安全风险。为帮助减轻这种风险，我们建议仅向这些用户提供执行任务所需的权限，并在不再需要这些用户时将其移除。有关角色的更多信息，请参阅 IAM 用户指南中的角色常见场景：用户、应用程序和服务。

主题

- 适用于 Cassandra AWS CLI客户端驱动程序的、 AWS 软件开发工具包或 Amazon Keyspaces Sigv4 插件所需的凭证
- 创建 IAM 用户以编程方式访问您账户中的 Amazon Keyspaces AWS
- 为 IAM 用户创建新访问密钥
- 如何管理 IAM 用户的访问密钥
- 使用 IAM 角色和 Sigv4 插件以及临时凭证连接 Amazon Keyspaces

适用于 Cassandra AWS CLI客户端驱动程序的、 AWS 软件开发工具包或 Amazon Keyspaces Sigv4 插件所需的凭证

对 IAM 用户或角色进行身份验证需要以下凭证：

`AWS_ACCESS_KEY_ID`

指定与 IAM 用户或角色关联的 AWS 访问密钥。

以编程方式连接 Amazon Keyspaces 需要使用访问密钥 `aws_access_key_id`。

`AWS_SECRET_ACCESS_KEY`

指定与访问密钥关联的私有密钥。这基本上是访问密钥的“密码”。

以编程方式连接 Amazon Keyspaces 需要使用 `aws_secret_access_key`。

`AWS_SESSION_TOKEN` – 可选。

指定在使用您直接从 AWS Security Token Service 操作中检索的临时安全凭证时需要的会话令牌值。有关更多信息，请参阅 the section called “使用临时凭证连接 Amazon Keyspaces”。

如果您使用 IAM 用户进行连接，则不需要 `aws_session_token`。

创建 IAM 用户以编程方式访问您账户中的 Amazon Keyspaces AWS

要通过 AWS CLI、软件开发工具包或 Sigv4 插件获取以编程方式访问 Amazon AWS Keyspaces 的证书，您需要先创建 IAM 用户或角色。以下步骤显示了创建 IAM 用户并将该 IAM 用户配置为以编程方式访问 Amazon Keyspaces 的过程：

1. 在 AWS Management Console、、Windows 工具中创建用户 PowerShell，或者使用 AWS API 操作创建用户。 AWS CLI如果您在中创建用户 AWS Management Console，则会自动创建凭据。
2. 如果以编程方式创建用户，您必须通过额外步骤为该用户创建访问密钥（访问密钥 ID 和秘密访问密钥）。
3. 向用户授予访问 Amazon Keyspaces 的权限。

有关创建用户所需的权限的信息，请参阅访问 IAM 资源所需的权限。

创建 IAM 用户（控制台）

您可以使用 AWS Management Console 来创建 IAM 用户。

创建具有编程访问权限的 IAM 用户（控制台）

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择 Users（用户），然后选择 Add users（添加用户）。
3. 为新用户键入用户名。这是的登录名。 AWS

> **Note**
>
> 用户名可以是一个最多由 64 个字母、数字和以下字符构成的组合：加号 (+)、等号 (=)、逗号 (,)、句点 (.)、at 符号 (@)、下划线 (_) 和连字符 (-)。账户中的名称必须唯一。名称不区分大小写。例如，您不能创建名为 TESTUSER 和 testuser 的两个用户。

4. 选择访问密钥 - 编程访问，为新用户创建一个访问密钥。您可以在转到最终页面后查看或下载访问密钥。

   选择下一步：权限。

5. 在设置权限页面，选择直接附加现有策略，为新用户分配权限。

   此选项显示您账户中可用的 AWS 托管和客户托管策略列表。您可以在搜索字段中输入 `keyspaces`，以仅显示与 Amazon Keyspaces 相关的策略。

   对于 Amazon Keyspaces，可用的托管策略是 `AmazonKeyspacesFullAccess` 和 `AmazonKeyspacesReadOnlyAccess`。有关每项策略的更多信息，请参阅 the section called “AWS 托管式策略”。

   出于测试目的和按照连接教程进行操作，请为新 IAM 用户选择`AmazonKeyspacesReadOnlyAccess`策略。注意：作为最佳实践，我们建议您遵循最低权限原则并创建自定义策略，限制对特定资源的访问并仅允许所需操作。有关 IAM policy 的更多信息以及查看 Amazon Keyspaces 的示例策略，请参阅the section called “Amazon Keyspaces 基于身份的策略”。创建自定义权限策略后，将您的策略附加到角色，然后让用户暂时扮演相应的角色。

   选择下一步：标签。

6. 在添加标签（可选）页面，您可以为用户添加标签，或选择下一步：审核。

7. 在审核页面，您可以看到目前所做的所有选择。如果您已准备好继续，请选择创建用户。

8. 要查看用户的访问密钥（访问密钥 ID 和秘密访问密钥），请选择密码和访问密钥旁边的 Show (显示)。要保存访问密钥，请选择下载 .csv，然后将文件保存到安全位置。

   > **Important**
   >
   > 这是您查看或下载秘密访问密钥的唯一机会，您需要提供这些信息，他们才能使用 SigV4 插件。将用户的新访问密钥 ID 和秘密访问密钥保存在安全的地方。完成此步骤后，您再也无法访问这些秘密访问密钥。

创建 IAM 用户 (AWS CLI)

您可以使用创建 AWS CLI IAM 用户。

创建具有编程访问权限的 IAM 用户 (AWS CLI)

1. 使用以下 AWS CLI 代码创建用户。

   - aws iam create-user

2. 向用户提供编程访问权限。这需要访问密钥，访问密钥可通过以下方式生成。

   - AWS CLI: aws iam create-access-key
   - 适用于 Windows 的工具 PowerShell：New-IAMAccessKey
   - IAM API：CreateAccessKey

   > **Important**
   >
   > 这是您查看或下载秘密访问密钥的唯一机会，您需要提供这些信息，他们才能使用 SigV4 插件。将用户的新访问密钥 ID 和秘密访问密钥保存在安全的地方。完成此步骤后，您再也无法访问这些秘密访问密钥。

3. 将 `AmazonKeyspacesReadOnlyAccess` 策略（用于定义用户权限的）附加到用户。注意：作为最最佳实践，我们建议您通过将用户添加到组并向该组附加策略（而不是直接附加到用户）来管理用户权限。

   - AWS CLI: aws iam attach-user-policy

为 IAM 用户创建新访问密钥

如果您已经有 IAM 用户，可以随时创建新的访问密钥。有关密钥管理的更多信息（例如如何轮换访问密钥），请参阅管理 IAM 用户的访问密钥。

为 IAM 用户创建访问密钥（控制台）

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择用户。
3. 选择所需用户的名称，以便为其创建访问密钥。
4. 在用户的摘要页面身上，选择安全凭证选项卡。
5. 在 Access keys（访问密钥）部分中，选择 Create access key（创建访问密钥）。

   要查看新访问密钥对，请选择 Show (显示)。您的凭证与下面类似：

- 访问密钥 ID：AKIAIOSFODNN7EXAMPLE
- 秘密访问密钥：wjal bPxRfi rxutnfemi/k7mdeng/ CYEXAMPLEKEY

> **Note**
>
> 关闭此对话框后，您将无法再次访问该秘密访问密钥。

6. 要下载密钥对，请选择 Download .csv file（下载 .csv 文件）。将密钥存储在安全位置。
7. 下载 csv 格式文件之后，选择 Close（关闭）。

在创建访问密钥时，预设情况下，密钥对处于活动状态，并且您可以立即使用此密钥对。

如何管理 IAM 用户的访问密钥

作为最佳实践，我们建议您不要直接将访问密钥嵌入到代码中。借 AWS 助 SDK 和 AWS 命令行工具，您可以将访问密钥放在已知位置，这样您就不必将其保存在代码中。在以下任一位置中放置访问密钥：

- 环境变量：在多租户系统上，选择用户环境变量，而不是系统环境变量。
- CLI 凭证文件 – 在运行 `credentials` 命令时，将更新 `config` 和 `aws configure` 文件。`credentials` 文件在 Linux、macOS 或 Unix 上位于 `~/.aws/credentials`，在 Windows 上位于 `C:\Users\`*`USERNAME`*`\.aws\credentials`。该文件可以包含 `default` 配置文件和任何命名配置文件的凭证详细信息。
- CLI 配置文件 – 在运行 `credentials` 命令时，将更新 `config` 和 `aws configure` 文件。`config` 文件在 Linux、macOS 或 Unix 上位于 `~/.aws/config`，在 Windows 上位于 `C:\Users\`*`USERNAME`*`\.aws\config`。该文件包含原定设置配置文件和任何命名配置文件的配置设置。

将访问密钥存储为环境变量是the section called “Java 4.x 的身份验证插件”的前提条件。客户端使用默认凭证提供程序链来搜索凭证，存储为环境变量的访问密钥优先于所有其他位置，例如配置文件。有关更多信息，请参阅配置设置和优先顺序。

下面的示例介绍您如何可以为默认用户配置环境变量。

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>
```

设置环境变量会更改使用的值，直到 Shell 会话结束或直到您将该变量设置为其他值。通过在 shell 的启动脚本中设置变量，可使变量在未来的会话中继续有效。

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>
```

使用 `set` 设置环境变量会更改使用的值，直到当前命令提示符会话结束，或者直到您将该变量设置为其他值。使用 `setx` 设置环境变量会更改当前命令提示符会话和运行该命令后创建的所有命令提示符会话中使用的值。它不 影响在运行该命令时已经运行的其他命令 shell。

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>"
```

如果您在 PowerShell 提示符处设置环境变量（如前面的示例所示），则它只会在当前会话的持续时间内保存该值。要使环境变量设置在所有会话 PowerShell 和命令提示符会话中保持不变，请使用控制面板中的系统应用程序将其存储。或者，您可以通过将变量添加到您的 PowerShell 个人资料中来为所有将来的 PowerShell 会话设置该变量。有关存储环境变量或跨会话保存环境变量的更多信息，请参阅PowerShell 文档。

使用 IAM 角色和 Sigv4 插件以及临时凭证连接 Amazon Keyspaces

为增强安全性，您可以使用临时凭证和 SigV4 插件进行身份验证。在许多情况下，您并不需要永不过期的长期访问密钥（如 IAM 用户访问密钥）。相反，您可以创建一个 IAM 角色并生成临时安全凭证。临时安全证书包括访问密钥 ID 和秘密访问密钥，以及一个指示证书何时到期的安全令牌。要详细了解如何使用 IAM 角色而不是长期访问密钥，请参阅切换到 IAM 角色 (AWS API)。

要开始使用临时凭证，您首先需要创建一个 IAM 角色。

创建授予对 Amazon Keyspaces 的只读访问权的 IAM 角色。

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色，然后选择创建角色。
3. 在创建角色页面上的选择受信实体类型下，选择 AWS 服务。在使用案例下，选择 Amazon EC2，然后选择下一步。
4. 在添加权限页面的权限策略下，从策略列表中选择 Amazon Keyspaces 只读访问，然后选择下一步。
5. 在命名、审核和创建页面上，为角色输入名称，并查看选择受信实体和添加权限部分。您还可以在此页面上为角色添加可选标签。完成后，选择创建角色。请记住此名称，因为您在启动 Amazon EC2 实例时会用到它。

要在代码中使用临时安全证书，您需要以编程方式调用 AWS Security Token Service API，`AssumeRole`并从您在上一步中创建的 IAM 角色中提取生成的证书和会话令牌。然后，您可以使用这些值作为后续调用的凭据 AWS。下面的示例展示了有关如何使用临时安全凭证的伪代码：

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

有关使用 Python 驱动程序实施临时凭证以访问 Amazon Keyspaces 的示例，请参阅???。

有关如何调用 `AssumeRole`、`GetFederationToken` 和其他 API 操作的详细信息，请参阅 AWS Security Token Service API 参考。有关从结果中获取临时安全凭证和会话令牌的信息，请参阅所用开发工具包的文档。您可以在主文档页面的 AWS 软件开发工具包和工具包部分找到所有软件开发工具包的AWS 文档。

Amazon Keyspaces 的服务端点

主题

- 端口和协议
- 全局端点
- AWS GovCloud (US) Region FIPS 端点

- 中国区域端点

端口和协议

您可以运行 cqlsh 客户端、使用获得 Apache 2.0 许可的 Cassandra 驱动程序或使用 AWS CLI 和 AWS SDK，以编程方式访问 Amazon Keyspaces。

下表显示了不同访问机制使用的端口和协议。

以编程方式访问	端口	协议
CQLSH	9142	TLS
Cassandra 驱动程序	9142	TLS
AWS CLI	443	HTTPS
AWS SDK	443	HTTPS

对于 TLS 连接，Amazon Keyspaces 使用 Starfield CA 对服务器进行身份验证。有关更多信息，请参阅the section called “如何为 TLS 手动配置 cqlsh 连接”，或者参阅the section called “使用 Cassandra 客户端驱动程序”中的开始前的准备工作部分。

全局端点

Amazon Keyspaces 支持以下 AWS 区域。下表显示了每个区域的可用服务端点。

区域名称	区域	端点	协议	
美国东部（俄亥俄州）	us-east-2	cassandra.us-east-2.amazonaws.com	HTTPS 和 TLS	
美国东部（弗吉尼亚州北部）	us-east-1	cassandra.us-east-1.amazonaws.com cassandra-fips.us-east-1.amazonaws.com	HTTPS 和 TLS TLS	

区域名称	区域	端点	协议
美国西部（北加利福尼亚）	us-west-1	cassandra.us-west-1.amazonaws.com	HTTPS 和 TLS
美国西部（俄勒冈州）	us-west-2	cassandra.us-west-2.amazonaws.com cassandra-fips.us-west-2.amazonaws.com	HTTPS 和 TLS TLS
亚太地区（香港）	ap-east-1	cassandra.ap-east-1.amazonaws.com	HTTPS 和 TLS
亚太地区（孟买）	ap-south-1	cassandra.ap-south-1.amazonaws.com	HTTPS 和 TLS
亚太地区（首尔）	ap-northeast-2	cassandra.ap-northeast-2.amazonaws.com	HTTPS 和 TLS
亚太地区（新加坡）	ap-southeast-1	cassandra.ap-southeast-1.amazonaws.com	HTTPS 和 TLS
亚太地区（悉尼）	ap-southeast-2	cassandra.ap-southeast-2.amazonaws.com	HTTPS 和 TLS
亚太地区（东京)	ap-northeast-1	cassandra.ap-northeast-1.amazonaws.com	HTTPS 和 TLS
加拿大（中部）	ca-central-1	cassandra.ca-central-1.amazonaws.com	HTTPS 和 TLS
欧洲（法兰克福）	eu-central-1	cassandra.eu-central-1.amazonaws.com	HTTPS 和 TLS
欧洲（爱尔兰）	eu-west-1	cassandra.eu-west-1.amazonaws.com	HTTPS 和 TLS

区域名称	区域	端点	协议
欧洲（伦敦）	eu-west-2	cassandra.eu-west-2.amazonaws.com	HTTPS 和 TLS
欧洲（巴黎）	eu-west-3	cassandra.eu-west-3.amazonaws.com	HTTPS 和 TLS
欧洲（斯德哥尔摩）	eu-north-1	cassandra.eu-north-1.amazonaws.com	HTTPS 和 TLS
中东（巴林）	me-south-1	cassandra.me-south-1.amazonaws.com	HTTPS 和 TLS
南美洲（圣保罗）	sa-east-1	cassandra.sa-east-1.amazonaws.com	HTTPS 和 TLS
AWS GovCloud（美国东部）	us-gov-east-1	cassandra.us-gov-east-1.amazonaws.com	HTTPS 和 TLS
AWS GovCloud（美国西部）	us-gov-west-1	cassandra.us-gov-west-1.amazonaws.com	HTTPS 和 TLS

AWS GovCloud (US) Region FIPS 端点

AWS GovCloud (US) Region 中可用的 FIPS 端点。有关更多信息，请参阅 AWS GovCloud (US) 用户指南中的 Amazon Keyspaces。

区域名称	区域	FIPS 端点	协议
AWS GovCloud（美国东部）	us-gov-east-1	cassandra.us-gov-east-1.amazonaws.com	HTTPS 和 TLS
AWS GovCloud（美国西部）	us-gov-west-1	cassandra.us-gov-west-1.amazonaws.com	HTTPS 和 TLS

中国区域端点

AWS 中国区域提供以下 Amazon Keyspaces 端点。

要访问这些端点，您必须注册一组中国区域独有的单独账户凭证。有关更多信息，请参阅中国注册、账户和凭证。

区域名称	区域	端点	协议
中国（北京）	cn-north-1	cassandra.cn-north-1.amazonaws.com.cn	HTTPS 和 TLS
中国（宁夏）	cn-northwest-1	cassandra.cn-northwest-1.amazonaws.com.cn	HTTPS 和 TLS

使用 `cqlsh` 连接 Amazon Keyspaces

要使用 `cqlsh` 连接 Amazon Keyspaces，您可以使用 `cqlsh-expansion`。该工具包包含常用的 Apache Cassandra 工具（例如 `cqlsh` 和帮助程序），能够对 Amazon Keyspaces 进行预配置，同时保持与 Apache Cassandra 的完全兼容。`cqlsh-expansion` 与 SigV4 身份验证插件集成，让您可以使用 IAM 访问密钥而不是用户名和密码进行连接。您只需安装 `cqlsh` 脚本即可建立连接，无需安装完整的 Apache Cassandra 发行版，因为 Amazon Keyspaces 是无服务器的。这一轻量级安装包包括 `cqlsh-expansion` 和经典 `cqlsh` 脚本，可以在任何支持 Python 的平台上安装。

有关 `cqlsh` 的一般信息，请参阅 `cqlsh：CQL Shell`。

主题

- 使用 cqlsh-expansion 连接 Amazon Keyspaces
- 如何为 TLS 手动配置 cqlsh 连接

使用 **cqlsh-expansion** 连接 Amazon Keyspaces

安装和配置 **cqlsh-expansion**

1. 要安装 cqlsh-expansion Python 软件包，您可以运行 pip 命令。该命令将使用 pip 安装在您的设备上安装 cqlsh-expansion 脚本以及一个包含依赖项列表的文件。--user flag 会告诉 pip 使用 Python 用户为平台安装目录。在基于 Unix 的系统上，应安装 ~/.local/ 目录。

   您需要使用 Python 3 来安装 cqlsh-expansion，要弄清楚您的 Python 版本，可使用 Python --version。要进行安装，可以运行以下命令。

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

   输出应如下所示：

   ```
   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
```

如果安装目录不在中PATH，则需要按照操作系统的说明进行添加。以下是 Ubuntu Linux 的一个示例。

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

要确认软件包已安装，可以运行以下命令。

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

输出应如下所示：

```
cqlsh 6.1.0
```

2. 要配置 cqlsh-expansion，您可以运行安装后脚本来自动完成以下步骤：

   1. 在用户主目录下创建 .cassandra 目录（如果该目录尚不存在）。
   2. 将预配置的 cqlshrc 配置文件复制到 .cassandra 目录。

3. 将 Starfield 数字证书复制到 `.cassandra` 目录。Amazon Keyspaces 使用此证书通过传输层安全性协议 (TLS) 配置安全连接。传输中加密可在数据进出 Amazon Keyspaces 时对其进行加密，从而提供额外一层数据保护。

要先查看脚本，您可以访问 `post_install.py` 中的 Github 代码库中。

要使用脚本，您可以运行以下命令。

```
cqlsh-expansion.init
```

> **Note**
>
> 使用 `pip uninstall` 卸载 `cqlsh-expansion` 时，安装后脚本创建的目录和文件不会随之删除，必须要手动删除。

使用 **`cqlsh-expansion`** 连接 Amazon Keyspaces

1. 配置您的 AWS 区域 并将其添加为用户环境变量。

   要在基于 Unix 的系统上将默认区域添加为环境变量，可以运行以下命令。在本示例中，我们使用的是美国东部（弗吉尼亚州北部）。

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

   要详细了解如何设置环境变量（包括其他平台），请参阅如何设置环境变量。

2. 找到您的服务端点。

   为您的区域选择合适的服务端点。要查看 Amazon Keyspaces 的可用端点，请参阅 the section called “服务端点”。在本示例中，我们使用的端点是 `cassandra.us-east-1.amazonaws.com`。

3. 配置身份验证方法。

   要增强安全性，建议您使用 IAM 访问密钥（IAM 用户、角色和联合身份）进行连接。

   在使用 IAM 访问密钥进行连接之前，您需要完成以下步骤：

a. 创建 IAM 用户，或者遵循最佳实践创建一个 IAM 用户可以代入的 IAM 角色。有关如何创建 IAM 访问密钥的更多信息，请参阅the section called “用于 AWS 身份验证的 IAM 证书”。

b. 创建一项 IAM 策略，至少向角色（或 IAM 用户）授予对 Amazon Keyspaces 的只读访问权限。要详细了解 IAM 用户或角色连接 Amazon Keyspaces 所需的权限，请参阅the section called “访问 Amazon Keyspaces 表”。

c. 将 IAM 用户的访问密钥添加到用户的环境变量中，如下例中所示。

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

要详细了解如何设置环境变量（包括其他平台），请参阅如何设置环境变量。

> **Note**
>
> 如果您从 Amazon EC2 实例进行连接，则还需要在安全组中配置一条出站规则，允许从该实例到 Amazon Keyspaces 的流量。有关如何查看和修改 EC2 出站规则的更多信息，请参阅《适用于 Linux 实例的 Amazon EC2 用户指南》中的向安全组添加规则。

4. 使用 `cqlsh-expansion` 和 SigV4 身份验证连接到 Amazon Keyspaces。

要使用 `cqlsh-expansion` 连接到 Amazon Keyspaces，您可以使用以下命令。确保将服务端点替换为您所在区域的正确端点。

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

如果连接成功，您应该会看到类似于以下内容的输出：

```
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>
```

如果您遇到连接错误，请参见the section called “Cqlsh 连接错误”以获取故障排除信息。

- 使用服务特定凭证连接 Amazon Keyspaces。

要使用 Cassandra 用于身份验证的传统用户名和密码组合进行连接，您必须首先为 Amazon Keyspaces 创建服务特定凭证，如the section called “服务特定凭证”中所述。您还必须向该用户授予访问 Amazon Keyspaces 的权限，更多信息请参阅the section called “访问 Amazon Keyspaces 表”。

为用户创建服务特定凭证和权限后，您必须更新 cqlshrc 文件，该文件通常位于用户目录路径 ~/.cassandra/ 中。在 cqlshrc 文件中，转到 Cassandra [authentication] 部分，使用 ";" 字符注释掉 [auth_provider] 下的 SigV4 模块和类，如以下示例所示。

```
[auth_provider]

; module = cassandra_sigv4.auth

; classname = SigV4AuthProvider
```

更新 cqlshrc 文件后，您可以使用以下命令通过服务特定凭证连接 Amazon Keyspaces。

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

清理

- 要删除 cqlsh-expansion 软件包，您可以使用 pip uninstall 命令。

  ```
  pip3 uninstall cqlsh-expansion
  ```

  pip3 uninstall 命令不会删除安装后脚本创建的目录和相关文件。要删除安装后脚本创建的文件夹和文件，您可以删除 .cassandra 目录。

如何为 TLS 手动配置 cqlsh 连接

Amazon Keyspaces 只接受使用传输层安全性协议 (TLS) 的安全连接。您可以使用 cqlsh-expansion 实用程序自动下载证书并安装预配置的 cqlshrc 配置文件。有关更多信息，请参阅本页上的the section called “使用cqlsh-expansion”。

如果要下载证书并手动配置连接，您可以使用以下步骤。

1. 使用以下命令下载 Starfield 数字证书，并将 `sf-class2-root.crt` 保存在本地或您的主目录中。

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

> **Note**
>
> 您还可以使用 Amazon 数字证书连接到 Amazon Keyspaces。如果您的客户端成功连接到 Amazon Keyspaces，您可以继续这样做。Starfield 证书为使用旧证书颁发机构的客户端提供了额外的向后兼容性。

2. 打开 Cassandra 主目录下的 `cqlshrc` 配置文件（例如 `${HOME}/.cassandra/cqlshrc`），然后添加以下行。

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

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

使用 AWS CLI

您可以使用 AWS Command Line Interface (AWS CLI) 从命令行管理多个 AWS 服务并通过脚本自动执行这些服务。使用 Amazon Keyspaces，您可以使用 AWS CLI 进行数据定义语言 (DDL) 操作，例如创建表格。此外，您还可以使用基础设施即代码 (IaC) 服务和工具，如 AWS CloudFormation 和 Terraform。

您必先获取访问密钥 ID 和秘密访问密钥，然后才能将 AWS CLI 与 Amazon Keyspaces 结合使用。有关更多信息，请参阅the section called “用于 AWS 身份验证的 IAM 证书”。

有关 AWS CLI 中可用于 Amazon Keyspaces 的所有命令的完整列表，请参阅 AWS CLI 命令参考。

主题

下载和配置 AWS CLI

AWS CLI 在 https://aws.amazon.com/cli 上提供。它在 Windows、macOS 或 Linux 上运行。下载 AWS CLI 后，请按照以下步骤进行安装和配置：

1. 转到 AWS Command Line Interface 用户指南
2. 按照有关安装 AWS CLI 和配置 AWS CLI 的说明进行操作

将 AWS CLI 与 Amazon Keyspaces 结合使用

命令行格式包含 Amazon Keyspaces 操作名称，后跟该操作的参数。AWS CLI 支持参数值的速记语法以及 JSON。以下 Amazon Keyspaces 示例使用了 AWS CLI 速记语法。有关详细信息，请参阅将速记语法与 AWS CLI 结合使用。

以下命令可创建一个名为 catalog 的密钥空间。

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

该命令将在输出中返回资源的 Amazon 资源名称 (ARN)。

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

要确认密钥空间目录是否存在，您可以使用以下命令。

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

该命令将在输出中返回以下值。

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

以下命令可创建一个名为 book_awards 的表。该表的分区键由 `year` 和 `award` 两列组成，聚类键由 `category` 和 `rank` 两列组成，两个聚类列均使用升序排序。（为便于阅读，本部分中的长命令分行显示。）

```
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}]'
```

该命令的输出结果如下。

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

要确认表的元数据和属性，您可以使用以下命令。

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

此命令将返回以下输出。

```
{
    "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": ""
    }
}
```

创建具有复杂架构的表时，从 JSON 文件加载表的架构定义可能会有所帮助。下面是一个示例配置文件。从 schema_definition.zip 下载架构定义示例 JSON 文件并提取 schema_definition.json，记下文件路径。在本示例中，按机构定义 JSON 文件位于当前目录下。有关不同的文件路径选项，请参阅如何从文件加载参数。

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

以下示例展示了如何创建一个名为 myTable 的简单表（带有其他选项）。请注意，为提高可读性，命令被分成了多个独立的行。该命令显示了如何创建表以及如何执行以下操作:

- 设置表的容量模式
- 为表启用时间点恢复
- 将表的默认生存时间 (TTL) 值设为一年
- 为表添加两个标签

```
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'
```

本示例展示了如何创建一个使用客户托管密钥进行加密的新表，以及如何启用 TTL 以允许您为列和行设置过期日期。要运行此示例，您必须使用自己的密钥替换客户托管的 AWS KMS 密钥的资源 ARN，并确保 Amazon Keyspaces 可以访问该密钥。

```
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'
```

使用 API

您可以使用 AWS 和 AWS Command Line Interface (AWS CLI) 与 Amazon Keyspaces 进行交互式操作。您可以使用 API 进行数据语言定义 (DDL) 操作，如创建密钥空间或表。此外，您还可以使用基础设施即代码 (IaC) 服务和工具，如 AWS CloudFormation 和 Terraform。

您必先获取访问密钥 ID 和秘密访问密钥，然后才能将 AWS CLI 与 Amazon Keyspaces 结合使用。有关更多信息，请参阅the section called “用于 AWS 身份验证的 IAM 证书”。

有关 API 中可用于 Amazon Keyspaces 的所有操作的完整列表，请参阅 Amazon Keyspaces API 参考。

将 Amazon Keyspaces 与 SDK 配合使用 AWS

AWS 软件开发套件 (SDK) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
AWS SDK for C++	AWS SDK for C++ 代码示例
AWS SDK for Go	AWS SDK for Go 代码示例
AWS SDK for Java	AWS SDK for Java 代码示例
AWS SDK for JavaScript	AWS SDK for JavaScript 代码示例
AWS SDK for Kotlin	AWS SDK for Kotlin 代码示例

SDK 文档	代码示例
AWS SDK for .NET	AWS SDK for .NET 代码示例
AWS SDK for PHP	AWS SDK for PHP 代码示例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 代码示例
AWS SDK for Ruby	AWS SDK for Ruby 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 代码示例

示例可用性

找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

使用 Cassandra 客户端驱动程序以编程方式访问 Amazon Keyspaces

您可以使用许多第三方开源 Cassandra 驱动程序连接 Amazon Keyspaces。Amazon Keyspaces 与支持 Apache Cassandra 3.11.2 版本的 Cassandra 驱动程序兼容。以下是我们测试过并推荐用于 Amazon Keyspaces 的驱动程序和最新版本：

- `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`

有关 Cassandra 驱动程序的更多信息，请参阅 Apache Cassandra 客户端驱动程序。

> **Note**
>
> 为了帮助您入门，您可以查看和下载使用常用驱动程序与 Amazon Keyspaces 建立连接的 end-to-end 代码示例。请参阅上的 Amazon Keyspaces 示例。 GitHub

本章中的教程包括一个简单的 CQL 查询，用于确认已成功建立与 Amazon Keyspaces 的连接。要了解连接到 Amazon Keyspaces 端点后如何使用密钥空间和表，请参阅 CQL 语言参考。有关展示如何从 Amazon VPC 终端节点连接到 Amazon Keyspaces 的 step-by-step 教程，请参阅。the section called “连接 VPC 终端节点”

主题

- 使用 Cassandra 客户端驱动程序以编程方式访问 Amazon Keyspaces
- 使用 Cassandra Python 客户端驱动程序以编程方式访问 Amazon Keyspaces
- 使用 Cassandra Node.js 客户端驱动程序以编程方式访问 Amazon Keyspaces
- 使用 Cassandra .NET Core 客户端驱动程序以编程方式访问 Amazon Keyspaces
- 使用 Cassandra Go 客户端驱动程序以编程方式访问 Amazon Keyspaces
- 使用 Cassandra Perl 客户端驱动程序以编程方式访问 Amazon Keyspaces

使用 Cassandra 客户端驱动程序以编程方式访问 Amazon Keyspaces

本部分介绍了如何使用 Java 客户端驱动程序连接 Amazon Keyspaces。

> **Note**
>
> Java 17 和 DataStax Java 驱动程序 4.17 目前仅支持测试版。有关更多信息，请参阅 https://docs.datastax.com/en/developer/java-driver/4.17/upgrade_guide/。

要为用户和应用程序提供凭证，以通过编程方式访问 Amazon Keyspaces 资源，您可以执行以下任一操作：

- 创建与特定 AWS Identity and Access Management (IAM) 用户关联的服务特定凭证。
- 为了增强安全性，我们建议为所有 AWS 服务中使用的 IAM 身份创建 IAM 访问密钥。借助适用于 Cassandra 客户端驱动程序的 Amazon Keyspaces SigV4 身份验证插件，您可以使用 IAM 访问密

钥而不是用户名和密码来验证对 Amazon Keyspaces 的调用。有关更多信息，请参阅 the section called “用于 AWS 身份验证的 IAM 证书”。

> **Note**
>
> 有关如何在 Spring Boot 中使用 Amazon Keyspaces 的示例，请参阅。https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/spring

主题

- 开始前的准备工作
- 使用适用于 Apache Cassandra 的 DataStax Java 驱动程序使用特定于服务的凭证连接亚马逊密钥空间的tep-by-step 教程
- 使用适用于 Apache Cassandra 的 4.x DataStax Java 驱动程序和 Sigv4 身份验证插件连接亚马逊密钥空间的tep-by-step 教程
- 使用适用于 Apache Cassandra 的 3.x DataStax Java 驱动程序和 SigV4 身份验证插件连接亚马逊密钥空间

开始前的准备工作

要连接到 Amazon Keyspaces，您需要先完成以下任务。

1. Amazon Keyspaces 要求使用传输层安全性协议 (TLS) 来帮助保护与客户端的连接。

   a. 使用以下命令下载 Starfield 数字证书，并将 `sf-class2-root.crt` 保存在本地或您的主目录中。

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

   > **Note**
   >
   > 您还可以使用 Amazon 数字证书连接到 Amazon Keyspaces。如果您的客户端成功连接到 Amazon Keyspaces，您可以继续这样做。Starfield 证书为使用旧证书颁发机构的客户端提供了额外的向后兼容性。

   b. 将 Starfield 数字证书转换为 trustStore 文件：

```
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
```

在这一步中，您需要为密钥存储创建一个密码，并信任该证书。交互式命令如下所示。

```
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. 将 trustStore 文件附加到 JVM 参数中：

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

使用适用于 Apache Cassandra 的 DataStax Java 驱动程序使用特定于服务的凭证连接亚马逊密钥空间的tep-by-step 教程

以下 step-by-step 教程将引导您使用特定于服务的凭证使用适用于 Cassandra 的 Java 驱动程序连接到 Amazon Keyspaces。具体而言，你将使用适用于 Apache Cassandr DataStax a 的 Java 驱动程序的 4.0 版本。

主题

- 步骤 1：先决条件
- 步骤 2：配置驱动程序
- 步骤 3：运行示例应用程序

步骤 1：先决条件

要学习本教程，你需要生成特定于服务的凭证，并将适用于 Apache Cassandra 的 DataStax Java 驱动程序添加到你的 Java 项目中。

- 完成the section called “服务特定凭证”中的步骤，为您的 Amazon Keyspaces IAM 用户生成服务特定凭证。如果您更喜欢使用 IAM 访问密钥进行身份验证，请参阅the section called “Java 4.x 的身份验证插件”。
- 将 Apache Cassandra 的 DataStax Java 驱动程序添加到你的 Java 项目中。确保您使用的驱动程序版本支持 Apache Cassandra 3.11.2。有关更多信息，请参阅 A pache Cassandra 的 DataStax Java 驱动程序文档。

步骤 2：配置驱动程序

您可以通过为应用程序创建配置文件来指定 DataStax Java Cassandra 驱动程序的设置。此配置文件会覆盖默认设置，并告诉驱动程序使用端口 9142 连接到 Amazon Keyspaces 服务端点。有关可用服务端点的列表，请参阅 the section called “服务端点”。

创建配置文件，并将该文件保存在应用程序的资源文件夹中，例如 `src/main/resources/application.conf`。打开 `application.conf`，然后添加以下配置设置。

1. 身份验证提供程序：使用 类创建身份验证提供程序。*`ServiceUserName`*并*`ServicePassword`*应与您按照中的步骤生成服务专用凭证时获得的用户名和密码相匹配。生成服务特定凭证

   > **Note**
   >
   > 您可以使用适用于 Apache Cassandra 的 DataStax Java 驱动程序的身份验证插件，而不是在驱动程序配置文件中对凭据进行硬编码，从而使用短期证书。要了解更多信息，请按照the section called “Java 4.x 的身份验证插件”中的说明进行操作。

2. 本地数据中心：将 `local-datacenter` 的值设置为要连接的区域。例如，如果应用程序要连接到 `cassandra.us-east-2.amazonaws.com`，则将本地数据中心设置为 `us-east-2`。有关所有可用的 AWS 区域，请参阅 ???。设置 slow-replica-avoidance = false 以针对更少的节点进行负载均衡。
3. SSL/TLS — EngineFactory 通过在配置文件中添加一段来初始化 SSL，其中有一行用于指定类别。`class = DefaultSslEngineFactory`提供 trustStore 文件的路径和您之前创建的密码。Amazon Keyspaces 不支持对等设备的 `hostname-validation`，因此请将此选项设为 false。

```
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**
>
> 除了在配置文件中添加 trustStore 的路径外，您还可以直接在应用程序代码中添加 trustStore 的路径，或者在 JVM 参数中添加 trustStore 的路径。

步骤 3：运行示例应用程序

本代码示例展示了一个简单的命令行应用程序，它使用我们之前创建的配置文件创建了一个连接到 Amazon Keyspaces 的连接池。它通过运行一个简单的查询来确认连接已建立。

```
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**
>
> 使用 `try` 代码块建立连接，以确保连接始终处于关闭状态。如果不使用 `try` 代码块，请记得关闭连接，以免泄漏资源。

使用适用于 Apache Cassandra 的 4.x DataStax Java 驱动程序和 Sigv4 身份验证插件连接亚马逊密钥空间的tep-by-step 教程

以下部分介绍如何使用适用于 Apache Cassandra 的开源 4.x DataStax Java 驱动程序的 SigV4 身份验证插件来访问亚马逊密钥空间（适用于 Apache Cassandra）。该插件可从GitHub存储库中获得。

使用 SigV4 身份验证插件，您可以在连接 Amazon Keyspaces 时为用户或角色使用 IAM 凭据。该插件不需要用户名和密码，而是使用访问密钥签署 API 请求。有关更多信息，请参阅 the section called “用于 AWS 身份验证的 IAM 证书”。

步骤 1：先决条件

要学习本教程，您需要完成以下任务。

- 请按照the section called “用于 AWS 身份验证的 IAM 证书”中的步骤为 IAM 用户或角色创建凭证（如果您尚未创建）。本教程假设访问密钥将存储为环境变量。有关更多信息，请参阅 the section called “如何管理访问密钥”。
- 将 Apache Cassandra 的 DataStax Java 驱动程序添加到你的 Java 项目中。确保您使用的驱动程序版本支持 Apache Cassandra 3.11.2。有关更多信息，请参阅 A pache Cassandra 的 DataStax Java 驱动程序文档。
- 将身份验证插件添加到您的应用程序。该身份验证插件支持 Apache Cassandr DataStax a 的 Java 驱动程序 4.x 版本。如果您使用的是 Apache Maven 或可以使用 Maven 依赖关系的构建系统，请将以下依赖关系添加到您的 `pom.xml` 文件中。

> **Important**
>
> 将插件版本替换为GitHub 存储库中显示的最新版本。

```
<dependency>
        <groupId>software.aws.mcs</groupId>
        <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin</artifactId>
        <version>4.0.9</version>
</dependency>
```

步骤 2：配置驱动程序

您可以通过为应用程序创建配置文件来指定 DataStax Java Cassandra 驱动程序的设置。此配置文件会覆盖默认设置，并告诉驱动程序使用端口 9142 连接到 Amazon Keyspaces 服务端点。有关可用服务端点的列表，请参阅 the section called “服务端点”。

创建配置文件，并将该文件保存在应用程序的资源文件夹中，例如 `src/main/resources/application.conf`。打开 `application.conf`，然后添加以下配置设置。

1. 身份验证提供程序：将 `advanced.auth-provider.class`设置为 `software.aws.mcs.auth.SigV4AuthProvider` 的新实例。Sigv4 AuthProvider 是插件提供的用于执行 Sigv4 身份验证的身份验证处理程序。
2. 本地数据中心：将 `local-datacenter` 的值设置为要连接的区域。例如，如果应用程序要连接到 `cassandra.us-east-2.amazonaws.com`，则将本地数据中心设置为 `us-east-2`。有关所有可用的 AWS 区域，请参阅 ???。设置 slow-replica-avoidance = false 以针对更少的节点进行负载均衡。
3. SSL/TLS — EngineFactory 通过在配置文件中添加一段来初始化 SSL，其中有一行用于指定类别。`class = DefaultSslEngineFactory`提供 trustStore 文件的路径和您之前创建的密码。Amazon Keyspaces 不支持对等设备的 `hostname-validation`，因此请将此选项设为 false。

```
datastax-java-driver {
        basic.contact-points = ["cassandra.us-east-2.amazonaws.com:9142"]
        basic.load-balancing-policy {
            class = DefaultLoadBalancingPolicy
            local-datacenter = us-east-2
            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
            }
        }
    }
```

> **Note**
>
> 除了在配置文件中添加 trustStore 的路径外，您还可以直接在应用程序代码中添加 trustStore 的路径，或者在 JVM 参数中添加 trustStore 的路径。

步骤 3：运行应用程序

本代码示例展示了一个简单的命令行应用程序，它使用我们之前创建的配置文件创建了一个连接到 Amazon Keyspaces 的连接池。它通过运行一个简单的查询来确认连接已建立。

```
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**
>
> 使用 `try` 代码块建立连接，以确保连接始终处于关闭状态。如果不使用 `try` 代码块，请记得关闭连接，以免泄漏资源。

使用适用于 Apache Cassandra 的 3.x DataStax Java 驱动程序和 SigV4 身份验证插件连接亚马逊密钥空间

以下部分介绍如何使用适用于 Apache Cassandra 的 3.x 开源 DataStax Java 驱动程序的 Sigv4 身份验证插件来访问亚马逊密钥空间。该插件可从GitHub 存储库中获得。

使用 SigV4 身份验证插件，您可以在连接 Amazon Keyspaces 时为用户和角色使用 IAM 凭据。该插件不需要用户名和密码，而是使用访问密钥签署 API 请求。有关更多信息，请参阅 the section called “用于 AWS 身份验证的 IAM 证书”。

步骤 1：先决条件

要运行此代码示例，您首先需要完成以下任务。

- 按照the section called “用于 AWS 身份验证的 IAM 证书”中的步骤为 IAM 用户或角色创建凭证。本教程假设访问密钥将存储为环境变量。有关更多信息，请参阅 the section called “如何管理访问密钥”。
- 请按照the section called “开始前的准备工作”中的步骤下载 Starfield 数字证书，将其转换为 trustStore 文件，然后将 trustStore 文件附加到应用程序的 JVM 参数中。
- 将 Apache Cassandra 的 DataStax Java 驱动程序添加到你的 Java 项目中。确保您使用的驱动程序版本支持 Apache Cassandra 3.11.2。有关更多信息，请参阅 A pache Cassandra 的 DataStax Java 驱动程序文档。
- 将身份验证插件添加到您的应用程序。该身份验证插件支持 Apache Cassandr DataStax a 的 Java 驱动程序版本 3.x。如果您使用的是 Apache Maven 或可以使用 Maven 依赖关系的构建系统，请将以下依赖关系添加到您的 `pom.xml` 文件中。将插件版本替换为GitHub 存储库中显示的最新版本。

```
<dependency>
        <groupId>software.aws.mcs</groupId>
        <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin_3</artifactId>
        <version>3.0.3</version>
</dependency>
```

步骤 2：运行应用程序

本代码示例展示了一个简单的命令行应用程序，它创建了一个连接到 Amazon Keyspaces 的连接池。它通过运行一个简单的查询来确认连接已建立。

```
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"));
    }
}
```

使用说明：

有关可用端点的列表，请参阅the section called “服务端点”。

请参阅以下存储库，获取实用的 Java 驱动程序策略、示例，以及结合使用 Java 驱动程序和 Amazon Keyspaces 的最佳实践，：https://github.com/aws-samples/amazon-keyspaces-java-driver-helpers。

使用 Cassandra Python 客户端驱动程序以编程方式访问 Amazon Keyspaces

在本部分中，我们介绍了如何使用 Python 客户端驱动程序连接 Amazon Keyspaces。要为用户和应用程序提供凭证，以通过编程方式访问 Amazon Keyspaces 资源，您可以执行以下任一操作：

- 创建与特定 AWS Identity and Access Management (IAM) 用户关联的服务特定凭证。
- 为了增强安全性，我们建议为所有 AWS 服务中使用的 IAM 用户或角色创建 IAM 访问密钥。借助适用于 Cassandra 客户端驱动程序的 Amazon Keyspaces SigV4 身份验证插件，您可以使用 IAM 访问密钥而不是用户名和密码来验证对 Amazon Keyspaces 的调用。有关更多信息，请参阅 the section called “用于 AWS 身份验证的 IAM 证书”。

主题

- 开始前的准备工作
- 使用适用于 Apache Cassandra 的 Python 驱动程序和服务特定凭证连接 Amazon Keyspaces
- 使用适用于 Apache Cassandra 的 Pyt DataStax hon 驱动程序和 SigV4 身份验证插件连接到亚马逊密钥空间

开始前的准备工作

在开始之前，您需要完成以下任务。

Amazon Keyspaces 要求使用传输层安全性协议 (TLS) 来帮助保护与客户端的连接。要使用 TLS 连接到 Amazon Keyspaces，您需要下载 Amazon 数字证书，并将 Python 驱动程序配置为使用 TLS。

使用以下命令下载 Starfield 数字证书，并将 `sf-class2-root.crt` 保存在本地或您的主目录中。

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

> **Note**
>
> 您还可以使用 Amazon 数字证书连接到 Amazon Keyspaces。如果您的客户端成功连接到 Amazon Keyspaces，您可以继续这样做。Starfield 证书为使用旧证书颁发机构的客户端提供了额外的向后兼容性。

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

使用适用于 Apache Cassandra 的 Python 驱动程序和服务特定凭证连接 Amazon Keyspaces

以下代码示例向您展示了如何使用 Python 客户端驱动程序和服务特定凭证连接 Amazon Keyspaces。

```
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)
```

使用说明：

1. 将 "*path_to_file*/sf-class2-root.crt" 替换为第一步中保存的证书的路径。
2. 按照以下步骤操作，确保和与您在生成服务特定凭证时获得的用户名和密码*ServicePassword*相匹配。*ServiceUserName*生成服务特定凭证
3. 有关可用端点的列表，请参阅the section called “服务端点”。

使用适用于 Apache Cassandra 的 Pyt DataStax hon 驱动程序和 SigV4 身份验证插件连接到亚马逊密钥空间

以下部分介绍如何使用适用于 Apache Cassandra 的开源 Pyth DataStax on 驱动程序的 Sigv4 身份验证插件来访问亚马逊密钥空间（适用于 Apache Cassandra）。

请先按照the section called “用于 AWS 身份验证的 IAM 证书”中的步骤为 IAM 角色创建凭证（如果您尚未创建）。本教程使用临时凭证，需要一个 IAM 角色。有关临时凭证的更多信息，请参阅the section called “使用临时凭证连接 Amazon Keyspaces”。

然后，将 Python Sigv4 身份验证插件从存储库添加到您的环境中。GitHub

```
pip install cassandra-sigv4
```

以下代码示例展示了如何使用适用于 Cassandra 的开源 Python 驱动程序和 SigV4 身份验证插件连接到 Amaz DataStax on Keyspaces。该插件依赖于 Python 的 AWS SDK (Boto3)。它使用 `boto3.session` 获取临时凭证。

```
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)
```

使用说明：

1. 将 "*path_to_file*/sf-class2-root.crt" 替换为第一步中保存的证书的路径。
2. 确保 *aws_access_key_id*、*aws_secret_access_key* 和 *aws_session_token* 与使用 boto3.session 获取的Access Key、Secret Access Key和Session Token相匹配。有关详细信息，请参阅AWS SDK for Python (Boto3) 中的凭证。
3. 有关可用端点的列表，请参阅the section called “服务端点”。

使用 Cassandra Node.js 客户端驱动程序以编程方式访问 Amazon Keyspaces

本部分介绍了如何使用 Node.js 客户端驱动程序连接 Amazon Keyspaces。要为用户和应用程序提供凭证，以通过编程方式访问 Amazon Keyspaces 资源，您可以执行以下任一操作：

- 创建与特定 AWS Identity and Access Management (IAM) 用户关联的服务特定凭证。
- 为了增强安全性，我们建议为所有 AWS 服务中使用的 IAM 用户或角色创建 IAM 访问密钥。借助适用于 Cassandra 客户端驱动程序的 Amazon Keyspaces SigV4 身份验证插件，您可以使用 IAM 访问密钥而不是用户名和密码来验证对 Amazon Keyspaces 的调用。有关更多信息，请参阅 the section called “用于 AWS 身份验证的 IAM 证书”。

主题

- 开始前的准备工作
- 使用适用于 Apache Cassandra 的 Node.js DataStax 驱动程序和特定于服务的凭证连接到亚马逊密钥空间
- 使用适用于 Apache Cassandra DataStax 的 Node.js 驱动程序和 SigV4 身份验证插件连接到亚马逊密钥空间

开始前的准备工作

在开始之前，您需要完成以下任务。

Amazon Keyspaces 要求使用传输层安全性协议 (TLS) 来帮助保护与客户端的连接。要使用 TLS 连接到 Amazon Keyspaces，您需要下载 Amazon 数字证书，并将 Python 驱动程序配置为使用 TLS。

使用以下命令下载 Starfield 数字证书，并将 `sf-class2-root.crt` 保存在本地或您的主目录中。

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

> **Note**
>
> 您还可以使用 Amazon 数字证书连接到 Amazon Keyspaces。如果您的客户端成功连接到 Amazon Keyspaces，您可以继续这样做。Starfield 证书为使用旧证书颁发机构的客户端提供了额外的向后兼容性。


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


使用适用于 Apache Cassandra 的 Node.js DataStax 驱动程序和特定于服务的凭证连接到亚马逊密钥空间

将您的驱动程序配置为使用 TLS 的 Starfield 数字证书，并使用服务特定凭证进行身份验证。例如：

```
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}`));
```

使用说明：

1. 将 "*path_to_file*/sf-class2-root.crt" 替换为第一步中保存的证书的路径。
2. 按照以下步骤操作，确保和与您在生成服务特定凭证时获得的用户名和密码*ServicePassword*相匹配。*ServiceUserName*生成服务特定凭证
3. 有关可用端点的列表，请参阅the section called “服务端点”。

使用适用于 Apache Cassandra DataStax 的 Node.js 驱动程序和 SigV4 身份验证插件连接到亚马逊密钥空间

以下部分介绍如何使用适用于 Apache Cassandra 的开源 DataStax Node.js 驱动程序的 Sigv4 身份验证插件来访问亚马逊密钥空间（适用于 Apache Cassandra）。

请按照the section called “用于 AWS 身份验证的 IAM 证书”中的步骤为 IAM 用户或角色创建凭证（如果您尚未创建）。

将 Node.js Sigv4 身份验证插件从存储库添加到您的应用程序中。GitHub 该插件支持 Cassandra 的 DataStax Node.js 驱动程序版本 4.x，并且依赖于 Node.js 的 AWS SDK。它使用 `AWSCredentialsProvider` 获取凭证。

```
$ npm install aws-sigv4-auth-cassandra-plugin --save
```

本代码示例展示了如何将 `SigV4AuthProvider` 的区域特定实例设置为身份验证提供商。

```
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}`));
```

使用说明：

1. 将 "*path_to_file*/sf-class2-root.crt" 替换为第一步中保存的证书的路径。
2. 确保*accessKeyId*和*secretAccessKey*与您使用获得的访问密钥和私有访问密钥相匹配AWSCredentialsProvider。有关更多信息，请参阅 Node.js 中的AWS SDK 中在 Node.js JavaScript 中设置凭证。
3. 要在代码之外存储访问密钥，请参阅the section called “如何管理访问密钥”中的最佳实践。
4. 有关可用端点的列表，请参阅the section called “服务端点”。

使用 Cassandra .NET Core 客户端驱动程序以编程方式访问 Amazon Keyspaces

本部分介绍了如何使用 .NET Core 客户端驱动程序连接 Amazon Keyspaces。设置步骤因环境和操作系统而异，您可能需要相应地进行修改。Amazon Keyspaces 要求使用传输层安全性协议 (TLS) 来帮助保护与客户端的连接。要使用 TLS 连接 Amazon Keyspaces，您需要下载 Starfield 数字证书，并将驱动程序配置为使用 TLS。

1. 下载 Starfield 证书并将其保存到本地目录，同时记下路径。以下是使用的示例 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. 使用 nuget 控制台SharpDriver 通过 nuget 安装 CassanDrac。

```
PM> Install-Package CassandraCSharpDriver
```

3. 以下示例使用.NET Core C# 控制台项目连接到 Amazon Keyspaces 并运行查询。

```
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);
            }
        }
    }
}
```

使用说明：

a. 将 "*path_to_file*/sf-class2-root.crt" 替换为第一步中保存的证书的路径。

b. 按照以下步骤操作，确保和与您在生成服务特定凭证时获得的用户名和密码*ServicePassword*相匹配。*ServiceUserName*生成服务特定凭证

c. 有关可用端点的列表，请参阅the section called “服务端点”。

使用 Cassandra Go 客户端驱动程序以编程方式访问 Amazon Keyspaces

本部分介绍了如何使用 Go 客户端驱动程序连接 Amazon Keyspaces。要为用户和应用程序提供凭证，以通过编程方式访问 Amazon Keyspaces 资源，您可以执行以下任一操作：

- 创建与特定 AWS Identity and Access Management (IAM) 用户关联的服务特定凭证。
- 为了增强安全性，我们建议为所有 AWS 服务中使用的 IAM 用户和角色创建 IAM 访问密钥。借助适用于 Cassandra 客户端驱动程序的 Amazon Keyspaces SigV4 身份验证插件，您可以使用 IAM 访问密钥而不是用户名和密码来验证对 Amazon Keyspaces 的调用。有关更多信息，请参阅 the section called “用于 AWS 身份验证的 IAM 证书”。

主题

- 开始前的准备工作
- 使用适用于 Apache Cassandra 的 Gocql 驱动程序和服务特定凭证连接 Amazon Keyspaces
- 使用适用于 Apache Cassandra 的 Go 驱动程序和 SigV4 身份验证插件连接 Amazon Keyspaces

开始前的准备工作

在开始之前，您需要完成以下任务。

Amazon Keyspaces 要求使用传输层安全性协议 (TLS) 来帮助保护与客户端的连接。要使用 TLS 连接到 Amazon Keyspaces，您需要下载 Amazon 数字证书，并将 Python 驱动程序配置为使用 TLS。

使用以下命令下载 Starfield 数字证书，并将 `sf-class2-root.crt` 保存在本地或您的主目录中。

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

> **Note**
>
> 您还可以使用 Amazon 数字证书连接到 Amazon Keyspaces。如果您的客户端成功连接到 Amazon Keyspaces，您可以继续这样做。Starfield 证书为使用旧证书颁发机构的客户端提供了额外的向后兼容性。


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


使用适用于 Apache Cassandra 的 Gocql 驱动程序和服务特定凭证连接 Amazon Keyspaces

1. 为您的应用程序创建一个目录。

   ```
   mkdir ./gocqlexample
   ```

2. 导航到新目录。

   ```
   cd gocqlexample
   ```

3. 为应用程序创建一个文件。

   ```
   touch cqlapp.go
   ```

4. 下载 Go 驱动程序。

   ```
   go get github.com/gocql/gocql
   ```

5. 将以下示例代码添加到 cqlapp.go 文件。

   ```
   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()
}
```

使用说明：

a. 将 "*path_to_file*/sf-class2-root.crt" 替换为第一步中保存的证书的路径。

b. 按照以下步骤操作，确保和与您在生成服务专用凭证时获得的用户名和密码*ServicePassword*相匹配。*ServiceUserName*生成服务特定凭证

c. 有关可用端点的列表，请参阅the section called “服务端点”。

6. 构建程序。

```
go build cqlapp.go
```

7. 运行程序。

```
./cqlapp
```

使用适用于 Apache Cassandra 的 Go 驱动程序和 SigV4 身份验证插件连接 Amazon Keyspaces

以下代码示例展示了如何使用开源 Go 驱动程序的 SigV4 身份验证插件访问 Amazon Keyspaces（Apache Cassandra 兼容）。

请按照the section called “用于 AWS 身份验证的 IAM 证书”中的步骤为 IAM 用户或角色创建凭证（如果您尚未创建）。

将 Go Sigv4 身份验证插件从存储库添加到您的应用程序中。GitHub该插件支持适用于 Cassandra 的开源 Go 驱动程序 1.2.x 版本，并且依赖于 Go 的 AWS SDK。

```
$ go mod init
$ go get github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin
```

在此代码示例中，Amazon Keyspaces 端点由 `Cluster` 类表示。它使用集群的身份验证器属性 `AwsAuthenticator` 来获取凭证。

```
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)
    }
}
```

使用说明：

1. 将 "*path_to_file*/sf-class2-root.crt" 替换为第一步中保存的证书的路径。
2. 确保*AccessKeyId*和*SecretAccessKey*与您使用获得的访问密钥和私有访问密钥相匹配AwsAuthenticator。有关详细信息，请参阅AWS SDK for Go 中的配置适用于 Go 的 AWS SDK。
3. 要在代码之外存储访问密钥，请参阅the section called “如何管理访问密钥”中的最佳实践。
4. 有关可用端点的列表，请参阅the section called “服务端点”。

使用 Cassandra Perl 客户端驱动程序以编程方式访问 Amazon Keyspaces

本部分介绍了如何使用 Perl 客户端驱动程序连接 Amazon Keyspaces。在本代码示例中，我们使用的是 Perl 5。Amazon Keyspaces 要求使用传输层安全性协议 (TLS) 来帮助保护与客户端的连接。

> **Important**
>
> 为了创建安全连接，我们的代码示例会在建立 TLS 连接之前使用 Starfield 数字证书对服务器进行身份验证。Perl 驱动程序不会验证服务器的 Amazon SSL 证书，这意味着无法确认您是否能连接到 Amazon Keyspaces。第二步，务必要在连接 Amazon Keyspaces 时配置驱动程序以使用 TLS，这可以确保在客户端和服务器之间传输的数据经过加密。

1. 从 https://metacpan.org/pod/DBD::Cassandra 下载 Cassandra DBI 驱动程序，并将其安装到您的 Perl 环境中。具体步骤取决于您的环境。下面是一个常见示例。

```
cpanm DBD::Cassandra
```

2. 为应用程序创建一个文件。

```
touch cqlapp.pl
```

3. 将以下示例代码添加到 cqlapp.pl 文件。

```
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**
>
> 按照以下步骤操作，确保和与您在生成服务专用凭证时获得的用户名和密码*ServicePassword*相匹配。*ServiceUserName*生成服务特定凭证

> **Note**
>
> 有关可用端点的列表，请参阅the section called “服务端点”。

4. 运行应用程序。

```
perl cqlapp.pl
```

教程：从亚马逊 Elastic Kubernetes Service 连接到亚马逊密钥空间 Amazon Kubernetes Service

本教程将引导您完成设置亚马逊弹性 Kubernetes Service（Amazon EKS）集群以托管使用 Sigv4 身份验证连接到亚马逊密钥空间的容器化应用程序所需的步骤。

Amazon EKS 是一项托管服务，无需安装、操作和维护您自己的 Kubernetes 控制平面。Kubernetes 是一个开源系统，可以实现容器化应用程序的管理、扩缩和部署自动化。

本教程提供了 step-by-step 有关配置、构建容器化 Java 应用程序并将其部署到 Amazon EKS 的指导。在最后一步中，您将运行应用程序将数据写入 Amazon Keyspaces 表。

主题

- 教程的先决条件
- 第 1 步：配置 Amazon EKS 集群并设置 IAM 权限
- 步骤 2：配置应用程序
- 第 3 步：创建应用程序映像并将 Docker 文件上传到您的 Amazon ECR 存储库
- 第 4 步：将应用程序部署到 Amazon EKS 并将数据写入您的 Amazon Keyspaces 表
- 第 5 步：（可选）清理

教程的先决条件

在开始学习本教程之前，请先创建以下 AWS 资源

1. 在开始本教程之前，请按照中的 AWS 设置说明进行操作访问 Amazon Keyspaces（Apache Cassandra 兼容）。这些步骤包括注册 AWS 和创建有权访问 Amazon Keyspaces 的 AWS Identity and Access Management (IAM) 委托人。
2. 在本教程的后面部分中，使用名称创建一个 Amazon Keyspaces 密钥空间`aws`和一个名称为的表`user`，您可以从在 Amazon EKS 中运行的容器化应用程序中写入该表格。您可以使用 AWS CLI 或来执行此操作`cqlsh`。

   AWS CLI

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

   要确认密钥空间已创建，可以使用以下命令。

   ```
   aws keyspaces list-keyspaces
   ```

   要创建表，可以使用以下命令。

   ```
   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}]'
   ```

   要确认您的表已创建，可以使用以下命令。

   ```
   aws keyspaces list-tables --keyspace-name 'aws'
   ```

   有关更多信息，请参阅《AWS CLI 命令参考》中的创建密钥空间和创建表。

   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
);
```

要验证您的表是否已创建，可以使用以下语句。

```
SELECT * FROM system_schema.tables WHERE keyspace_name = "aws";
```

您的表应在此语句的输出中列出。请注意，创建表之前可能会有一段延迟。有关更多信息，请参阅the section called “CREATE TABLE”。

3. 创建一个 Fargate-Linux 节点类型的 Amazon EKS 集群。Fargate 是一款无服务器计算引擎，允许您在不管理亚马逊 EC2 实例的情况下部署 Kubernetes Pod。要学习本教程而不必更新所有示例命令中的集群名称，请按照 Amazon EKS 用户指南中的 Amazon EKS 入门`eksctl`中的说明创建一个集群。`my-eks-cluster`创建集群后，请验证您的节点和两个默认 Pod 是否运行且运行正常。您可以使用以下命令执行此操作。

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

您应该会看到类似于此输出的内容。

```
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. 安装 Docker。有关如何在亚马逊 EC2 实例上安装 Docker 的说明，请参阅亚马逊弹性容器注册表用户指南中的安装 Docker。

   Docker 适用于许多不同的操作系统，包括大多数现代 Linux 分发版 (如 Ubuntu) 甚至 MacOS 和 Windows。有关如何在特定的操作系统上安装 Docker 的更多信息，请转到 Docker 安装指南。

5. 创建 Amazon ECR 存储库。Amazon ECR 是一项 AWS 托管容器镜像注册服务，您可以将其与首选 CLI 配合使用，以推送、拉取和管理 Docker 镜像。有关 Amazon ECR 存储库的更多信

息，请参阅亚马逊弹性容器注册表用户指南。您可以使用以下命令创建名为的存储库my-ecr-repository。

```
aws ecr create-repository --repository-name my-ecr-repository
```

完成先决条件步骤后，继续执行the section called “步骤 1：配置 Amazon EKS 集群”。

第 1 步：配置 Amazon EKS 集群并设置 IAM 权限

配置 Amazon EKS 集群并创建允许 Amazon EKS 服务账户连接到您的 Amazon Keyspaces 表所需的 IAM 资源

1. 为 Amazon EKS 集群创建 Open ID Connect (OIDC) 提供商。这是为服务账户使用 IAM 角色所必需的。有关 OIDC 提供商以及如何创建它们的更多信息，请参阅 A ma zon EKS 用户指南中的为您的集群创建 IAM OIDC 提供商。

   a. 使用以下命令为您的集群创建 IAM OIDC 身份提供商。此示例假设您的集群名称为my-eks-cluster。如果您的集群名称不同，请记住在所有 future 命令中更新名称。

   ```
   eksctl utils associate-iam-oidc-provider --cluster my-eks-cluster --approve
   ```

   b. 使用以下命令确认 OIDC 身份提供商已在 IAM 中注册。

   ```
   aws iam list-open-id-connect-providers --region aws-region
   ```

   输出应如下所示：记下 OIDC 的 Amazon 资源名称 (ARN)，下一步为服务账户创建信任策略时需要该名称。

   ```
   {
       "OpenIDConnectProviderList": [
           ..
           {
               "Arn": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.aws-
   region.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
           }
       ]
   }
   ```

2. 为 Amazon EKS 集群创建一个服务账户。服务帐号为 Pod 中运行的进程提供身份。Pod 是最小、最简单的 Kubernetes 对象，你可以用它来部署容器化应用程序。接下来，创建一个 IAM 角色，服务账号可以代入该角色以获取资源权限。您可以从 Pod 访问任何 AWS 服务，该服务账号已配置为使用服务账号，该账号可以代入具有该服务访问权限的 IAM 角色。

   a. 为服务帐号创建新的命名空间。命名空间有助于隔离为本教程创建的集群资源。您可以使用以下命令创建新的命名空间。

   ```
   kubectl create namespace my-eks-namespace
   ```

   b. 要使用自定义命名空间，必须将其与 Fargate 配置文件关联。下面是一个代码示例。

   ```
   eksctl create fargateprofile \
       --cluster my-eks-cluster \
       --name my-fargate-profile \
       --namespace my-eks-namespace \
       --labels *=*
   ```

   c. 使用以下命令`my-eks-serviceaccount`在 Amazon EKS 集群`my-eks-namespace`的命名空间中创建一个名称为的服务账户。

   ```
   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. 运行以下命令创建指示 IAM 角色信任您的服务账号的信任策略文件。委托人必须先建立这种信任关系，然后才能担任职务。您需要对文件进行以下编辑：

   - 对于`Principal`，输入 IAM 返回给命令的 ARN。`list-open-id-connect-providers`ARN 包含您的账号和地区。
   - 在`condition`语句中，替换 AWS 区域 和 OIDC ID。
   - 确认服务帐号名称和命名空间是否正确。

   下一步创建 IAM 角色时，您需要附加信任策略文件。

```
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
```

可选：您还可以在StringEquals或StringLike条件中添加多个条目，以允许多个服务帐号或命名空间代入该角色。要允许您的服务账户在其他 AWS 账户中担任 IAM 角色，请参阅 A mazon EKS 用户指南中的跨账户 IAM 权限。

3. 创建一个名为 Amazon EKS 服务账户my-iam-role的 IAM 角色，该角色的名称将由该账户代替。将上一步中创建的信任策略文件附加到该角色。信任策略指定了 IAM 角色可以信任的服务账号和 OIDC 提供商。

```
aws iam create-role --role-name my-iam-role --assume-role-policy-document file://
trust-relationship.json --description "EKS service account role"
```

4. 通过附加访问策略将 IAM 角色权限分配给 Amazon Keyspaces。

   a. 附加访问策略以定义 IAM 角色可以对特定 Amazon Keyspaces 资源执行的操作。在本教程中，我们使用 AWS 托管策略AmazonKeyspacesFullAccess，因为我们的应用程序将向您的 Amazon Keyspaces 表中写入数据。但是，作为最佳实践，建议创建实现最低权限原则的

自定义访问策略。有关更多信息，请参阅the section called “Amazon Keyspaces 如何与 IAM 配合使用”。

```
aws iam attach-role-policy --role-name my-iam-role --policy-
arn=arn:aws:iam::aws:policy/AmazonKeyspacesFullAccess
```

使用以下语句确认策略已成功关联到 IAM 角色。

```
aws iam list-attached-role-policies --role-name my-iam-role
```

输出应如下所示：

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonKeyspacesFullAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonKeyspacesFullAccess"
        }
    ]
}
```

b. 使用服务账户可以代入的 IAM 角色的 Amazon 资源名称 (ARN) 对其进行注释。请务必使用您的账户 ID 更新角色 ARN。

```
kubectl annotate serviceaccount -n my-eks-namespace my-eks-serviceaccount
 eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/my-iam-role
```

5. 确认 IAM 角色和服务账号配置正确。

a. 使用以下语句确认 IAM 角色的信任策略配置正确。

```
aws iam get-role --role-name my-iam-role --query Role.AssumeRolePolicyDocument
```

输出应如下所示：

```
{
    "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. 确认 Amazon EKS 服务账户已使用 IAM 角色进行注释。

```
kubectl describe serviceaccount my-eks-serviceaccount -n my-eks-namespace
```

输出应如下所示：

```
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>
[...]
```

创建 Amazon EKS 服务账户、IAM 角色并配置所需的关系和权限后，请继续the section called “步骤 2：配置应用程序”。

步骤 2：配置应用程序

在此步骤中，您将使用 Sigv4 插件构建连接到 Amazon Keyspaces 的应用程序。你可以从 Github 上的 Amazon Keyspaces 示例代码存储库中查看和下载示例 Java 应用程序。或者，您可以使用自己的应用程序继续操作，确保完成所有配置步骤。

配置您的应用程序并添加所需的依赖关系。

1. 您可以使用以下命令克隆 Github 存储库来下载示例 Java 应用程序。

```
git clone https://github.com/aws-samples/amazon-keyspaces-examples.git
```

2. 下载 Github 存储库后，解压缩下载的文件并导航到该文件的`resources`目录。`application.conf`

   a. 应用程序配置

   在此步骤中，您将配置 SigV4 身份验证插件。您可以在应用程序中使用以下示例。如果您尚未这样做，则需要生成 IAM 访问密钥（访问密钥 ID 和私有访问密钥），并将其保存在 AWS 配置文件中或作为环境变量保存。有关详细说明，请参阅 the section called “ AWS 身份验证所需的凭据”。根据需要更新 Amazon Keyspaces 的 AWS 区域和服务终端节点。有关更多服务终端节点，请参阅the section called “服务端点”。用您自己的信任库位置、信任库名称和信任库密码替换。

```
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. 添加 STS 模块依赖关系。

这增加了使用返回应用程序需要提供的 AWS 证书的功能，以便服务账户可以担任 IAM 角色。WebIdentityTokenCredentialsProvider您可以根据以下示例执行此操作。

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

c. 添加 Sigv4 依赖关系。

此软件包实现了向 Amazon Keyspaces 进行身份验证所需的 SigV4 身份验证插件

```
        <dependency>
            <groupId>software.aws.mcs</groupId>
            <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin</
artifactId>
            <version>4.0.3</version>
        </dependency>
```

3. 添加日志依赖关系。

没有日志，就无法对连接问题进行故障排除。在本教程中，我们使用slf4j作为日志框架，并logback.xml用于存储日志输出。我们将日志级别设置为debug以建立连接。您可以使用以下示例来添加依赖关系。

```
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>2.0.5</version>
        </dependency>
```

您可以使用以下代码片段来配置日志记录。

```
<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**
>
> 该debug级别是调查连接失败所必需的。通过应用程序成功连接到 Amazon Keyspaces 后，您可以根据需要将日志级别更改warning为info或。

第 3 步：创建应用程序映像并将 Docker 文件上传到您的 Amazon ECR 存储库

在此步骤中，您将编译示例应用程序，构建 Docker 映像，然后将该映像推送到您的 Amazon ECR 存储库。

构建您的应用程序，构建 Docker 镜像，然后将其提交到 Amazon 弹性容器注册表

1. 为版本设置用于定义您的环境变量 AWS 区域。将示例中的区域替换为您自己的区域。

```
export CASSANDRA_HOST=cassandra.aws-region.amazonaws.com:9142
export CASSANDRA_DC=aws-region
```

2. 使用以下命令使用 Apache Maven 3.6.3 或更高版本编译应用程序。

```
mvn clean install
```

   这将创建一个包含所有依赖项的JARtarget文件。

3. 使用以下命令检索下一步所需的 ECR 存储库 URI。请务必将该区域更新为你一直在使用的区域。

```
aws ecr describe-repositories --region aws-region
```

   输出应如以下示例所示。

```
"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. 使用上一步中的存储库 URI 从应用程序的根目录构建 Docker 镜像。根据需要修改 Docker 文件。在构建命令中，请务必替换您的账户 ID，并将设置为 Amazon ECR 存储库`my-ecr-repository`所在的区域。 AWS 区域

```
docker build -t 111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-
repository:latest .
```

5. 检索身份验证令牌将 Docker 镜像推送到 Amazon ECR。您可以使用以下命令执行此操作。

```
aws ecr get-login-password --region aws-region | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.aws-region.amazonaws.com
```

6. 首先，检查您的 Amazon ECR 存储库中是否有现有图像。您可使用以下命令。

```
aws ecr describe-images --repository-name my-ecr-repository --region aws-region
```

然后，将 Docker 镜像推送到存储库。您可使用以下命令。

```
docker push 111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-repository:latest
```

第 4 步：将应用程序部署到 Amazon EKS 并将数据写入您的 Amazon Keyspaces 表

在本教程的这一步中，您将为您的应用程序配置 Amazon EKS 部署，并确认应用程序正在运行并且可以连接到 Amazon Keyspaces。

要将应用程序部署到 Amazon EKS，您需要在名为的文件中配置所有相关设置deployment.yaml。然后，Amazon EKS 将使用此文件来部署应用程序。文件中的元数据应包含以下信息：

- 应用程序名称应用程序的名称。在本教程中，我们使用my-keyspaces-app。
- Kubernetes 命名空间是 Amazon EKS 集群的命名空间。在本教程中，我们使用my-eks-namespace。
- 亚马逊 EKS 服务账户命名亚马逊 EKS 服务账户的名称。在本教程中，我们使用my-eks-serviceaccount。
- 图像名称应用程序映像的名称。在本教程中，我们使用my-keyspaces-app。
- 图片 URI 来自亚马逊 ECR 的 Docker 镜像 URI。
- AWS 账户 ID 您的 AWS 账户 ID。
- IAM 角色 ARN 为服务账户创建的 IAM 角色的 ARN。在本教程中，我们使用my-iam-role。
- AWS 区域 在 AWS 区域 你创建 Amazon EKS 集群的 Amazon EKS 集群中。

在此步骤中，您将部署并运行连接到 Amazon Keyspaces 并将数据写入表的应用程序。

1. 配置 deployment.yaml 文件。您需要替换以下值：

   - name
   - namespace
   - serviceAccountName
   - image
   - AWS_ROLE_ARN value
   - The AWS 区域 in CASSANDRA_HOST
   - AWS_REGION

   你可以使用以下文件作为示例。

```
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. 部署 deployment.yaml。

```
kubectl apply -f deployment.yaml
```

输出应如下所示：

```
deployment.apps/my-keyspaces-app created
```

3. 检查您的 Amazon EKS 集群命名空间中 Pod 的状态。

```
kubectl get pods -n my-eks-namespace
```

输出应类似于以下示例：

```
NAME                     READY STATUS RESTARTS AGE
my-keyspaces-app-123abcde4f-g5hij 1/1 Running 0 75s
```

要了解更多详细信息，您可以使用以下命令。

```
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. 查看 Pod 的日志，确认您的应用程序正在运行并且可以连接到您的 Amazon Keyspaces 表。您可以使用以下命令执行此操作。请务必替换您的部署名称。

```
kubectl logs -f my-keyspaces-app-123abcde4f-g5hij -n my-eks-namespace
```

您应该能够看到确认与 Amazon Keyspaces 连接的应用程序日志条目，如下例所示。

```
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. 对您的 Amazon Keyspaces 表运行以下 CQL 查询，以确认您的表中已写入一行数据：

```
SELECT * from aws.user;
```

您应看到以下输出：

```
fname    | lname | username | last_update_date
----------+-------+----------+---------------------------
random    | k     | test     | 2023-12-07 13:58:31.57+0000
```

第 5 步：（可选）清理

按照以下步骤删除本教程中创建的所有资源。

移除在本教程中创建的资源

1. 删除您的部署。您可以使用以下命令来执行此操作。

```
kubectl delete deployment my-keyspaces-app -n my-eks-namespace
```

2. 删除 Amazon EKS 集群和其中包含的所有 Pod。这还会删除相关资源，例如服务帐号和 OIDC 身份提供商。您可以使用以下命令来执行此操作。

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

3. 删除用于具有亚马逊密钥空间访问权限的 Amazon EKS 服务账户的 IAM 角色。首先，您必须删除附加到该角色的托管策略。

```
aws iam detach-role-policy --role-name my-iam-role --policy-arn
 arn:aws:iam::aws:policy/AmazonKeyspacesFullAccess
```

然后，您可以使用以下命令删除该角色。

```
aws iam delete-role --role-name my-iam-role
```

有关更多信息，请参阅 IAM 用户指南中的删除 IAM 角色 (AWS CLI)。

4. 删除 Amazon ECR 存储库，包括存储在其中的所有图像。您可以使用以下命令执行此操作。

```
aws ecr delete-repository \
      --repository-name my-ecr-repository \
      --force \
      --region aws-region
```

请注意，删除包含图像的存储库需要使用该force标志。要先删除您的图片，您可以使用以下命令执行此操作。

```
aws ecr batch-delete-image \
      --repository-name my-ecr-repository \
      --image-ids imageTag=latest \
```

```
    --region aws-region
```

有关更多信息，请参阅 Amazon 弹性容器注册表用户指南中的删除镜像。

5. 删除 Amazon Keyspaces 密钥空间和表。删除密钥空间会自动删除该密钥空间中的所有表。您可以使用以下选项之一来执行此操作。

AWS CLI

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

要确认密钥空间已删除，可以使用以下命令。

```
aws keyspaces list-keyspaces
```

要先删除表，可以使用以下命令。

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

要确认您的表已删除，可以使用以下命令。

```
aws keyspaces list-tables --keyspace-name 'aws'
```

有关更多信息，请参阅《AWS CLI 命令参考》中的删除密钥空间和删除表。

cqlsh

```
DROP KEYSPACE IF EXISTS "aws";
```

要验证您的密钥空间是否已删除，可以使用以下语句。

```
SELECT * FROM system_schema.keyspaces ;
```

您的密钥空间不应在此语句的输出中列出。请注意，在删除密钥空间之前，可能会有一段延迟。有关更多信息，请参阅the section called “DROP KEYSPACE”。

要先删除表，可以使用以下命令。

```
DROP TABLE "aws.user"
```

要确认您的表已删除，可以使用以下命令。

```
SELECT * FROM system_schema.tables WHERE keyspace_name = "aws";
```

您的表不应在此语句的输出中列出。请注意，删除表之前可能会有一段延迟。有关更多信息，请参阅 the section called “DROP TABLE”。

教程：使用接口 VPC 端点连接到 Amazon Keyspaces

本教程介绍了如何针对 Amazon Keyspaces 设置和使用接口 VPC 端点。

接口 VPC 端点可以在 Amazon VPC 中运行的虚拟私有云 (VPC) 与 Amazon Keyspaces 之间实现私有通信。接口 VPC 终端节点由提供支持 AWS PrivateLink，这是一项支持 VPC 和 AWS 服务之间私有通信的 AWS 服务。有关更多信息，请参阅 the section called “使用接口 VPC 端点”。

主题

教程先决条件和注意事项

在开始本教程之前，请按照中的 AWS 设置说明进行操作访问 Amazon Keyspaces（Apache Cassandra 兼容）。这些步骤包括注册 AWS 和创建有权访问 Amazon Keyspaces 的 AWS Identity and Access Management (IAM) 委托人。记下 IAM 用户的姓名和访问密钥，因为您稍后需要用到这些信息。

创建一个名为 myKeyspace 的键空间和至少一个表，以便稍后测试 VPC 端点连接。您可以在开始使用中找到详细说明。

完成先决条件步骤后，继续执行步骤 1：启动 Amazon EC2 实例。

步骤 1：启动 Amazon EC2 实例

在此步骤中，您将在默认 Amazon VPC 中启动 Amazon EC2 实例。然后，您可以为 Amazon Keyspaces 创建和使用 VPC 端点。

启动 Amazon EC2 实例

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 选择启动实例，然后执行以下操作：

   从 EC2 控制台控制面板中，在启动实例框中选择 启动实例，然后从显示的选项中选择启动实例。

   在名称与标签下，针对名称，为实例输入一个描述性名称。

   在应用程序和操作系统映像（亚马逊机器映像）下：

   - 选择快速启动，然后选择 Ubuntu。这是适用于您的实例的操作系统 (OS)。
   - 在亚马逊机器映像 (AMI) 下，您可以使用标记为符合免费套餐条件的默认映像。Amazon Machine Image (AMI) 是基本配置，用作您的实例的模板。

   在实例类型下：

   - 从实例类型列表中选择 t2.micro 实例类型，这是默认选择的类型。

   在密钥对（登录）下，对于密钥对名称，选择以下选项之一：

   - 如果没有 Amazon EC2 密钥对，请选择创建新密钥对，然后按照说明操作。系统会要求您下载私有密钥文件（.pem 文件）。稍后登录 Amazon EC2 实例时，您需要使用此文件，因此请记下文件路径。
   - 如果您现在已有 Amazon EC2 密钥对，请转至选择密钥对，然后从列表中选择您的密钥对。您必须有可用的私有密钥文件 (.pem 文件) 才能登录 Amazon EC2 实例。

   在网络设置下：

   - 选择编辑。
   - 选择选择现有安全组。

- 在安全组列表中，选择默认。这是 VPC 的默认安全组。

继续打开摘要。

- 在摘要窗格中，查看您的实例配置摘要。在您准备就绪后，选择启动实例。

3. 在新 Amazon EC2 实例的完成屏幕上，选择连接到实例磁贴。下一个屏幕会显示连接到新实例所需的信息和步骤。记下以下信息：

   - 保护密钥文件的示例命令
   - 连接字符串
   - 公有 IPv4 DNS 名称

   记下此页上的信息后，您可以继续执行本教程的下一步（第 2 步：配置 Amazon EC2 实例）。

> **Note**
>
> Amazon EC2 实例需要几分钟才能变为可用。在继续下一步之前，请确保实例状态为 `running`，并且已通过其所有状态检查。

第 2 步：配置 Amazon EC2 实例

当您的 Amazon EC2 实例可用时，您可以登录该实例并为首次使用做好准备。

> **Note**
>
> 以下步骤假设您从运行 Linux 的计算机连接到您的 Amazon EC2 实例。有关其他连接方式，请参阅适用于 Linux 实例的 Amazon EC2 用户指南中的连接到您的 Linux 实例。

配置您的 Amazon EC2 实例

1. 您需要对进入您的 Amazon EC2 实例的入站 SSH 流量进行授权。为此，请创建一个新的 EC2 安全组，然后将该安全组分配给您的 EC2 实例。

   a. 在导航窗格中，选择安全组。

   b. 选择创建安全组。在创建安全组窗口中，执行以下操作：

- 安全组名称 – 输入安全组的名称。例如：my-ssh-access
- 描述 - 输入对安全组的简短描述。
- VPC - 选择默认 VPC。
- 在入站规则部分，选择添加规则并执行以下操作：
  - 类型 - 选择 SSH。
  - 来源 - 选择我的 IP。
  - 选择 添加规则。

在页面底部，确认配置设置并选择创建安全组。

c. 在导航窗格中，选择实例。

d. 选择在 步骤 1：启动 Amazon EC2 实例 中启动的 Amazon EC2 实例。

e. 选择操作，选择安全，然后选择更改安全组。

f. 在更改安全组中，选择您之前创建的安全组（例如：my-ssh-access）。还应选择现有 default 安全组。确认配置设置并选择分配安全组。

2. 使用以下命令保护您的私有密钥文件不被访问。如果跳过此步骤，则连接将会失败。

```
chmod 400 path_to_file/my-keypair.pem
```

3. 使用 ssh 命令登录您的 Amazon EC2 实例，如以下示例所示。

```
ssh -i path_to_file/my-keypair.pem ubuntu@public-dns-name
```

您需要指定私有密钥文件（.pem 文件）和实例的公有 DNS 名称。（请参阅 步骤 1：启动 Amazon EC2 实例。）

登录 ID 为 ubuntu。不需要密码。

有关允许连接 Amazon EC2 实例的更多信息以及有关 AWS CLI 的说明，请参阅适用于 Linux 实例的 Amazon EC2 用户指南中的为 Linux 实例授权入站流量。

4. 下载并安装最新版本的 AWS Command Line Interface。

a. 安装 unzip。

```
sudo apt install unzip
```

b. 使用 AWS CLI下载 zip 文件。

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
 "awscliv2.zip"
```

c. 解压缩该文件。

```
unzip awscliv2.zip
```

d. 安装 AWS CLI。

```
sudo ./aws/install
```

e. 确认 AWS CLI 安装版本。

```
aws --version
```

输出应该如下所示：

```
aws-cli/2.9.19 Python/3.9.11 Linux/5.15.0-1028-aws exe/x86_64.ubuntu.22 prompt/
off
```

5. 配置您的 AWS 证书，如以下示例所示。出现提示时，输入您的 AWS 访问密钥 ID、密钥和默认区域名称。

```
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. 您必须使用与 Amazon Keyspaces 的 cqlsh 连接来确认您的 VPC 终端节点配置正确。如果您在中使用本地环境或使用 Amazon Keyspaces CQL 编辑器 AWS Management Console，则连接将自动通过公有终端节点而不是您的 VPC 终端节点进行。要使用 cqlsh 来测试本教程中的 VPC 端点连接，请按照使用 cqlsh 连接 Amazon Keyspaces中的设置说明操作。

现在，您可以为 Amazon Keyspaces 创建 VPC 端点。

步骤 3：为 Amazon Keyspaces 创建 VPC 端点

在这一步中，您将使用 AWS CLI为 Amazon Keyspaces 创建 VPC 端点。要使用 VPC 控制台创建 VPC 端点，您可以按照 AWS PrivateLink 指南中的创建 VPC 端点进行操作。筛选服务名称时，请输入 **`Cassandra`**。

使用创建 VPC 终端节点 AWS CLI

1. 在开始之前，请验证您是否可以使用 Amazon Keyspaces 的公共端点与其进行通信。

```
aws keyspaces list-tables --keyspace-name 'myKeyspace'
```

输出显示了包含在指定键空间中的 Amazon Keyspaces 表的列表。如果您没有任何表，该列表将为空。

```
{
    "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. 验证 Amazon Keyspaces 是否为在当前 AWS 区域创建 VPC 终端节点的可用服务。（命令以粗体文本显示，后面是示例输出。）

```
aws ec2 describe-vpc-endpoint-services

{
    "ServiceNames": [
        "com.amazonaws.us-east-1.cassandra",
        "com.amazonaws.us-east-1.cassandra-fips"
```

```
    ]
}
```

在示例输出中，Amazon Keyspaces 是可用的服务之一，因此您可以继续为其创建 VPC 端点。

3. 确定您的 VPC 标识符。

```
aws ec2 describe-vpcs

{
    "Vpcs": [
        {
            "VpcId": "vpc-a1234bcd",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "111.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

在示例输出中，VPC ID 为 vpc-a1234bcd。

4. 使用筛选条件来收集有关 VPC 子网的详细信息。

```
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,


        }
    ]
}
```

在示例输出中，有两个可用的子网 ID：subnet-920aacf9 和 subnet-4c713600。

5. 创建 VPC 终端节点。对于 --vpc-id 参数，指定上一步中的 VPC ID。对于 --subnet-id 参数，指定上一步中的 子网 ID。使用 --vpc-endpoint-type 参数将端点定义为接口。有关命令的更多信息，请参阅 AWS CLI 命令参考中的 create-vpc-endpoint。

```
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"
    }
}

}
```

步骤 4：为 VPC 端点连接配置权限

本步骤中的过程演示了如何配置规则和权限，以便将 VPC 端点与 Amazon Keyspaces 配合使用。

为新端点配置入站规则以允许 TCP 入站流量

1. 在 Amazon VPC 控制台的左侧面板上，选择端点，然后选择您在前面的步骤中创建的端点。
2. 选择安全组，然后选择与该端点关联的安全组。
3. 选择入站规则，然后选择 编辑入站规则。
4. 添加类型为自定义 TCP 的入站规则。对于端口范围，输入 **9142**。
5. 选择保存规则以保存新入站规则。

配置 IAM 用户权限

1. 确认用于连接 Amazon Keyspaces 的 IAM 用户是否具有相应的权限。在 AWS Identity and Access Management (IAM) 中，您可以使用 AWS 托管策略向 IAM 用户授`AmazonKeyspacesReadOnlyAccess`予对 Amazon Keyspaces 的读取权限。
   a. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
   b. 在 IAM 控制台控制面板上，选择 Users (用户)，然后从列表中选择您的 IAM 用户。
   c. 在 Summary (摘要) 页上，选择 Add permissions (添加权限)。
   d. 选择直接附加现有策略。
   e. 从策略列表中选择 AmazonKeyspacesReadOnlyAccess，然后选择下一步：查看。
   f. 选择添加权限。
2. 验证您是否可以通过 VPC 端点访问 Amazon Keyspaces。

```
aws keyspaces list-tables --keyspace-name 'my_Keyspace'
```

   如果你愿意，你可以尝试使用其他一些针对 Amazon Keyspaces 的 AWS CLI 命令。有关更多信息，请参阅 AWS CLI 命令参考。

> **Note**
>
> IAM 用户或角色访问 Amazon Keyspaces 所需的最低权限是对系统表的读取权限，如以下策略所示。有关基于策略的权限的更多信息，请参阅 the section called “基于身份的策略示例”。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:555555555555:/keyspace/system*"
         ]
      }
   ]
}
```

3. 向 IAM 用户授予对具有 VPC 的 Amazon EC2 实例的读取权限。

   将 Amazon Keyspaces 与 VPC 端点配合使用时，您需要向访问 Amazon Keyspaces 的 IAM 用户或角色授予对您的 Amazon EC2 实例和 VPC 的只读权限，以收集端点和网络接口数据。Amazon Keyspaces 将这一信息存储在 `system.peers` 表中，并使用它来管理连接。

   > **Note**
   >
   > 托管式策略 `AmazonKeyspacesReadOnlyAccess_v2` 和 `AmazonKeyspacesFullAccess` 包含允许 Amazon Keyspaces 访问 Amazon EC2 实例以读取可用接口 VPC 端点信息所需的权限。

   a. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。

   b. 在 IAM 控制台控制面板上，选择策略。

c. 选择创建策略，然后选择 JSON 选项卡。

d. 复制以下策略并选择下一步：标签。

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

e. 选择下一步：查看，输入策略的名称 `keyspacesVPCendpoint`，然后选择创建策略。

f. 在 IAM 控制台控制面板上，选择 Users (用户)，然后从列表中选择您的 IAM 用户。

g. 在 Summary (摘要) 页上，选择 Add permissions (添加权限)。

h. 选择直接附加现有策略。

i. 从策略列表中，依次选择 keyspacesVPCendpoint 和下一步：查看。

j. 选择添加权限。

4. 要验证 Amazon Keyspaces `system.peers` 表是否更新了 VPC 信息，请从您的 Amazon EC2 实例使用 `cqlsh` 运行以下查询。如果步骤 2 中您没有在 Amazon EC2 实例上安装 `cqlsh`，请按照the section called “使用cqlsh-expansion”中的说明进行操作。

```
SELECT peer FROM system.peers;
```

输出返回带有私有 IP 地址的节点，具体取决于您所在 AWS 地区的 VPC 和子网设置。

```
peer
---------------
112.11.22.123
112.11.22.124
112.11.22.125
```

> **Note**
>
> 您必须使用与 Amazon Keyspaces 的 `cqlsh` 连接来确认您的 VPC 终端节点配置正确。如果您使用本地环境或在 AWS Management Console中使用 Amazon Keyspaces CQL 编辑器，则连接将自动通过公有端点而不是您的 VPC 端点进行。如果您看到九个 IP 地址，则这些地址是 Amazon Keyspaces 针对公有端点连接自动写入 `system.peers` 表的条目。

步骤 5：使用配置监控 CloudWatch

此步骤向您展示如何使用亚马逊 CloudWatch 监控与 Amazon Keyspaces 的 VPC 终端节点连接。

AWS PrivateLink 向发布 CloudWatch 有关您的接口端点的数据点。您可使用指标来验证系统是否正常运行。中的`AWS/PrivateLinkEndpoints`命名空间 CloudWatch 包括接口端点的指标。有关更多信息，请参阅AWS PrivateLink 指南 AWS PrivateLink中的CloudWatch 指标。

创建包含 VPC 终端节点指标的 CloudWatch 控制面板

1. 打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择控制面板。然后选择创建控制面板。输入控制面板的名称，然后选择创建。
3. 在添加小部件下，选择数字。
4. 在 “指标” 下，选择 AWS/ PrivateLinkEndpoints。
5. 选择端点类型、服务名称、VPC 端点 ID、VPC ID。
6. 选择 `ActiveConnections` 和 `NewConnections` 指标，然后选择创建小部件。
7. 保存控制面板。

`ActiveConnections` 指标的定义为端点在过去一分钟内收到的并发活动连接数。`NewConnections` 指标的定义为过去一分钟内通过端点建立的新连接数。

有关创建仪表板的更多信息，请参阅《CloudWatch 用户指南》中的创建仪表板。

步骤 6：（可选）为应用程序配置连接池大小的最佳实践

本部分介绍如何根据应用程序的查询吞吐量要求来确定理想的连接池大小。

Amazon Keyspaces 允许每个 TCP 连接每秒最多 3000 次 CQL 查询。因此，驱动程序可以与 Amazon Keyspaces 建立的连接数量几乎不受限制。但是，我们建议您将连接池大小与应用程序的要求相匹配，并在使用 Amazon Keyspaces 和 VPC 端点连接时考虑可用的端点。

您可以在客户端驱动程序中配置连接池的大小。例如，如果本地池大小为 2，VPC 接口端点在 3 个可用区中创建，那么驱动程序会建立 6 个用于查询的连接（总共 7 个，包括一个控制连接）。使用这 6 个连接，您每秒最多可以支持 18000 次 CQL 查询。

如果您的应用程序需要支持每秒 40000 次 CQL 查询，请根据需要的查询数量来确定所需的连接池大小。要支持每秒 40000 次 CQL 查询，您需要将本地池大小至少配置为 5，这样每秒至少可以支持 45000 次 CQL 查询。

您可以使用AWS/Cassandra命名空间中的PerConnectionRequestRateExceeded CloudWatch指标来监控是否超过每个连接每秒最大操作数的配额。PerConnectionRequestRateExceeded 指标可以显示向 Amazon Keyspaces 发出的、超出每连接请求速率限额的请求数量。

本步骤中的代码示例显示了在使用接口 VPC 端点时如何估算和配置连接池。

Java

您可以在 Java 驱动程序中配置每个池的连接数。有关 Java 客户端驱动程序连接的完整示例，请参阅the section called “使用 Cassandra Java 客户端驱动程序”。

客户端驱动程序启动后，首先为管理任务（例如架构和拓扑更改）建立控制连接。然后创建其他连接。

在以下示例中，本地池大小驱动程序配置被指定为 2。如果 VPC 终端节点是在 VPC 内的 3 个子网中创建的，则接口终端节点 CloudWatch 的结果为 7NewConnections，如以下公式所示。

```
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
          }
}
```

如果活动连接的数量与您配置的池大小（跨子网聚合）+ 1 个控制连接不匹配，则无法创建连接。

Node.js

您可以在 Node.js 驱动程序中配置每个池的连接数。有关 Node.js 客户端驱动程序连接的完整示例，请参阅the section called “使用 Cassandra Node.js 客户端驱动程序”。

在以下示例中，本地池大小驱动程序配置被指定为 1。如果 VPC 终端节点是在 VPC 内的 4 个子网中创建的，则接口终端节点将获得 5 个NewConnections子网，如以下公式所示。 CloudWatch

```
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 }
        });
```

第 7 步：（可选）清除

如果要删除您在本教程中创建的资源，请按照以下程序操作。

删除用于 Amazon Keyspaces 的 VPC 端点

1. 登录到您的 Amazon EC2 实例。
2. 确定用于 Amazon Keyspaces 的 VPC 端点 ID。如果不使用 `grep` 参数，则系统会显示所有服务的 VPC 端点信息。

```
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"
    }
}
```

   在示例输出中，VPC 终端节点 ID 为`vpce-9b15e2f2`。
3. 删除 VPC 终端节点。

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2

{
    "Unsuccessful": []
}
```

空数组 [] 表示成功（没有失败请求）。

终止 Amazon EC2 实例

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在导航窗格中，选择 Instances (实例)。
3. 选择您的 Amazon EC2 实例。
4. 选择操作，选择实例状态，然后选择终止。
5. 在确认窗口中，选择是，终止。

配置针对 Amazon Keyspaces 的跨账户访问

您可以创建和使用单独的 AWS 账户来隔离资源，并用于不同的环境，例如开发环境和生产环境。本主题介绍了如何在 Amazon Virtual Private Cloud 中使用接口 VPC 端点对 Amazon Keyspaces 进行跨账户访问。有关 IAM 跨账户访问配置的更多信息，请参阅《IAM 用户指南》中使用单独的开发账户和生产账户的示例场景。

有关 Amazon Keyspaces 和私有 VPC 端点的更多信息，请参阅the section called “使用接口 VPC 端点”。

主题

- 为共享 VPC 中的 Amazon Keyspaces 配置跨账户访问
- 为没有共享 VPC 的 Amazon Keyspaces 配置跨账户访问

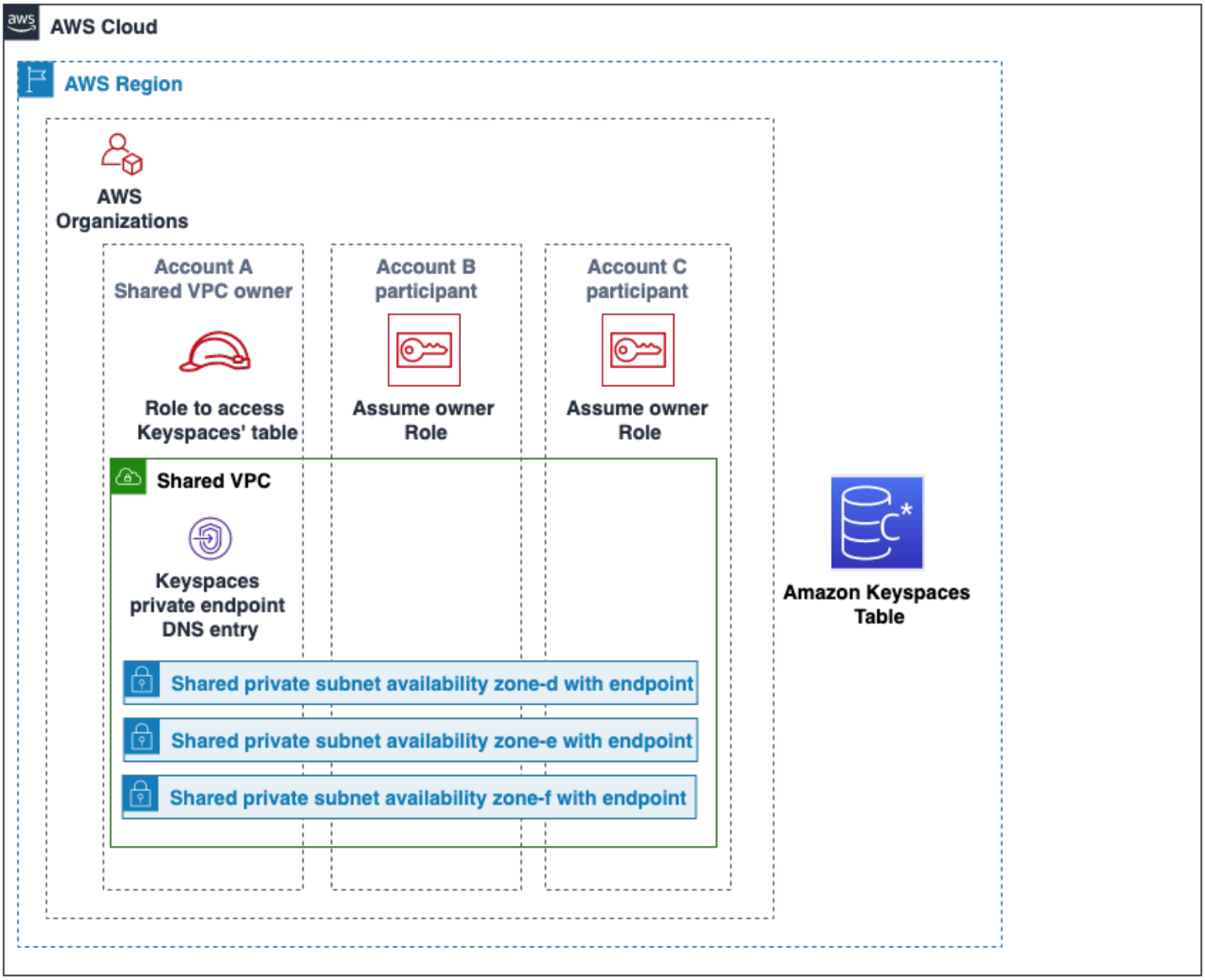
为共享 VPC 中的 Amazon Keyspaces 配置跨账户访问

您可以创建不同的 AWS 账户，将资源与应用程序分开。例如，您可以为 Amazon Keyspaces 表创建一个账户，为开发环境中的应用程序创建一个不同的账户，为生产环境中的应用程序创建另一个账户。本主题将引导您完成在共享 VPC 中使用接口 VPC 端点为 Amazon Keyspaces 设置跨账户访问所需的配置步骤。

有关如何为 Amazon Keyspaces 配置 VPC 端点的详细步骤，请参阅the section called “步骤 3：为 Amazon Keyspaces 创建 VPC 端点”。

在本例中，我们在共享 VPC 中使用了以下三个账户：

- Account A：此账户包含基础设施，包括 VPC 端点、VPC 子网和 Amazon Keyspaces 表。
- Account B：此账户包含开发环境中的一个应用程序，该应用程序需要连接到Account A 中的 Amazon Keyspaces 表。
- Account C：此账户包含生产环境中的一个应用程序，该应用程序需要连接到Account A 中的 Amazon Keyspaces 表。



Account A 包含Account B 和Account C 需要访问的资源，因此Account A 是信任账户。Account B 和Account C 包含需要访问Account A 中的资源的主体，因此Account B 和Account C 是受信账户。信任账户通过共享 IAM 角色向受信账户授予权限。以下程序概述了Account A 中所需的配置步骤。

Account A 所需的配置

1. 使用 AWS Resource Access Manager 为子网创建资源共享，并与Account B 和Account C 共享私有子网。

   Account B 和Account C 现在可以在与它们共享的子网中查看和创建资源。

2. 创建由 AWS PrivateLink 提供支持的 Amazon Keyspaces 私有 VPC 端点。这将在 Amazon Keyspaces 服务端点的共享子网和 DNS 条目上创建多个端点。

3. 创建 Amazon Keyspaces 密钥空间和表。

4. 创建一个 IAM 角色，该角色拥有对 Amazon Keyspaces 表的完全访问权限、对 Amazon Keyspaces 系统表的读取权限，并且能够描述 Amazon EC2 VPC 资源，如以下策略示例所示。

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

5. 配置账户 B 和账户 C 可以作为受信任账户代入的 IAM 角色信任策略，如下例所示。

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

```
    ]
}
```

有关跨账户 IAM 策略的更多信息，请参阅《IAM 用户指南》中的跨账户策略。

Account B 和 Account C 中的配置

1. 在Account B 和Account C 中创建新角色并附加以下策略，允许主体代入在Account A 中创建的共享角色。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

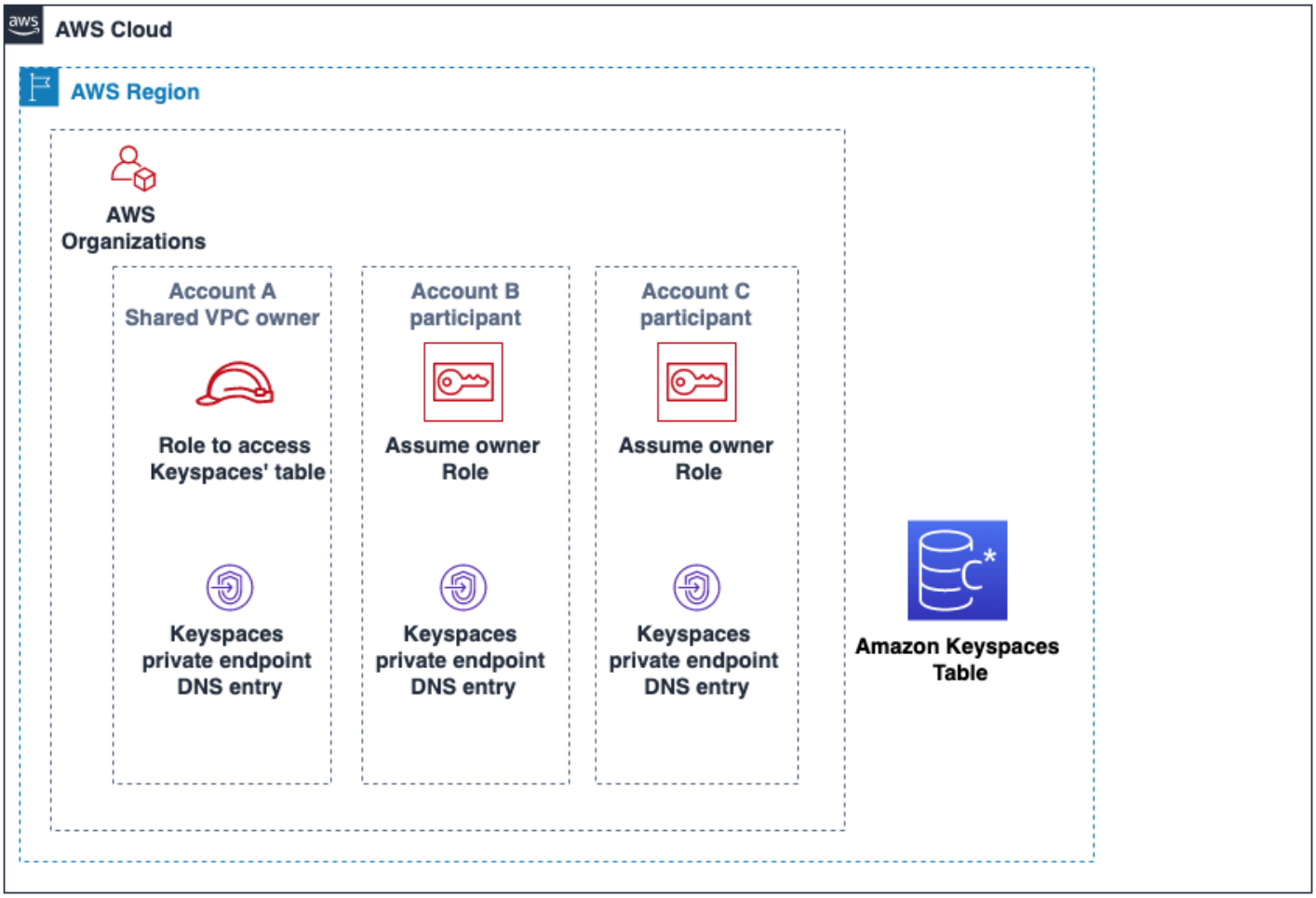
允许主体代入共享角色是通过 AWS Security Token Service (AWS STS) 的 AssumeRole API 实现的。有关更多信息，请参阅《IAM 用户指南》中的向您拥有的其他 AWS 账户中的 IAM 用户提供访问权限。

2. 在Account B 和Account C 中，您可以创建使用 SIGV4 身份验证插件的应用程序，该插件让应用程序可以代入共享角色，通过共享 VPC 中的 VPC 端点连接到位于Account A 的 Amazon Keyspaces 表。有关 SIGV4 身份验证插件的更多信息，请参阅the section called “创建凭证”。

为没有共享 VPC 的 Amazon Keyspaces 配置跨账户访问

如果 Amazon Keyspaces 表和私有 VPC 端点由不同的账户拥有但不共享 VPC，则应用程序仍可使用 VPC 端点进行跨账户连接。由于账户不共享 VPC 端点，Account A、Account B 和Account C 需要各自的 VPC 端点。在 Cassandra 客户端驱动程序中，Amazon Keyspaces 显示为单节点集群而不是多节点集群。连接后，客户端驱动程序会到达 DNS 服务器，服务器会返回账户 VPC 中的一个可用终端节点。

您还可以使用公共端点或在每个账户中部署私有 VPC 端点，在没有共享 VPC 端点的情况下跨不同账户访问 Amazon Keyspaces 表。不使用共享 VPC 时，每个账户都需要自己的 VPC 端点。在本示例中，`Account A`、`Account B` 和`Account C` 都需要各自的 VPC 端点才能访问`Account A` 中的表。在此配置中使用 VPC 端点时，Amazon Keyspaces 在 Cassandra 客户端驱动程序中显示为单节点集群，而不是多节点集群。连接后，客户端驱动程序会到达 DNS 服务器，服务器会返回账户 VPC 中的一个可用终端节点。但客户端驱动程序无法访问 `system.peers` 表来发现其他端点。由于可用主机较少，驱动程序建立的连接也较少。要对此进行调整，请将驱动程序的连接池设置提高 3 倍。



Amazon Keyspaces（Apache Cassandra 兼容）

如果您刚刚接触 Apache Cassandra 和 Amazon Keyspaces（Apache Cassandra 兼容），那么本教程正好适合您。在本教程中，您将会安装成功使用 Amazon Keyspaces 所需的所有程序和驱动程序。

有关使用不同 Cassandra 客户端驱动程序以编程方式连接 Amazon Keyspaces 的教程，请参阅the section called “使用 Cassandra 客户端驱动程序”。

主题

教程先决条件和注意事项

在开始本教程之前，请按照中的 AWS 设置说明进行操作访问 Amazon Keyspaces（Apache Cassandra 兼容）。这些步骤包括注册 AWS 和创建有权访问 Amazon Keyspaces 的 AWS Identity and Access Management (IAM) 用户。

此外，如果您使用 `cqlsh` 或 Apache 2.0 许可的 Cassandra 客户端驱动程序完成本教程，请完成使用 `cqlsh` 连接 Amazon Keyspaces中的设置说明。

完成先决条件步骤后，继续执行教程步骤 1：在 Amazon Keyspaces 中创建键空间和表。

教程步骤 1：在 Amazon Keyspaces 中创建键空间和表

在本部分中，您将使用控制台创建键空间，并向其中添加表。

> **Note**
>
> 开始之前，请确保您已具备所有教程先决条件。

主题

创建键空间

键空间 对与一个或多个应用程序相关的表进行分组。键空间包含一个或多个表，并为其包含的所有表定义复制策略。有关键空间的更多信息，请参阅以下主题：

- 使用键空间：the section called “创建键空间”
- 数据定义语言 (DDL) 语句：Keyspaces
- Amazon Keyspaces（Apache Cassandra 兼容）限额

创建键空间时，必须指定键空间名称。

> **Note**
>
> 键空间的复制策略必须为 `SingleRegionStrategy`。`SingleRegionStrategy` 会在其 AWS 区域中跨三个可用区复制数据。

使用 控制台

使用控制台创建键空间

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 在导航窗格中，选择 Keyspaces (键空间)。
3. 选择 Create keyspace (创建键空间)。
4. 在 Keyspace name (键空间名称) 框中，输入 **`myGSGKeyspace`** 作为键空间的名称。

   名称约束：

   - 不能为空。
   - 允许的字符：字母数字字符和下划线 (_)。
   - 最大长度为 48 个字符。
5. 要创建键空间，请选择 Create keyspace (创建键空间)。
6. 通过执行以下操作，验证键空间 `myGSGKeyspace` 是否已创建：

   a. 在导航窗格中，选择 Keyspaces (键空间)。

   b. 在键空间列表中，查找键空间 `myGSGKeyspace`。

使用 CQL

以下过程使用 CQL 创建键空间。

使用 CQL 创建键空间

1. 打开命令 shell，输入以下内容：

   ```
   cqlsh
   ```

2. 使用以下 CQL 命令创建键空间。

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

   `SingleRegionStrategy`使用复制系数为三，并在其所在区域的三个 AWS 可用区之间复制数据。

   > **Note**
   >
   > Amazon Keyspaces 默认所有输入为小写，除非括在引号中。在此情况下，请标注 `"myGSGKeyspace"`。

3. 验证键空间是否已创建。

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

   应当列出您的键空间。

创建表

表是组织和存储数据的位置。表的主键决定如何在表中对数据进行分区。主键由一个必需的分区键和一个或多个可选的聚类列组成。组成主键的组合值在表的所有数据中必须是唯一的。有关表的更多信息，请参阅以下主题：

- 使用表：the section called “创建表”
- DDL 语句：表
- 表资源管理：无服务器资源管理
- 监控表资源利用率：the section called “使用监控 CloudWatch”

- Amazon Keyspaces（Apache Cassandra 兼容）限额

创建表时，应指定以下内容：

- 表的名称。
- 表中每一列的名称和数据类型。
- 表的主键。
  - 分区键 – 必需
  - 聚类列 – 可选

可使用以下过程创建具有指定列、数据类型、分区键和聚类列的表。

使用 控制台

以下过程创建包含如下列和数据类型的表 `employees_tbl`。

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

使用控制台创建表

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 在导航窗格中，选择 Keyspaces (键空间)。
3. 选择 `myGSGKeyspace` 作为要在其中创建此表的键空间。
4. 选择创建表。
5. 在 Table name (表名称) 框中，输入 **`employees_tbl`** 作为表的名称。

   名称约束：

   - 不能为空。

- 允许的字符：字母数字字符和下划线 (_)。
- 最大长度为 48 个字符。

6. 在 Columns (列) 部分中，为要添加到此表的每一列重复以下步骤。

   添加以下列和数据类型。

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

   a. 名称 – 输入列的名称。

      名称约束：

      - 不能为空。
      - 允许的字符：字母数字字符和下划线 (_)。
      - 最大长度为 48 个字符。

   b. 类型 – 在数据类型列表中，为此列选择数据类型。

   c. 如果要添加另一列，请选择 Add column (添加列)。

7. 在分区键下，选择id作为分区键。每个表都需要一个分区键。分区键可以由一列或多列构成。
8. 添加 division 作为聚类列。聚类列是可选的，决定着每个分区内的排序顺序。

   a. 要添加聚类列，请选择 Add clustering column (添加聚类列)。

   b. 在 Column (列) 列表中，选择 division (分类)。在 Order (顺序) 列表中，选择 ASC (升序) 按照升序对此列中的值进行排序。（选择 DESC (降序) 可按降序排列。）

9. 在表设置部分中，选择默认设置。
10. 选择创建表。
11. 验证表是否已创建。

    a. 在导航窗格中，选择表。

    b. 确认您的表位于表列表中。

c. 选择表的名称。

d. 确认所有列和数据类型都正确无误。

> **Note**
>
> 列的顺序可能与添加到表中的顺序不同。

e. 在聚类列中，确认 division (分类) 标识为 true。其他所有表列应当为 false。

使用 CQL

以下过程使用 CQL 创建包含如下列和数据类型的表。id 列将作为分区键。

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

使用 CQL 创建表

1. 打开命令 shell，输入以下内容：

   cqlsh

2. 在 cqlsh 提示符 (cqlsh>) 处，指定要在其中创建表的键空间。

```
USE "myGSGKeyspace" ;
```

3. 在键空间提示符 (cqlsh:keyspace_name>) 处，通过在命令窗口中输入以下代码来创建表。

```
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 是默认的聚类顺序。也可以指定 DESC 按降序排列。

请注意，id 列将作为分区键。然后，division 是按升序 (ASC) 排列的聚类列。

4. 验证表是否已创建。

```
SELECT * from system_schema.tables WHERE keyspace_name='myGSGKeyspace' ;
```

应当列出您的表。

5. 验证表的结构。

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

确认所有列和数据类型均符合预期。列的顺序可能与 CREATE 语句中的顺序不同。

要对表中的数据执行 CRUD（创建、读取、更新和删除）操作，请继续执行the section called “步骤 2：CRUD 操作”。

教程步骤 2：创建、读取、更新和删除数据 (CRUD)

在本节中，您将使用控制台中的 CQL 编辑器，对表中的数据执行 CRUD（创建、读取、更新和删除）操作。您也可以使用 cqlsh 运行命令。

主题

- 教程：在 Amazon Keyspaces 表中插入和加载数据
- 教程：从 Amazon Keyspaces 表中读取

- 教程：更新 Amazon Keyspaces 表中的数据
- 教程：删除 Amazon Keyspaces 表中的数据

教程：在 Amazon Keyspaces 表中插入和加载数据

要在 employees_tbl 表中创建数据，请使用 INSERT 语句添加单行。

1. 在使用 cqlsh 将数据写入 Amazon Keyspaces 表之前，您必须将当前 cqlsh 会话的写入一致性设置为 LOCAL_QUORUM。有关支持的一致性级别的更多信息，请参阅the section called “写入一致性级别”。请注意，如果您在 AWS Management Console中使用 CQL 编辑器，则不需要执行此步骤。

```
CONSISTENCY LOCAL_QUORUM;
```

2. 要插入单个记录，请在 CQL 编辑器中运行以下命令。

```
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. 通过运行以下命令，验证数据是否已正确添加到表中。

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

使用 cqlsh 从文件插入多个记录

1. 下载以下压缩文件 sampledata.zip 中包含的样本数据文件 (employees.csv)。此 CSV（逗号分隔值）文件包含以下数据。记住文件的保存路径。

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. 打开命令 shell，输入以下内容：

cqlsh

3. 在 cqlsh 提示符 (cqlsh>) 处，指定键空间。

```
USE "myGSGKeyspace" ;
```

4. 将写入一致性设置为 LOCAL_QUORUM。有关支持的一致性级别的更多信息，请参阅the section called “写入一致性级别”。

```
CONSISTENCY LOCAL_QUORUM;
```

5. 在键空间提示符 (cqlsh:keyspace_name>) 处，运行以下查询。

```
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. 通过运行以下查询，验证数据是否已正确添加到表中。

```
SELECT * FROM employees_tbl ;
```

教程：从 Amazon Keyspaces 表中读取

在教程：在 Amazon Keyspaces 表中插入和加载数据一节中，您使用 SELECT 语句验证了已成功将数据添加到表中。在本节中，您可以细化使用 SELECT，以只显示特定列以及满足特定条件的行。

SELECT 语句的一般形式如下所示。

```
SELECT column_list FROM table_name [WHERE condition [ALLOW FILTERING]] ;
```

主题

- 选择表中的所有数据
- 选择列的子集
- 选择行的子集

选择表中的所有数据

SELECT 语句最简单的形式是返回表中的所有数据。

> **Important**
>
> 在生产环境中，运行此命令通常不是最佳做法，该命令将返回表中的所有数据。

选择表的所有数据

- 运行以下 查询。

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

将通配符 (*) 用于 column_list 可选择所有列。

选择列的子集

查询列的子集

- 要仅检索 id、name 和 manager_id 列，请运行以下查询。

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

输出将按照 SELECT 语句中列出的顺序，仅包含指定的列。

选择行的子集

查询大型数据集时，您可能只需要满足特定条件的记录。为此，您可以在 SELECT 语句末尾追加一个 WHERE 子句。

查询行的子集

- 要仅检索 ID 为 '234-56-7890' 的员工的记录，请运行以下查询。

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

前面的 SELECT 语句仅返回 id 为 234-56-7890 的行。

了解 **WHERE** 子句

WHERE 子句用于筛选数据，并仅返回满足指定条件的数据。指定的条件可以是简单条件或复合条件。

如何在 **WHERE** 子句中使用条件

- 简单条件 – 单列

  ```
  WHERE column_name=value
  ```

  只要满足以下任何条件，便可以在 WHERE 子句中使用简单条件：
  - 该列是表的主键中唯一的列。
  - 在 WHERE 子句中的条件后添加 ALLOW FILTERING。

    请注意，使用 ALLOW FILTERING 可能会导致性能不稳定，特别是对于多分区的大型表。
- 复合条件 – 通过 AND 连接的多个简单条件

  ```
  WHERE column_name1=value1 AND column_name2=value2 AND column_name3=value3...
  ```

  只要满足以下任何条件，便可以在 WHERE 子句中使用复合条件：
  - WHERE 子句中的列与表主键中的列完全匹配，不多也不少。
  - 在 WHERE 子句中的复合条件后添加 ALLOW FILTERING，如下例所示。

    ```
    SELECT * FROM my_table WHERE col1=5 AND col2='Bob' ALLOW FILTERING ;
    ```

    请注意，使用 ALLOW FILTERING 可能会导致性能不稳定，特别是对于多分区的大型表。

试试看

创建您自己的 CQL 查询，以从 employees_tbl 表中查找以下内容：

- 查找所有员工的 name、project 和 id。
- 查找实习生 Bob 正在从事的项目（在输出中至少包括他的姓名、项目和角色）。
- 高级：创建应用程序，以查找与实习生 Bob 具有相同经理的所有员工。提示：这可能需要多个查询。
- 高级：创建应用程序，以查找正从事项目 NightFlight 的所有员工的选定列。提示：解决这个问题可能需要多个语句。

教程：更新 Amazon Keyspaces 表中的数据

要更新 employees_tbl 表中的数据，请使用 UPDATE 语句。

UPDATE 语句的一般形式如下所示。

```
UPDATE table_name SET column_name=new_value WHERE primary_key=value ;
```

> **Tip**
>
> - 您可以使用逗号分隔的 column_names 和值列表来更新多列，如下例所示。
>
>   ```
>   UPDATE my_table SET col1='new_value_1', col2='new_value2' WHERE id='12345' ;
>   ```
>
> - 如果主键由多列构成，则必须将所有主键列及其值包含在 WHERE 子句中。
> - 不能更新主键中的任何列，因为这会更改记录的主键。

更新单个单元格

使用 employees_tbl 表，给 ID 为 567-89-0123 的员工加薪。

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

验证该员工的薪酬等级现在是否为 5。

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

试试看

高级：贵公司雇用了 Bob 作为实习生。更改其记录，使其角色为 'IC'，薪酬等级为 2。

教程：删除 Amazon Keyspaces 表中的数据

要删除 employees_tbl 表中的数据，请使用 DELETE 语句。

您可以从行或分区中删除数据。删除数据时要小心，因为删除是不可逆操作。

从表中删除一行或所有行不会删除该表。因此，您可以用数据重新填充该表。删除表将删除表及其中的所有数据。要再次使用表，必须重新创建表并向其中添加数据。删除键空间会删除键空间及其中的所有表。要使用键空间和表，必须重新创建，然后用数据填充。

删除单元格

从行中删除列会从指定单元格中删除数据。使用 SELECT 语句显示该列时，数据将显示为 *null*，尽管该位置并未存储 null 值。

删除一个或多个特定列的一般语法如下所示。

```
DELETE column_name1[, column_name2...] FROM table_name WHERE condition ;
```

在 employees_tbl 表中，可以看到 CEO 具有 "None" 位经理。首先，删除该单元格，以便其中不会有任何数据。

删除特定单元格

1. 运行以下 DELETE 查询。

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

2. 验证删除操作是否如预期。

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

删除行

有时候可能需要删除整个行，例如当员工退休时。删除行的一般语法如下所示。

```
DELETE FROM table_name WHERE condition ;
```

删除行

1. 运行以下 DELETE 查询。

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

2. 验证删除操作是否如预期。

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

教程步骤 3：删除 Amazon Keyspaces 中的表和键空间

为避免为不需要的表和数据支付费用，请删除所有不使用的表和键空间。表删除后，该表及其数据将被删除，其费用累计将停止。但是，键空间仍然保留。键空间删除后，键空间及其所有表将被删除，其费用累计将停止。

删除表

您可以使用控制台或 CQL 删除表。表删除后，该表及其所有数据都将被删除。

使用 控制台

以下过程使用 AWS Management Console删除表及其所有数据。

使用控制台删除表

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 在导航窗格中，选择表。
3. 选中要删除的每个表名称左侧的框。
4. 选择删除。
5. 在 Delete table (删除表) 屏幕上，在框中输入 **`Delete`**。然后，选择 Delete table (删除表)。
6. 要验证表是否已删除，请在导航窗格中选择 Tables (表)，然后确认该 `employees_tbl` 表已不再列出。

使用 CQL

以下过程使用 CQL 删除表及其所有数据。

使用 CQL 删除表

1. 打开命令 shell，输入以下内容：

   ```
   cqlsh
   ```

2. 通过在键空间提示符 (cqlsh:*keyspace_name*>) 处输入以下命令来删除表。

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

3. 验证表是否已删除。

   ```
   SELECT * FROM system_schema.tables WHERE keyspace_name = 'myGSGKeyspace' ;
   ```

   您的表不应再列出。

删除键空间

您可以使用 AWS Management Console 或 CQL 删除密钥空间。键空间删除后，键空间及其所有表和数据都将被删除。

使用 AWS Management Console

以下过程使用 AWS Management Console删除键空间及其所有表和数据。

使用控制台删除键空间

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 在导航窗格中，选择 Keyspaces (键空间)。
3. 选中要删除的每个键空间名称左侧的框。
4. 选择删除。
5. 在 Delete keyspace (删除键空间) 屏幕上，在框中输入 **`Delete`**。然后，选择 Delete keyspace (删除键空间)。
6. 要验证键空间 myGSGKeyspace 是否已删除，请在导航窗格中选择 Keyspaces (键空间)，然后确认该键空间已不再列出。因为您删除了其键空间，所以表下的 employees_tbl 表也不应被列出。

使用 CQL

以下过程使用 CQL 删除键空间及其所有表和数据。

使用 CQL 删除键空间

1. 打开命令 shell，输入以下内容：

   cqlsh

2. 通过在键空间提示符 (cqlsh:*keyspace_name*>) 处输入以下命令来删除键空间。

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

3. 验证键空间是否已删除。

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

   您的键空间不应再列出。请注意，由于这是异步操作，因此在删除密钥空间之前可能会有一段延迟。

迁移到 Amazon Keyspaces

Amazon Keyspaces（Apache Cassandra 兼容）是一种可扩展、高可用、托管式的 Apache Cassandra 兼容数据库服务。您可以按照本部分中的步骤将数据从在本地或 Amazon Elastic Compute Cloud (Amazon EC2) 上运行的 Cassandra 数据库迁移到 Amazon Keyspaces。

建议您遵循以下最佳实践，确保迁移成功：

- 将迁移分解为较小的组件。

  考虑以下迁移单位及其在原始数据大小方面的潜在占用空间。在一个或多个阶段迁移少量数据可能有助于简化迁移。

  按集群：一次性迁移所有 Cassandra 数据。这种方法可能适用于较小的集群。

  按键空间或表：将迁移分解为键空间或表组。此方法可以帮助您根据每个工作负载的要求分阶段迁移数据。

  按数据：考虑迁移特定用户组或产品的数据，进一步减少数据大小。

- 根据简便性，确定要首先迁移的数据的优先顺序。

  考虑一下您是否有可以首先且更轻松地迁移的数据，例如，在特定时间段内不会更改的数据、来自夜间批处理作业的数据、离线期间未使用的数据或来自内部应用程序的数据。

- 使用特定的工具。
  - 使用 cqlsh `COPY FROM` 命令快速开始将数据加载到 Amazon Keyspaces。cqlsh 包含在 Apache Cassandra 中，最适用于加载小型数据集或测试数据。如需分步指导，请参阅 the section called “使用 cqlsh 加载数据”。
  - 对于具有大型数据集的生产工作负载，您可以使用适用于 Apache Cassandra 的 DataStax 批量加载器通过 `dsbulk` 命令将数据加载到 Amazon Keyspaces。DSBulk 提供了更强大的导入功能，可从 GitHub 存储库中获取该工具。如需分步指导，请参阅 the section called “使用 DSBulk 加载数据”。
  - 要了解如何使用 Apache Cassandra Spark 连接器向 Amazon Keyspaces 写入数据，请参阅 与 Apache Spark 集成。
  - 对于复杂迁移，请考虑使用提取、转换、加载 (ETL) 工具。您可以使用 AWS Glue 来快速有效地执行数据转换迁移。有关更多信息，请参阅 Migrate Apache Cassandra workloads to Amazon Keyspaces using AWS Glue。

主题

教程：使用 cqlsh 将数据加载到 Amazon Keyspaces

本 step-by-step 教程将指导你使用命令将数据从 Apache Cassandra 迁移到 Amazon Keyspaces。`cqlsh COPY`在本教程中，您将执行以下操作：

主题

先决条件

在开始本教程之前，您必须完成以下任务：

1. 如果您还没有这样做，请 AWS 账户 按照中的步骤进行注册the section called “设置 AWS Identity and Access Management”。
2. 按照the section called “使用控制台生成服务特定凭证”中的步骤创建特定于服务的凭证。
3. 设置 Cassandra 查询语言 Shell (cqlsh) 连接，并按照the section called “使用 `cqlsh`” 中的步骤确认您可以连接到 Amazon Keyspaces。

步骤 1：创建源 CSV 文件和目标表

在本教程中，我们使用名为 `keyspaces_sample_table.csv` 的逗号分隔值 (CSV) 文件作为用于数据迁移的源文件。提供的示例文件包含名为 `book_awards` 的表中的几行数据。

1. 创建源文件。您可以选择以下选项之一：

- 下载以下存档文件 samplemigration.zip 中包含的示例 CSV 文件 (keyspaces_sample_table.csv)。解压缩存档文件并记下指向 keyspaces_sample_table.csv 的路径。
- 要使用您自己存储在 Apache Cassandra 数据库中的数据来填充 CSV 文件，您可以使用 cqlsh COPY TO 语句来填充源 CSV 文件，如以下示例所示。

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

确保您创建的 CSV 文件符合以下要求：

- 第一行包含列名称。
- 源 CSV 文件中的列名称与目标表中的列名称相匹配。
- 数据用逗号分隔。
- 所有数据值均为有效的 Amazon Keyspaces 数据类型。请参阅 the section called “数据类型”。

2. 在 Amazon Keyspaces 中创建目标键空间和表。

a. 使用 cqlsh 连接到 Amazon Keyspaces，将以下示例中的服务端点、用户名和密码替换为您自己的值。

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

b. 使用名称 catalog 创建新的键空间，如以下示例所示。

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

c. 当新的键空间可用后，使用以下代码创建目标表 book_awards。

```
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)
```

```
    );
```

如果 Apache Cassandra 是您的原始数据来源，那么创建带有匹配标题的 Amazon Keyspaces 目标表的一种简单方法是从源表生成 CREATE TABLE 语句，如以下语句所示。

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

然后，在 Amazon Keyspaces 中创建目标表，其列名称和数据类型与 Cassandra 源表中的描述相匹配。

步骤 2：准备数据

为高效传输准备源数据的过程包含两个步骤。第一步，随机化数据。第二步，分析数据以确定相应的 cqlsh 参数值和所需的表设置。

随机化数据

cqlsh COPY FROM 命令按数据在 CSV 文件中显示的顺序读取和写入数据。如果使用 cqlsh COPY TO 命令创建源文件，将在 CSV 中按键排序顺序写入数据。Amazon Keyspaces 在内部使用分区键对数据进行分区。尽管 Amazon Keyspaces 具有内置逻辑来帮助对针对同一分区键的请求进行负载均衡，但如果您随机排列顺序，则可以更快、更高效地加载数据。这是因为您可以利用 Amazon Keyspaces 在写入不同分区时会出现的内置负载均衡功能。

要将写入操作均匀地分布在分区中，您必须随机化源文件中的数据。您可以编写一个应用程序来执行此操作，也可以使用开源工具来执行此操作，比如 Shuf。Shuf 在 Linux 发行版、macOS（通过在 Homebrew 中安装 coreutils）和 Windows [通过使用 Windows Subsystem for Linux (WSL)] 上免费提供。您还需要执行一个额外步骤来防止包含列名称的标题行在此步骤中被随机排序。

要在保留标题的同时随机化源文件，请输入以下代码。

```
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 将数据重写到名为 keyspace.table.csv 的新 CSV 文件中。现在，您可以删除 keyspaces_sample_table.csv 文件，您不再需要此文件了。

分析数据

通过分析数据来确定平均行大小和最大行大小。

执行此操作出于以下原因：

- 平均行大小有助于估算要传输的数据总量。
- 您需要平均行大小来预置上传数据所需的写入容量。
- 您可以确保每行的大小小于 1MB，这是 Amazon Keyspaces 中的最大行大小。

> **Note**
>
> 此限额指的是行大小，而不是分区大小。与 Apache Cassandra 分区不同，Amazon Keyspaces 分区实际上可以不受大小限制。分区键和聚类列需要额外的元数据存储空间，您必须将其加到行的原始大小中。有关更多信息，请参阅the section called “计算行大小”。

以下代码使用 AWK 分析 CSV 文件并打印平均行大小和最大行大小。

```
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
```

运行此代码将生成以下输出。

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

在本教程的下一步中，您将使用平均行大小来预置表的写入容量。

步骤 3：为表设置吞吐容量

本教程向您展示了如何调整 cqlsh 以在设定的时间范围内加载数据。由于您提前知道自己要执行多少读取和写入操作，因此可以使用预置容量模式。完成数据传输后，应该将表的容量模式设置为与应用程序的流量模式相匹配。要了解有关容量管理的更多信息，请参阅 无服务器资源管理。

使用预置容量模式，您可以提前指定要为表预置多少读取和写入容量。写入容量按小时计费，并以写入容量单位 (WCU) 计量。每个 WCU 的写入容量足以支持每秒写入 1KB 数据。加载数据时，写入速率必须低于目标表上设置的最大 WCU（参数：`write_capacity_units`）。

默认情况下，您可以为一个表预置最多 40000 个 WCU，可以为账户中的所有表预置最多 80000 个 WCU。如果您需要更多容量，可以在服务限额控制台中请求提高限额。有关限额的更多信息，请参阅限额。

计算一次插入所需的平均 WCU 数量

每秒插入 1KB 数据需要 1 个 WCU。如果您的 CSV 文件有 360000 行，并且您想在 1 小时内加载所有数据，则必须每秒写入 100 行（360000 行/60 分/60 秒 = 每秒 100 行）。如果每行最多有 1KB 数据，每秒要插入 100 行，则必须为表预置 100 个 WCU。如果每行有 1.5KB 数据，则需要两个 WCU 才能每秒插入一行。因此，要每秒插入 100 行，必须预置 200 个 WCU。

要确定每秒插入一行需要多少个 WCU，请将平均行大小（以字节为单位）除以 1024，然后向上舍入到最接近的整数。

例如，如果平均行大小为 3000 字节，则需要三个 WCU 才能每秒插入一行。

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

计算数据加载时间和容量

现在您已经知道了 CSV 文件中的平均行大小和行数，接下来可以计算在给定时间内加载数据需要多少个 WCU，以及使用不同的 WCU 设置在 CSV 文件中加载所有数据所需花费的大致时间。

例如，如果文件中的每行大小为 1KB，而 CSV 文件中有 1000000 行，要在 1 小时内加载数据，则需要为表预置至少 278 个 WCU 才能在 1 小时内完成。

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

配置预置容量设置

您可以在创建表时或使用 ALTER TABLE CQL 命令来设置表的写入容量设置。以下是使用 ALTER TABLE CQL 语句来更改表的预置容量设置的语法。

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

有关完整的语言参考，请参阅 the section called “ALTER TABLE”。

步骤 4：配置 cqlsh COPY FROM 设置

本部分概述如何确定 cqlsh COPY FROM 的参数值。cqlsh COPY FROM 命令读取您之前准备的 CSV 文件，并使用 CQL 将数据插入到 Amazon Keyspaces 中。该命令将行分开，并将 INSERT 操作分配给一组 Worker。每个 Worker 与 Amazon Keyspaces 建立连接并通过该通道发送 INSERT 请求。

cqlsh COPY 命令没有在 Worker 之间均匀分配工作的内部逻辑。但是，您可以手动对其进行配置，以确保均匀分配工作。首先查看以下关键的 cqlsh 参数：

- DELIMITER：如果您使用逗号以外的分隔符，则可以设置此参数，此参数默认为逗号。
- INGESTRATE：cqlsh COPY FROM 每秒尝试处理的目标行数。如果未设置，则默认为 100000。
- NUMPROCESSES：cqlsh 为 COPY FROM 任务创建的子 Worker 进程的数量。此设置的最大值为 16，默认值为 num_cores - 1，其中 num_cores 是运行 cqlsh 的主机上的处理内核数。
- MAXBATCHSIZE：批次大小决定了在单个批次中插入到目标表中的最大行数。如果未设置，cqlsh 将使用插入 20 行的批次。
- CHUNKSIZE：传递给子 Worker 的工作单元的大小。默认情况下，它设置为 5000。
- MAXATTEMPTS：重试失败 Worker 块的最大次数。达到最大尝试次数后，失败记录将写入一个新的 CSV 文件中，您可以在调查失败后再次运行该文件。

根据您为目标表预置的 WCU 数量设置 INGESTRATE。cqlsh COPY FROM 命令的 INGESTRATE 不是限制，而是目标平均值。这意味着它可以（并且经常）突破您设定的数字。要允许暴增并确保有足够的容量来处理数据加载请求，请将 INGESTRATE 设置为表写入容量的 90%。

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

接下来，将 NUMPROCESSES 参数设置为比系统上的内核数少一个。要弄清楚系统的内核数，您可以运行以下代码。

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

在本教程中，我们使用以下值。

```
NUMPROCESSES = 4
```

每个进程都会创建一个 Worker，并且每个 Worker 都会与 Amazon Keyspaces 建立连接。Amazon Keyspaces 在每个连接上每秒可支持最多 3000 个 CQL 请求。这意味着您必须确保每个 Worker 每秒处理的请求少于 3000 个。

与 INGESTRATE 一样，Worker 经常会突破您设置的数字，并且不受时钟秒数的限制。因此，考虑到暴增，请将 cqlsh 参数设置为每个 Worker 每秒处理 2500 个请求。要计算分配给 Worker 的工作量，请使用以下准则。

- INGESTRATE 除以 NUMPROCESSES。
- 如果 INGESTRATE/NUMPROCESSES > 2500，请降低 INGESTRATE 以使此公式成立。

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

在配置设置以优化示例数据的上传之前，让我们回顾一下 cqlsh 默认设置，看看使用它们会如何影响数据上传过程。由于 cqlsh COPY FROM 使用 CHUNKSIZE 创建工作块（INSERT 语句）以分配给 Worker，因此工作不会自动均匀分配。根据 INGESTRATE 设置，有些 Worker 可能会处于闲置状态。

要在 Worker 之间均匀分配工作并使每个 Worker 保持每秒 2500 个请求的最佳速率，必须通过更改输入参数来设置 CHUNKSIZE、MAXBATCHSIZE、和 INGESTRATE。要优化数据加载期间的网络流量利用率，请为 MAXBATCHSIZE 选择一个接近最大值 30 的值。通过将 CHUNKSIZE 更改为 100，将 MAXBATCHSIZE 更改为 25，10000 行将均匀分配给四个 Worker（10000/2500 = 4）。

以下代码示例说明了如何执行此操作。

```
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
```

总而言之，在设置 cqlsh COPY FROM 参数时使用以下公式：

- INGESTRATE = 写入容量单位 * 0.90
- NUMPROCESSES = 内核数 - 1（默认设置）
- INGESTRATE/NUMPROCESSES = 2500（这必须是一个真语句。）
- MAXBATCHSIZE = 30（默认为 20。Amazon Keyspaces 最多可接受 30 个批次。）
- CHUNKSIZE = (INGESTRATE/NUMPROCESSES)/MAXBATCHSIZE

现在您已经计算了 NUMPROCESSES、INGESTRATE 和 CHUNKSIZE，接下来可以加载数据。

步骤 5：运行 cqlsh COPY FROM 命令

要运行 cqlsh COPY FROM 命令，请完成以下步骤。

1. 使用 cqlsh 连接到 Amazon Keyspaces。
2. 使用以下代码选择键空间。

```
USE catalog;
```

3. 将写入一致性设置为 LOCAL_QUORUM。为了确保数据的持久性，Amazon Keyspaces 不允许使用其他写入一致性设置。查看以下代码。

```
CONSISTENCY LOCAL_QUORUM;
```

4. 使用以下代码示例准备 cqlsh COPY FROM 语法。

```
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. 运行上一步中准备的语句。cqlsh 会回显您配置的所有设置。

   a. 确保设置与您的输入相匹配。请参阅以下示例。

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

   b. 查看传输的行数和当前的平均速率，如以下示例所示。

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

   c. 当 cqlsh 完成数据上传后，查看数据加载统计信息（读取的文件数、运行时和跳过的行数）的摘要，如以下示例所示。

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

在本教程的最后一步中，您已将数据上传到 Amazon Keyspaces。

> **Important**
>
> 现在您已经传输了数据，接下来调整目标表的容量模式设置，使其与应用程序的常规流量模式相匹配。在更改之前，您的预置容量按小时费率收费。

故障排除

数据上传完成后，检查是否跳过了行。为此，请导航到源 CSV 文件的源目录并搜索具有以下名称的文件。

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

cqlsh 将所有跳过的数据行写入具有该名称的文件中。如果文件存在于源目录中且其中包含数据，则说明这些行未上传到 Amazon Keyspaces。要重试这些行，请先检查上传过程中是否遇到任何错误，然后相应地调整数据。要重试这些行，您可以重新运行进程。

常见错误

行未加载的最常见原因是容量错误和解析错误。

将数据上传到 Amazon Keyspaces 时出现无效请求错误

在以下示例中，源表包含一个计数器列，该列会生成来自 cqlsh COPY 命令的记录的批处理调用。Amazon Keyspaces 不支持记录的批处理调用。

```
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
```

要解决此错误，可以使用 DSBulk 迁移数据。有关更多信息，请参阅the section called "使用 DSBulk 加载数据"。

将数据上传到 Amazon Keyspaces 时出现解析器错误

以下示例显示了由于 `ParseError` 而跳过的行。

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

要解决此错误，您需要确保要导入的数据与 Amazon Keyspaces 中的表模式相匹配。查看导入文件中是否存在解析错误。您可以尝试通过 INSERT 语句来使用单行数据，从而隔离错误。

将数据上传到 Amazon Keyspaces 时出现容量错误

```
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 使用 ReadTimeout 和 WriteTimeout 异常指示写入请求何时因吞吐容量不足而失败。为了帮助诊断容量不足的异常，Amazon Keyspaces 在亚马逊上发布了WriteThrottleEventsReadThrottledEvents指标。 CloudWatch有关更多信息，请参阅the section called “使用监控 CloudWatch”。

将数据上传到 Amazon Keyspaces 时出现 cqlsh 错误

要帮助对 cqlsh 错误进行问题排查，请重新运行带有 --debug 标志的失败命令。

使用不兼容的 cqlsh 版本时，您会看到以下错误。

```
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
```

通过运行以下命令确认已安装正确版本的 cqlsh。

```
cqlsh --version
```

输出应该类似于以下内容。

```
cqlsh 5.0.1
```

如果您使用的是 Windows，请将 cqlsh 的所有实例替换为 cqlsh.bat。例如，要检查 Windows 中的 cqlsh 版本，请运行以下命令。

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

cqlsh 客户端从服务器连续收到三个任意类型的错误后，与 Amazon Keyspaces 的连接失败。cqlsh 客户端失败时显示以下消息。

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

要解决此错误，您需要确保要导入的数据与 Amazon Keyspaces 中的表模式相匹配。查看导入文件中是否存在解析错误。您可以尝试通过 INSERT 语句来使用单行数据，从而隔离错误。

客户端会自动尝试重新建立连接。

教程：使用 DSBulk 将数据加载到 Amazon Keyspaces

本 step-by-step 教程将指导您使用上提供的 DataStax 批量加载器 (DSBulk) 将数据从 Apache Cassandra 迁移到亚马逊密钥空间。GitHub在本教程中，您完成以下步骤：

主题

先决条件

在开始本教程之前，您必须完成以下任务：

1. 如果您还没有这样做，请按照中的步骤注册一个 AWS 帐户the section called “设置 AWS Identity and Access Management”。
2. 按照the section called “用于 AWS 身份验证的 IAM 证书”中的步骤创建凭证。
3. 创建 JKS 信任存储文件。

   a. 使用以下命令下载 Starfield 数字证书，并将 `sf-class2-root.crt` 保存在本地或您的主目录中。

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

> **Note**
>
> 您还可以使用 Amazon 数字证书连接到 Amazon Keyspaces。如果您的客户端成功连接到 Amazon Keyspaces，您可以继续这样做。Starfield 证书为使用旧证书颁发机构的客户端提供了额外的向后兼容性。

b. 将 Starfield 数字证书转换为 trustStore 文件：

```
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
```

在这一步中，您需要为密钥存储创建一个密码，并信任该证书。交互式命令如下所示。

```
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. 设置 Cassandra 查询语言 Shell (cqlsh) 连接，并按照the section called “使用 cqlsh” 中的步骤确认您可以连接到 Amazon Keyspaces。
5. 下载并安装 DSBulk。

   a. 要下载 DSBulk，您可以使用以下代码。

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

   b. 然后，解压 tar 文件，将 DSBulk 添加到 PATH，如以下示例所示。

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

   c. 创建一个 `application.conf` 文件来存储 DSBulk 要使用的设置。您可以将以下示例保存为 `./dsbulk_keyspaces.conf`。如果您不在本地节点上，请将 `localhost` 替换为本地 Cassandra 集群的联系点，例如 DNS 名称或 IP 地址。记下文件名和路径，因为稍后您需要在 `dsbulk load` 命令中指定这些内容。

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

d. 要启用 SigV4 支持，请从下载阴影jar文件GitHub并将其放入 DSBulk 文件lib夹，如下例所示。

```
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
```

步骤 1：创建源 CSV 文件和目标表

在本教程中，我们使用名为 keyspaces_sample_table.csv 的逗号分隔值 (CSV) 文件作为用于数据迁移的源文件。提供的示例文件包含名为 book_awards 的表中的几行数据。

1. 创建源文件。您可以选择以下选项之一：
   - 下载以下存档文件 samplemigration.zip 中包含的示例 CSV 文件 (keyspaces_sample_table.csv)。解压缩存档文件并记下指向 keyspaces_sample_table.csv 的路径。
   - 要使用您自己存储在 Apache Cassandra 数据库中的数据来填充 CSV 文件，您可以使用 dsbulk unload 来填充源 CSV 文件，如下例所示。

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

     确保您创建的 CSV 文件符合以下要求：

     - 第一行包含列名称。
     - 源 CSV 文件中的列名称与目标表中的列名称相匹配。
     - 数据用逗号分隔。
     - 所有数据值均为有效的 Amazon Keyspaces 数据类型。请参阅 the section called “数据类型”。
2. 在 Amazon Keyspaces 中创建目标键空间和表。

   a. 使用 cqlsh 连接到 Amazon Keyspaces，将以下示例中的服务端点、用户名和密码替换为您自己的值。

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

b. 使用以下示例中所示的名称 catalog 创建新的键空间。

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

c. 在新键空间变为可用状态后，使用以下代码创建目标表 book_awards。要了解有关异步资源创建以及如何检查资源是否可用的更多信息，请参阅the section called “创建键空间”。

```
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)
   );
```

如果 Apache Cassandra 是您的原始数据来源，那么创建带有匹配标题的 Amazon Keyspaces 目标表的一种简单方法是从源表生成 CREATE TABLE 语句，如以下语句所示。

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

然后，在 Amazon Keyspaces 中创建目标表，其列名称和数据类型与 Cassandra 源表中的描述相匹配。

步骤 2：准备数据

为高效传输准备源数据的过程包含两个步骤。第一步，随机化数据。第二步，分析数据以确定相应的 dsbulk 参数值和所需的表设置。

随机化数据

dsbulk 命令按数据在 CSV 文件中显示的顺序读取和写入数据。如果使用 dsbulk 命令创建源文件，将在 CSV 中按键排序顺序写入数据。Amazon Keyspaces 在内部使用分区键对数据进行分区。尽管 Amazon Keyspaces 具有内置逻辑来帮助对针对同一分区键的请求进行负载均衡，但如果您随机排列顺序，则可以更快、更高效地加载数据。这是因为您可以利用 Amazon Keyspaces 在写入不同分区时会出现的内置负载均衡功能。

要将写入操作均匀地分布在分区中，您必须随机化源文件中的数据。您可以编写一个应用程序来执行此操作，也可以使用开源工具来执行此操作，比如 Shuf。Shuf 在 Linux 发行版、macOS（通过在 Homebrew 中安装 coreutils）和 Windows [通过使用 Windows Subsystem for Linux (WSL)] 上免费提供。您还需要执行一个额外步骤来防止包含列名称的标题行在此步骤中被随机排序。

要在保留标题的同时随机化源文件，请输入以下代码。

```
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 将数据重写到名为 `keyspace.table.csv` 的新 CSV 文件中。现在，您可以删除 `keyspaces_sample_table.csv` 文件，您不再需要此文件了。

分析数据

通过分析数据来确定平均行大小和最大行大小。

执行此操作出于以下原因：

- 平均行大小有助于估算要传输的数据总量。
- 您需要平均行大小来预置上传数据所需的写入容量。
- 您可以确保每行的大小小于 1MB，这是 Amazon Keyspaces 中的最大行大小。

> **Note**
>
> 此限额指的是行大小，而不是分区大小。与 Apache Cassandra 分区不同，Amazon Keyspaces 分区实际上可以不受大小限制。分区键和聚类列需要额外的元数据存储空间，您必须将其加到行的原始大小中。有关更多信息，请参阅 the section called “计算行大小”。

以下代码使用 AWK 分析 CSV 文件并打印平均行大小和最大行大小。

```
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
```

运行此代码将生成以下输出。

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

确保最大行大小不超过 1MB。否则，必须拆分行或压缩数据，使行大小小于 1MB。在本教程的下一步中，您将使用平均行大小来预置表的写入容量。

步骤 3：为表设置吞吐容量

本教程向您展示了如何调整 DSBulk 以在设定的时间范围内加载数据。由于您提前知道自己要执行多少读取和写入操作，因此可以使用预置容量模式。完成数据传输后，应该将表的容量模式设置为与应用程序的流量模式相匹配。要了解有关容量管理的更多信息，请参阅 无服务器资源管理。

使用预置容量模式，您可以提前指定要为表预置多少读取和写入容量。写入容量按小时计费，并以写入容量单位 (WCU) 计量。每个 WCU 的写入容量足以支持每秒写入 1KB 数据。加载数据时，写入速率必须低于目标表上设置的最大 WCU（参数：`write_capacity_units`）。

默认情况下，您可以为一个表预置最多 40000 个 WCU，可以为账户中的所有表预置最多 80000 个 WCU。如果您需要更多容量，可以在服务限额控制台中请求提高限额。有关限额的更多信息，请参阅限额。

计算一次插入所需的平均 WCU 数量

每秒插入 1KB 数据需要 1 个 WCU。如果您的 CSV 文件有 360000 行，并且您想在 1 小时内加载所有数据，则必须每秒写入 100 行（360000 行/60 分/60 秒 = 每秒 100 行）。如果每行最多有 1KB 数据，每秒要插入 100 行，则必须为表预置 100 个 WCU。如果每行有 1.5KB 数据，则需要两个 WCU 才能每秒插入一行。因此，要每秒插入 100 行，必须预置 200 个 WCU。

要确定每秒插入一行需要多少个 WCU，请将平均行大小（以字节为单位）除以 1024，然后向上舍入到最接近的整数。

例如，如果平均行大小为 3000 字节，则需要三个 WCU 才能每秒插入一行。

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

计算数据加载时间和容量

现在您已经知道了 CSV 文件中的平均行大小和行数，接下来可以计算在给定时间内加载数据需要多少个 WCU，以及使用不同的 WCU 设置在 CSV 文件中加载所有数据所需花费的大致时间。

例如，如果文件中的每行大小为 1KB，而 CSV 文件中有 1000000 行，要在 1 小时内加载数据，则需要为表预置至少 278 个 WCU 才能在 1 小时内完成。

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

配置预置容量设置

您可以在创建表时或使用 ALTER TABLE 命令来设置表的写入容量设置。以下是使用 ALTER TABLE 命令来更改表的预置容量设置的语法。

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

有关完整的语言参考，请参阅 the section called “CREATE TABLE” 和 the section called “ALTER TABLE”。

步骤 4：配置 DSBulk 设置

本部分概述配置 DSBulk 以将数据上传到 Amazon Keyspaces 所需的步骤。您可以使用配置文件来配置 DSBulk。您可以直接从命令行指定配置文件。

1. 为迁移到 Amazon Keyspaces 创建一个 DSBulk 配置文件，在此示例中，我们使用的文件名是 dsbulk_keyspaces.conf。在 DSBulk 配置文件中指定以下设置。

   a. *PlainTextAuthProvider*：使用 PlainTextAuthProvider 类创建身份验证提供者。ServiceUserName 和 ServicePassword 应该与您按照the section called “创建凭证”中的步骤生成特定于服务的凭证时获得的用户名和密码相匹配。

   b. *local-datacenter*— 将的值设置 AWS 区域 为local-datacenter要连接的。例如，如果应用程序要连接到 cassandra.us-east-2.amazonaws.com，则将本地数据中心设置为 us-east-2。有关所有可用信息 AWS 区域，请参阅the section called “服务端点”。为了避免复制，请将 slow-replica-avoidance 设置为 false。

   c. *SSLEngineFactory*：要配置 SSL/TLS，初始化 SSLEngineFactory，方法是在配置文件中添加一个部分，其中只有一行，用于指定类 class = DefaultSslEngineFactory。提供指向 cassandra_truststore.jks 的路径和您之前创建的密码。

   d. *consistency*：将一致性级别设置为 LOCAL_QUORUM。不支持其他写入一致性级别，有关更多信息，请参阅 the section called “支持的 Cassandra 一致性级别”。

   e. 可以在 Java 驱动程序中配置每个池的连接数。在此示例中，将 advanced.connection.pool.local.size 设置为 3。

以下是完整的示例配置文件。

```
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. 查看 DSBulk load 命令的参数。

   a. *executor.maxPerSecond*：load 命令尝试每秒同时处理的最大行数。如果未设置，则使用 -1 禁用此设置。

      根据您为目标表预置的 WCU 数量设置 executor.maxPerSecond。load 命令的 executor.maxPerSecond 不是限制，而是目标平均值。这意味着它可以（并且经常）突破您设定的数字。要允许暴增并确保有足够的容量来处理数据加载请求，请将 executor.maxPerSecond 设置为表写入容量的 90%。

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

在本教程中，我们将 executor.maxPerSecond 设置为 5。

> **Note**
>
> 如果您使用的是 DSBulk 1.6.0 或更高版本，则可以改用 dsbulk.engine.maxConcurrentQueries。

b. 为 DSBulk load 命令配置以下这些附加参数。

- *batch-mode*：此参数告诉系统按分区键对操作进行分组。我们建议禁用批处理模式，因为它可能会导致热键场景和原因WriteThrottleEvents。
- *driver.advanced.retry-policy-max-retries*：这决定了重试失败查询的次数。如果未设置，则默认值为 10。您可以根据需要调整此值。
- *driver.basic.request.timeout*：系统等待查询返回的时间（以分钟为单位）。如果未设置，则默认值为“5 分钟”。您可以根据需要调整此值。

步骤 5：运行 DSBulk **load** 命令

在本教程的最后一步中，您要将数据上传到 Amazon Keyspaces。

要运行 DSBulk load 命令，请完成以下步骤。

1. 运行以下代码，将您的 csv 文件中的数据上传到 Amazon Keyspaces 表中。请确保更新指向您之前创建的应用程序配置文件的路径。

```
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. 输出包括详细说明成功和不成功操作的日志文件的位置。该文件存储在以下目录中。

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

3. 日志文件条目将包括指标，如以下示例所示。检查以确保行数与 csv 文件中的行数一致。

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

> **Important**
>
> 现在您已经传输了数据，接下来调整目标表的容量模式设置，使其与应用程序的常规流量模式相匹配。在更改之前，您的预置容量按小时费率收费。有关更多信息，请参阅 the section called “读/写容量模式”。

使用软件开发工具包的 Amazon Keyspaces 的代码示例 AWS

以下代码示例展示了如何将 Amazon Keyspaces 与 AWS 软件开发套件 (SDK) 一起使用。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

开始使用

Hello Amazon Keysp

以下代码示例展示了如何开始使用 Amazon Keyspaces。

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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();
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考ListKeyspaces中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListKeyspaces中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/**
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}")
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用ListKeyspaces于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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"))
```

- 有关 API 的详细信息，请参阅适用ListKeyspaces于 Python 的AWS SDK (Boto3) API 参考。

代码示例

- 使用软件开发工具包对 Amazon Keyspaces 执行的操作 AWS
  - CreateKeyspace与 S AWS DK 或命令行工具配合使用

- CreateTable与 S AWS DK 或命令行工具配合使用
- DeleteKeyspace与 S AWS DK 或命令行工具配合使用
- DeleteTable与 S AWS DK 或命令行工具配合使用
- GetKeyspace与 S AWS DK 或命令行工具配合使用
- GetTable与 S AWS DK 或命令行工具配合使用
- ListKeyspaces与 S AWS DK 或命令行工具配合使用
- ListTables与 S AWS DK 或命令行工具配合使用
- RestoreTable与 S AWS DK 或命令行工具配合使用
- UpdateTable与 S AWS DK 或命令行工具配合使用

- 使用软件开发工具包的 Amazon Keyspaces 场景 AWS
  - 开始使用软件开发工具包使用 Amazon Keyspaces 密钥空间和表 AWS

使用软件开发工具包对 Amazon Keyspaces 执行的操作 AWS

以下代码示例演示了如何使用软件开发工具包执行各个 Amazon Keyspaces 操作。 AWS 这些代码节选调用了 Amazon Keyspaces API，是必须在上下文中运行的较大型程序的代码节选。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。有关完整列表，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）API 参考。

示例

- CreateKeyspace与 S AWS DK 或命令行工具配合使用
- CreateTable与 S AWS DK 或命令行工具配合使用
- DeleteKeyspace与 S AWS DK 或命令行工具配合使用
- DeleteTable与 S AWS DK 或命令行工具配合使用
- GetKeyspace与 S AWS DK 或命令行工具配合使用
- GetTable与 S AWS DK 或命令行工具配合使用
- ListKeyspaces与 S AWS DK 或命令行工具配合使用
- ListTables与 S AWS DK 或命令行工具配合使用
- RestoreTable与 S AWS DK 或命令行工具配合使用
- UpdateTable与 S AWS DK 或命令行工具配合使用

CreateKeyspace与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 CreateKeyspace。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考CreateKeyspace中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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);
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CreateKeyspace中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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}")
    }
}
```

- 有关 API 的详细信息，请参阅适用CreateKeyspace于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
```

- 有关 API 的详细信息，请参阅适用CreateKeyspace于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`CreateTable`与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 `CreateTable`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考CreateTable中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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);
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CreateTable中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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}")
    }
}
```

- 有关 API 的详细信息，请参阅适用CreateTable于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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"]
```

- 有关 API 的详细信息，请参阅适用CreateTable于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteKeyspace`与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 `DeleteKeyspace`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DeleteKeyspace中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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);
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteKeyspace中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun deleteKeyspace(keyspaceNameVal: String?) {
    val deleteKeyspaceRequest = DeleteKeyspaceRequest {
        keyspaceName = keyspaceNameVal
    }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient.deleteKeyspace(deleteKeyspaceRequest)
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteKeyspace于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
```

- 有关 API 的详细信息，请参阅适用DeleteKeyspace于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DeleteTable与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 DeleteTable。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DeleteTable中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteTable中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun deleteTable(keyspaceNameVal: String?, tableNameVal: String?) {
    val tableRequest = DeleteTableRequest {
        keyspaceName = keyspaceNameVal
        tableName = tableNameVal
    }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient.deleteTable(tableRequest)
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteTable于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
```

- 有关 API 的详细信息，请参阅适用DeleteTable于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

GetKeyspace与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 GetKeyspace。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[GetKeyspace](#)中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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);
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考[GetKeyspace](#)中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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")
    }
}
```

- 有关 API 的详细信息，请参阅适用GetKeyspace于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
```

- 有关 API 的详细信息，请参阅适用GetKeyspace于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetTable`与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 `GetTable`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考GetTable中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考GetTable中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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}")
```

```
            }
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用GetTable于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
```

- 有关 API 的详细信息，请参阅适用GetTable于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

ListKeyspaces与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 ListKeyspaces。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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}");
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考ListKeyspaces中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListKeyspaces中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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}")
            }
    }
```

```
}
```

- 有关 API 的详细信息，请参阅适用ListKeyspaces于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
```

- 有关 API 的详细信息，请参阅适用ListKeyspaces于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListTables`与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 `ListTables`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考ListTables中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListTables中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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.toString() +
                        " Table name: " + obj.tableName
                )
            }
    }
}
```

- 有关 API 的详细信息，请参阅适用ListTables于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
```

- 有关 API 的详细信息，请参阅适用ListTables于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`RestoreTable`与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 `RestoreTable`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考RestoreTable中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    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);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考RestoreTable中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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}")
```

```
    }
}
```

- 有关 API 的详细信息，请参阅适用RestoreTable于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
```

- 有关 API 的详细信息，请参阅适用RestoreTable于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

UpdateTable与 S AWS DK 或命令行工具配合使用

以下代码示例演示如何使用 UpdateTable。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用密钥空间和表

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考UpdateTable中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   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);
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考UpdateTable中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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)
    }
}
```

- 有关 API 的详细信息，请参阅适用UpdateTable于 K otlin 的AWS SDK API 参考。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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
```

- 有关 API 的详细信息，请参阅适用UpdateTable于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包的 Amazon Keyspaces 场景 AWS

以下代码示例向您展示了如何使用软件开发工具包在 Amazon Keyspaces 中实现常见场景。 AWS 这些场景显示了如何通过在 Amazon Keyspaces 中调用多个函数来完成特定任务。每个场景都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行代码的说明。

示例

- 开始使用软件开发工具包使用 Amazon Keyspaces 密钥空间和表 AWS

开始使用软件开发工具包使用 Amazon Keyspaces 密钥空间和表 AWS

以下代码示例显示了如何：

- 创建密钥空间和表。表架构保存电影数据并启用了 point-in-time 恢复。
- 使用带有 Sigv4 身份验证的安全 TLS 连接连接到密钥空间。
- 查询表。添加、检索和更新电影数据。
- 更新表。添加一列来跟踪观看的电影。
- 将表还原到以前的状态并清理资源。

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
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!;
        }
    }
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的以下主题。
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces

- ListTables
- RestoreTable
- UpdateTable

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/**
 * 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);
        }
    }
}
```

- 有关 API 详细信息，请参阅 AWS SDK for Java 2.x API 参考中的以下主题。
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/**
 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.toString() +
                        " 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}")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的以下主题。
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
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.")
```

定义一个包装密钥空间和表操作的类。

```
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
```

定义一个类，用于创建与密钥空间的 TLS 连接、使用 SigV4 进行身份验证，并向密钥空间中的表发送 CQL 查询。

```
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],
        )
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Keyspaces 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

Amazon Keyspaces（Apache Cassandra 兼容）库和工具

本节提供有关 Amazon Keyspaces（Apache Cassandra 兼容）库、代码示例和工具的信息。

主题

库和示例

您可以在 GitHub 上的 AWS 和 AWS samples 存储库中找到 Amazon Keyspaces 开源库和开发人员工具。

Amazon Keyspaces（Apache Cassandra 兼容）开发人员工具包

此存储库提供了 Docker 映像，其中包含适用于 Amazon Keyspaces 的实用开发人员工具。例如，它包括包含最佳实践的 CQLSHRC 文件、用于 cqlsh 的可选 AWS 身份验证扩展，以及用于执行常见任务的助手工具。该工具包针对 Amazon Keyspaces 进行了优化，但也适用于 Apache Cassandra 集群。

https://github.com/aws-samples/amazon-keyspaces-toolkit.

Amazon Keyspaces（Apache Cassandra 兼容）示例

此存储库是我们的 Amazon Keyspaces 示例代码的官方列表。存储库按语言细分为几个部分（参见 Examples）。每种语言都有自己的示例子部分。这些示例演示了常见的 Amazon Keyspaces 服务实现和模式，您可以在构建应用程序时使用。

https://github.com/aws-samples/amazon-keyspaces-examples/.

AWS 签名版本 4 (SigV4) 身份认证插件

借助这些插件，您可以使用 AWS Identity and Access Management (IAM) 用户和角色来管理对 Amazon Keyspaces 的访问。

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。

重点介绍的示例和开发人员工具存储库

下面是精选的适用于 Amazon Keyspaces（Apache Cassandra 兼容）的实用社区工具。

Amazon Keyspaces 协议缓冲区

您可以将协议缓冲区 (Protobuf) 与 Amazon Keyspaces 配合使用，以提供 Apache Cassandra User Defined Types (UDT) 的替代方案。Protobuf 是一种免费的开源跨平台数据格式，用于序列化结构化数据。您可以使用 CQL BLOB 数据类型存储 Protobuf 数据并重构 UDT，同时跨应用程序和编程语言保留结构化数据。

此存储库提供了一个代码示例，用于连接到 Amazon Keyspaces、创建新表并插入包含 Protobuf 消息的行，然后以强一致性读取该行。

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/protobuf-user-defined-types

为 Amazon Keyspaces（Apache Cassandra 兼容）指标创建 Amazon CloudWatch 控制面板的 AWS CloudFormation 模板

此存储库提供了用于快速设置 Amazon Keyspaces 的 CloudWatch 指标的 AWS CloudFormation 模板。使用此模板可以提供包含常用指标的可部署的预构建 CloudWatch 控制面板，让您更容易上手。

https://github.com/aws-samples/amazon-keyspaces-cloudwatch-cloudformation-templates.

将 Amazon Keyspaces（Apache Cassandra 兼容）与 AWS Lambda 结合使用

该存储库包含一些示例，展示了如何从 Lambda 连接到 Amazon Keyspaces。下面是一些示例。

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。

另一个展示如何从 Python Lambda 部署和使用 Amazon Keyspaces 的 Lambda 示例可从以下存储库中获得。

https://github.com/aws-samples/aws-keyspaces-lambda-python

将 Amazon Keyspaces（Apache Cassandra 兼容）与 Spring 结合使用

此示例向您展示了如何将 Amazon Keyspaces 与 Spring Boot 结合使用。

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/spring

将 Amazon Keyspaces（Apache Cassandra 兼容）与 Scala 结合使用

此示例展示了如何使用基于 Scala 的 SigV4 身份验证插件连接到 Amazon Keyspaces。

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/scala/datastax-v4/connection-sigv4

将 Amazon Keyspaces（Apache Cassandra 兼容）与 AWS Glue 结合使用

此示例展示了如何将 Amazon Keyspaces 与 AWS Glue 结合使用。

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/scala/datastax-v4/aws-glue

Amazon Keyspaces（Apache Cassandra 兼容）Cassandra query language (CQL) 到 AWS CloudFormation 的转换器

该软件包实现了一种命令行工具，用于将 Apache Cassandra Query Language (CQL) 脚本转换为 AWS CloudFormation (CloudFormation) 模板，该工具允许在 CloudFormation 堆栈中轻松管理 Amazon Keyspaces 架构。

https://github.com/aws/amazon-keyspaces-cql-to-cfn-converter.

Java 版 Apache Cassandra 驱动程序的 Amazon Keyspaces（Apache Cassandra 兼容）助手

此存储库包含将 DataStax Java 驱动程序用于 Amazon Keyspaces（Apache Cassandra 兼容）时的驱动程序策略、示例和最佳实践。

https://github.com/aws-samples/amazon-keyspaces-java-driver-helpers.

Amazon Keyspaces（Apache Cassandra 兼容）快速压缩演示

此存储库演示了如何压缩、存储和读取/写入大型对象，以提高性能，降低吞吐量和存储成本。

https://github.com/aws-samples/amazon-keyspaces-compression-example.

Amazon Keyspaces（Apache Cassandra 兼容）和 Amazon S3 编解码器演示

自定义 Amazon S3 编解码器支持 UUID 指针到 Amazon S3 对象的透明、用户可配置的映射。

https://github.com/aws-samples/amazon-keyspaces-large-object-s3-demo.

将 Amazon Keyspaces 与 Apache Spark 集成

Apache Spark 是一款用于大规模数据分析的开源引擎。Apache Spark 让您能够更有效地分析存储在 Amazon Keyspaces 中的数据。您还可以使用 Amazon Keyspaces 为应用程序提供对 Spark 中的分析数据的毫秒级一致读取权限。开源 Spark Cassandra Connector 可以简化 Amazon Keyspaces 和 Spark 之间的数据读写。

Amazon Keyspaces 可以使用完全托管的无服务器数据库服务来简化 Cassandra 工作负载在基于 Spark 的分析管道中的运行，从而对 Spark Cassandra Connector 提供支持。有了 Amazon Keyspaces，您无需担心 Spark 会与您的表争夺底层基础设施资源。Amazon Keyspaces 表会根据您的应用程序流量自动扩缩。

以下教程将会介绍使用 Spark Cassandra Connector 向 Amazon Keyspaces 读取和写入数据所需的步骤和最佳实践。本教程演示了如何使用 Spark Cassandra Connector 从文件中加载数据并将其写入 Amazon Keyspaces 表，从而将数据迁移到 Amazon Keyspaces。然后，本教程展示了如何使用 Spark Cassandra Connector 从 Amazon Keyspaces 读回数据。进行这一操作的目的是在基于 Spark 的分析管道中运行 Cassandra 工作负载。

主题

使用 Spark Cassandra Connector 建立与 Amazon Keyspaces 的连接的先决条件

在使用 Spark Cassandra Connector 连接到 Amazon Keyspaces 之前，您需要确保安装以下内容。Amazon Keyspaces 与 Spark Cassandra Connector 的兼容性已通过以下推荐版本进行了测试：

- Java 版本 8
- Scala 2.12

- Spark 3.4
- Cassandra Connector 2.5 及更高版本
- Cassandra 驱动程序 4.12

1. 要安装 Scala，请按照 https://www.scala-lang.org/download/scala2.html 中的说明进行操作。
2. 要安装 Spark 3.4.1，请按照以下示例进行操作。

```
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
```

第 1 步：配置 Amazon Keyspaces 以便与 Apache Cassandra Spark Connector 集成

在本步骤中，您将确认您的账户的分区程序是否与 Apache Spark Connector 兼容，并设置所需的 IAM 权限。以下最佳实践可以帮助您为表预配置足够的读/写容量。

1. 确认 `Murmur3Partitioner` 分区程序是您账户的默认分区程序。该分区程序与 Spark Cassandra Connector 兼容。有关分区程序以及如何更改分区程序的更多信息，请参阅the section called “使用分区程序”。
2. 使用 Apache Spark 和接口 VPC 端点为 Amazon Keyspaces 设置 IAM 权限。
   - 分配对用户表的读/写权限和对系统表的读取权限，如下面列出的 IAM 策略示例所示。
   - 使用 Spark 通过 VPC 端点访问 Amazon Keyspaces 的客户端需要在 system.peers 表中填充您的可用接口 VPC 端点。

```
{
   "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. 请考虑以下最佳实践，为您的 Amazon Keyspaces 表配置足够的读/写吞吐容量，以支持来自 Spark Cassandra Connector 的流量。

   - 开始使用按需容量来帮助您测试场景。
   - 要优化生产环境的表吞吐量成本，请对来自 Connector 的流量使用速率限制器，并将您的表配置为使用具有自动扩缩功能的预配置容量。有关更多信息，请参阅the section called “使用 auto scaling 管理吞吐容量”。
   - 你可以使用 Cassandra 驱动程序附带的固定速率限制器。AWS 示例存储库中有一些为 Amazon Keyspaces 量身定制的速率限制器。
   - 有关容量管理的更多信息，请参阅the section called “读/写容量模式”。

步骤 2：配置 Apache Cassandra Spark Connector

Apache Spark 是一种通用计算平台，可以用不同的方式进行配置。要配置 Spark 和 Spark Cassandra Connector 以便与 Amazon Keyspaces 集成，我们建议您从下一节介绍的最低配置设置开始，然后根据您的工作负载提高配置设置。

- 创建小于 8 MB 的 Spark 分区。

  在 Spark 中，分区代表可以并行运行的原子数据块。当您使用 Spark Cassandra Connector 向 Amazon Keyspaces 写入数据时，Spark 分区越小，任务要写入的记录量就越小。如果 Spark 任务遇到多个错误，则在进行指定次数的重试后，任务将会失败。为避免重新执行大型任务和重新处理大量数据，请将 Spark 分区的大小保持在较小的范围内。

- 每个执行程序采用较低的并发写入次数和较高的重试次数。

  如果操作超时，Amazon Keyspaces 会向 Cassandra 驱动程序返回容量不足错误。您无法通过更改配置的超时持续时间来解决因容量不足而导致的超时问题，因为 Spark Cassandra Connector 会尝试使用 `MultipleRetryPolicy` 以透明方式重试请求。为确保重试不会让驱动程序的连接池不堪重负，请让每个执行程序采用较低的并发写入次数和较高的重试次数。以下代码段是一个这方面的示例。

  ```
  spark.cassandra.query.retry.count = 500
  spark.cassandra.output.concurrent.writes = 3
  ```

- 分解总吞吐量并将其分配到多个 Cassandra 会话中。
  - Cassandra Spark Connector 会为每个 Spark 执行程序创建一个会话。我们可以把这一会话视为确定所需吞吐量和所需连接数量的扩展单元。
  - 在定义每个执行程序的内核数和每个任务的内核数时，请从低开始并根据需要增加数量。
  - 设置 Spark 任务失败次数以便在出现暂时错误时继续处理。在您熟悉应用程序的流量特征和要求后，我们建议将 `spark.task.maxFailures` 设置为一个有界值。
  - 例如，以下配置可以让每个执行程序在每个会话中处理两个并发任务：

    ```
    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
    ```

- 关闭批处理。
  - 我们建议您关闭批处理以改进随机访问模式。以下代码段是一个这方面的示例。

    ```
    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)
    ```

- 将 **`SPARK_LOCAL_DIRS`** 设置为具有足够空间的本地高速磁盘。

- 默认情况下，Spark 会将地图输出文件和弹性分布式数据集 (RDD) 保存到/tmp 文件夹。根据您的 Spark 主机的配置，这可能会导致设备没有剩余空间这一类型的错误。
- 要将 SPARK_LOCAL_DIRS 环境变量设置为名为 /example/spark-dir 的目录，您可以使用以下命令。

```
export SPARK_LOCAL_DIRS=/example/spark-dir
```

步骤 3：创建应用程序配置文件

要将开源 Spark Cassandra Connector 与 Amazon Keyspaces 配合使用，您需要提供一个应用程序配置文件，其中包含连接 DataStax Java 驱动程序所需的设置。您可以使用特定于服务的凭证或 SigV4 插件进行连接。

如果您尚未执行这一操作，则需要将 Starfield 数字证书转换为 trustStore 文件。您可以按照 Java 驱动程序连接教程的the section called “开始前的准备工作”中的详细步骤进行操作。请记下 trustStore 文件路径和密码，因为您在创建应用程序配置文件时需要使用这一信息。

使用 Sigv4 身份验证进行连接

本节向您展示了一个示例 application.conf 文件，您可以在连接 AWS 凭证和 SigV4 插件时使用这一文件。如果您尚未执行这一操作，则需要生成 IAM 访问密钥（一个访问密钥 ID 和一个私密访问密钥），并将其保存在 AWS 配置文件中或作为环境变量保存。有关详细说明，请参阅the section called “ AWS 身份验证所需的凭据”。

在以下示例中，请将文件路径替换为您的 trustStore 文件，并且替换密码。

```
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
}
```

更新此配置文件并将其保存为 /home/user1/application.conf。下列示例使用这一路径。

使用特定于服务的凭证进行连接

本节向您展示了一个示例 application.conf 文件，您可以在使用特定于服务的凭证进行连接时使用这一文件。如果您尚未执行这一操作，则需要为 Amazon Keyspaces 生成特定于服务的凭证。有关详细说明，请参阅the section called “服务特定凭证”。

在以下示例中，将 username 和 password 替换为您自己的凭证。另外，请将文件路径替换为您的 trustStore 文件，并且替换密码。

```
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
                }
            }
        }
}
```

更新此配置文件并将其保存为 /home/user1/application.conf，以便与代码示例一起使用。

设置固定连接速率

要强制每个 Spark 执行程序都使用固定速率，您可以定义一个请求调节器。请求调节器可以限制每秒的请求速率。Spark Cassandra Connector 会为每个执行程序都部署一个 Cassandra 会话。使用以下公式可以帮助您在表中实现一致的吞吐量。

```
max-request-per-second * numberOfExecutors = total throughput against a table
```

您可以将此示例添加到之前创建的应用程序配置文件中。

```
datastax-java-driver {
  advanced.throttler {
    class = RateLimitingRequestThrottler

    max-requests-per-second = 3000
    max-queue-size = 30000
    drain-interval = 1 millisecond
  }
}
```

步骤 4：在 Amazon Keyspaces 中准备源数据和目标表

在本步骤中，您将创建一个包含示例数据和 Amazon Keyspaces 表的源文件。

1. 创建源文件。您可以选择以下选项之一：
   - 在本教程中，您会使用名为 keyspaces_sample_table.csv 的逗号分隔值 (CSV) 文件作为数据迁移的源文件。提供的示例文件包含名为 book_awards 的表中的几行数据。

- 下载以下存档文件 samplemigration.zip 中包含的示例 CSV 文件 (keyspaces_sample_table.csv)。解压缩存档文件并记下指向 keyspaces_sample_table.csv 的路径。

- 如果您想使用自己的 CSV 文件将数据写入 Amazon Keyspaces，请确保数据经过随机化处理。直接从数据库读取或导出到平面文件的数据通常按分区和主键排序。将有序的数据导入 Amazon Keyspaces 可能会导致数据被写入较小的 Amazon Keyspaces 分区，从而造成流量分布不均匀。这可能会导致性能降低和错误率上升。

  相比之下，将数据随机化有助于更均匀地在分区之间分配流量，从而利用 Amazon Keyspaces 内置的负载均衡功能。您可以使用多种工具将数据随机化。有关使用开源工具 Shuf 的示例，请参阅数据迁移教程中的the section called “步骤 2：准备数据”。以下示例展示了如何将数据随机处理为 DataFrame。

```
import org.apache.spark.sql.functions.randval
shuffledDF = dataframe.orderBy(rand())
```

2. 在 Amazon Keyspaces 中创建目标键空间和表。

   a. 使用 cqlsh 连接到 Amazon Keyspaces，并将以下示例中的服务端点、用户名和密码替换成您自己的值。

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

   b. 使用以下示例中所示的名称 catalog 创建新的键空间。

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

   c. 在新键空间变为可用状态后，使用以下代码创建目标表 book_awards。要了解有关异步资源创建以及如何检查资源是否可用的更多信息，请参阅the section called “创建键空间”。

```
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)
```

```
    );
```

步骤 5：使用 Apache Cassandra Spark Connector 写入和读取 Amazon Keyspaces 数据

在本步骤中，您首先需要使用 Spark Cassandra Connector 将示例文件中的数据加载到 DataFrame 中。接下来，您需要将数据从 DataFrame 写入您的 Amazon Keyspaces 表中。您也可以单独使用这一部分执行其他操作，比如将数据迁移到 Amazon Keyspaces 表中。最后，您需要使用 Spark Cassandra Connector 将表中的数据读入到一个 DataFrame 中。您也可以单独使用这一部分执行其他操作，比如从 Amazon Keyspaces 表中读取数据，以便使用 Apache Spark 执行数据分析。

1. 启动 Spark Shell，如以下示例所示。请注意，此示例使用的是 Sigv4 身份验证。

```
./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. 使用以下代码导入 Spark Cassandra Connector。

```
import org.apache.spark.sql.cassandra._
```

3. 要从 CSV 文件中读取数据并将其存储在 DataFrame 中，您可以使用以下示例代码。

```
var df =
 spark.read.option("header","true").option("inferSchema","true").csv("keyspaces_sample_tabl
```

   您可以使用以下命令来显示结果。

```
scala> df.show();
```

   输出应如下所示：

```
+----------------+----+-----------+----+------------------+--------------------
+-------------+
|           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|
+----------------+----+-----------+----+------------------+--------------------
+-------------+
```

您可以确认 DataFrame 中的数据的架构，如以下示例所示。

```
scala> df.printSchema
```

输出应如下所示：

```
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. 使用以下命令将 DataFrame 中的数据写入 Amazon Keyspaces 表中。

```
df.write.cassandraFormat("book_awards", "catalog").mode("APPEND").save()
```

5. 要确认数据已保存，您可以将其读回到一个数据框，如以下示例所示。

```
var newDf = spark.read.cassandraFormat("book_awards", "catalog").load()
```

然后，你可以显示现在包含在数据框中的数据。

```
scala> newDf.show()
```

该命令应生成如下所示的输出。

```
+--------------------+-----------------+---------------+----------+-------------
+----+----+
|          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|
+--------------------+-----------------+---------------+----------+-------------
+----+----+
```

排除将 Spark Cassandra Connector 与 Amazon Keyspaces 配合使用时的常见错误

如果您在使用 Amazon Virtual Private Cloud 并连接到 Amazon Keyspaces，则在使用 Spark 连接器时遇到的最常见的错误往往由以下配置问题引起。

- VPC 中使用的 IAM 用户或角色缺少访问 Amazon Keyspaces 中的 `system.peers` 表所需的权限。有关更多信息，请参阅the section called “使用接口 VPC 端点信息填充 `system.peers` 表条目”。
- IAM 用户或角色缺少对用户表的读/写权限，也缺少对 Amazon Keyspaces 中的系统表的读取权限。有关更多信息，请参阅the section called “步骤 1：配置 Amazon Keyspaces”。
- 创建 SSL/TLS 连接时，Java 驱动程序配置没有禁用主机名验证。有关示例，请参阅 the section called “步骤 2：配置驱动程序”。

有关详细的链接故障排除步骤，请参阅the section called “VPC 端点连接错误”。

此外，您可以使用 Amazon CloudWatch 指标来帮助您解决 Amazon Keyspaces 中的 Spark Cassandra Connector 配置问题。要了解有关将 Amazon Keyspaces 与 CloudWatch 结合使用的更多信息，请参阅the section called “使用监控 CloudWatch”。

以下部分介绍了在使用 Spark Cassandra Connector 时需要观察的最有用的指标。

PerConnectionRequestRateExceeded

Amazon Keyspaces 的配额为每个连接每秒 3000 个请求。每个 Spark 执行程序都会与 Amazon Keyspaces 建立连接。多次重试可能会耗尽每个连接的请求速率配额。如果您超出此配额，Amazon Keyspaces 会在 CloudWatch 中发布一个 `PerConnectionRequestRateExceeded` 指标。

如果您看到 PerConnectionRequestRateExceeded 事件以及其他系统或用户错误，则很可能是 Spark 在进行多次重试，超出了每个连接分配的请求数。

如果您看到 `PerConnectionRequestRateExceeded` 事件但没有看到其他错误，则您可能需要增加驱动程序设置中的连接数以提高吞吐量，或者可能需要增加 Spark 作业中的执行程序数量。

StoragePartitionThroughputCapacityExceeded

Amazon Keyspaces 的配额为每秒 1000 个 WCU 或 WRU /每个分区每秒 3000 个 RCU 或 RRU。如果您看到 `StoragePartitionThroughputCapacityExceeded` CloudWatch 事件，则可能表

明加载的数据没有经过随机化处理。有关如何随机处理数据的示例，请参阅the section called “步骤 4：在 Amazon Keyspaces 中准备源数据和目标表”。

常见错误和警告

如果您在使用 Amazon Virtual Private Cloud 并连接到 Amazon Keyspaces，则 Cassandra 驱动程序可能会在 `system.peers` 表中发出有关控制节点本身的警告消息。有关更多信息，请参阅the section called “常见错误和警告”。您可以放心地忽略这一警告。

Amazon Keyspaces（Apache Cassandra 兼容）故障排除

以下各节介绍如何在使用 Amazon Keyspaces（Apache Cassandra 兼容）时对可能遇到的常见配置问题进行故障排除。

有关专门针对 IAM 访问的故障排除指南，请参阅the section called “故障排除”。

如需了解有关安全性最佳实践的更多信息，请参阅 the section called “安全最佳实践”。

主题

Amazon Keyspaces 中的连接故障排除

连接时遇到问题？ 以下介绍一些常见问题以及如何解决这些问题。

连接到 Amazon Keyspaces 端点时出错

连接失败和连接错误会导致不同的错误信息。以下部分涵盖了最常见的场景。

主题

我无法使用 cqlsh 连接到 Amazon Keyspaces

您尝试使用 cqlsh 连接到 Amazon Keyspaces 端点，但连接失败并显示 **`Connection error`**。

如果您尝试连接到 Amazon Keyspaces 表，但 cqlsh 配置不正确，连接将失败。以下部分举例说明了在使用 cqlsh 尝试建立连接时导致连接错误的最常见配置问题。

> **Note**
>
> 如果您尝试从 VPC 连接到 Amazon Keyspaces，需要额外的权限。要使用 VPC 端点成功配置连接，请按照the section called “连接 VPC 终端节点” 中的步骤操作。

您尝试使用 cqlsh 连接到 Amazon Keyspaces，但出现连接 **`timed out`** 错误。

如果您没有提供正确的端口，则可能会出现这种情况，从而导致以下错误。

```
#  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")})
```

要解决此问题，请确认使用端口 9142 进行连接。

您尝试使用 cqlsh 连接到 Amazon Keyspaces，但出现 **`Name or service not known`** 错误。

如果您使用的端点拼写错误或不存在，则可能出现这种情况。在以下示例中，端点名称存在拼写错误。

```
#  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
```

要在使用公共端点连接时解决此问题，请从 the section called “服务端点”中选择一个可用的端点，并确认该端点的名称没有任何错误。如果使用 VPC 端点进行连接，请验证 cqlsh 配置中的 VPC 端点信息是否正确。

您尝试使用 cqlsh 连接 Amazon Keyspaces，但收到 **`OperationTimedOut`** 错误。

Amazon Keyspaces 要求为连接启用 SSL，以确保强大的安全性。如果您收到以下错误，则可能是缺少 SSL 参数。

```
# 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',)})
#
```

要解决此问题，请将以下标志添加到 cqlsh 连接命令。

```
--ssl
```

您尝试使用 cqlsh 连接 Amazon Keyspaces，但收到 **`SSL transport factory requires a valid certfile to be specified`** 错误。

出现这种情况是因为缺少 SSL/TLS 证书的路径，从而导致以下错误。

```
# 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.
#
```

要解决此问题，请添加证书文件在计算机上的路径。

```
certfile =  path_to_file/sf-class2-root.crt
```

您尝试使用 cqlsh 连接 Amazon Keyspaces，但收到 **`No such file or directory`** 错误。

如果计算机上证书文件的路径错误，可能出现这种情况，从而导致以下错误。

```
# 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')})
#
```

要解决此问题，请验证计算机上的证书文件路径是否正确。

您尝试使用 cqlsh 连接 Amazon Keyspaces，但收到 **[X509] PEM lib** 错误。

如果 SSL/TLS 证书文件 `sf-class2-root.crt` 无效，则可能出现这种情况，从而导致以下错误。

```
# 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)")})
#
```

要解决此问题，请使用以下命令下载 Starfield 数字证书。将 `sf-class2-root.crt` 保存在本地或主目录中。

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

您尝试使用 cqlsh 连接 Amazon Keyspaces，但收到 **unknown** SSL 错误。

如果 SSL/TLS 证书文件 `sf-class2-root.crt` 为空，则可能出现这种情况，从而导致以下错误。

```
# 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)")})
#
```

要解决此问题，请使用以下命令下载 Starfield 数字证书。将 `sf-class2-root.crt` 保存在本地或主目录中。

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

您尝试使用 cqlsh 连接 Amazon Keyspaces，但收到 **`SSL: CERTIFICATE_VERIFY_FAILED`** 错误。

如果无法验证 SSL/TLS 证书文件，则可能出现这种情况，从而导致以下错误。

```
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)")})
```

要解决此问题，请使用以下命令重新下载证书文件。将 `sf-class2-root.crt` 保存在本地或主目录中。

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

您尝试使用 cqlsh 连接到 Amazon Keyspaces，但收到了 **`Last error: timed out`** 错误。

如果您没有在 Amazon EC2 安全组中为 Amazon Keyspaces 配置出站规则，则可能会出现这种情况，从而导致以下错误。

```
# 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")})
#
```

要确认此问题是由于 Amazon EC2 实例的配置造成的`cqlsh`，而不是由 Amazon EC2 实例的配置引起的，您可以尝试使用连接您的密钥空间 AWS CLI，例如使用以下命令。

```
aws keyspaces list-tables --keyspace-name 'my_keyspace'
```

如果此命令也超时，则表示 Amazon EC2 实例的配置不正确。

要确认您是否有足够的权限访问 Amazon Keyspaces，您可以使用 AWS CloudShell 进行连接。`cqlsh`如果该连接已建立，则需要配置 Amazon EC2 实例。

要解决此问题，请确认您的 Amazon EC2 实例具有允许流向 Amazon Keyspaces 的出站规则。如果情况并非如此，则需要为 EC2 实例创建一个新的安全组，并添加一条允许出站流量 Amazon Keyspaces

资源的规则。要更新出站规则以允许流向 Amazon Keyspaces，请从 “类型” 下拉菜单中选择 CQLSH/CAS SANDRA。

使用出站流量规则创建新的安全组后，您需要将其添加到实例中。选择实例，然后依次选择操作、安全性和更改安全组。添加带有出站规则的新安全组，但要确保默认组也保持可用。

有关如何查看和修改 EC2 出站规则的更多信息，请参阅《适用于 Linux 实例的 Amazon EC2 用户指南》中的向安全组添加规则。

您尝试使用 cqlsh 连接 Amazon Keyspaces，但收到 **`Unauthorized`** 错误。

如果您在 IAM 用户策略中缺少 Amazon Keyspaces 权限，则可能会出现这种情况，从而导致以下错误。

```
# 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."',)})
#
```

要解决此问题，请确保 IAM 用户 `testuser-at-12345678910` 有权访问 Amazon Keyspaces。有关授予 Amazon Keyspaces 访问权限的 IAM 策略的示例，请参阅 the section called “基于身份的策略示例”。

有关专门针对 IAM 访问的故障排除指南，请参阅the section called “故障排除”。

您尝试使用 cqlsh 连接 Amazon Keyspaces，但收到 **`Bad credentials`** 错误。

如果用户名或密码错误，则可能出现这种情况，从而导致以下错误。

```
# 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"',)})
#
```

要解决此问题，请验证代码中的 *`USERNAME`* 和 *`PASSWORD`* 与生成服务专用凭证时获得的用户名和密码相匹配。

> **Important**
>
> 如果您在尝试使用 cqlsh 连接时仍然看到错误，请使用 `--debug` 选项重新运行命令，并在联系 AWS Support 时提供详细的输出。

我无法使用 Cassandra 客户端驱动程序连接 Amazon Keyspaces

以下各节显示了使用 Cassandra 客户端驱动程序进行连接时最常见的错误。

您尝试使用驱动程序和 SigV4 插件连接 Amazon Keyspaces，但收到 **`AttributeError`** 错误。

如果证书配置不正确，则会导致以下错误。

```
cassandra.cluster.NoHostAvailable: ('Unable to connect to any servers',
 {'44.234.22.154:9142': AttributeError("'NoneType' object has no attribute
 'access_key'")})
```

要解决此问题，请验证您在使用 Sigv4 插件时传递了与 IAM 用户或角色关联的证书。SigV4 插件需要以下凭证。

- `AWS_ACCESS_KEY_ID`— 指定与 IAM 用户或角色关联的 AWS 访问密钥。
- `AWS_SECRET_ACCESS_KEY` - 指定与访问密钥关联的秘密密钥。这基本上是访问密钥的“密码”。

要了解有关访问密钥和 SigV4 插件的更多信息，请参阅the section called “用于 AWS 身份验证的 IAM 证书”。

您尝试使用驱动程序连接 Amazon Keyspaces 表，但收到 **`PartialCredentialsError`** 错误。

如果缺少 `AWS_SECRET_ACCESS_KEY`，可能会导致以下错误。

```
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')})
```

要解决此问题，请验证您在使用 SigV4 插件时传递了 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY`。要了解有关访问密钥和 SigV4 插件的更多信息，请参阅the section called “用于 AWS 身份验证的 IAM 证书”。

您尝试使用驱动程序连接 Amazon Keyspaces 表，但收到 **`Invalid signature`** 错误。

如果您使用了错误的凭证，则可能会出现这种情况，从而导致以下错误。

```
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"')})
```

要解决此问题，请验证您传递的凭证与您为访问 Amazon Keyspaces 而配置的 IAM 用户或角色相关联。要了解有关访问密钥和 SigV4 插件的更多信息，请参阅the section called “用于 AWS 身份验证的 IAM 证书”。

我的 VPC 端点连接无法正常工作

您尝试使用 VPC 端点连接 Amazon Keyspaces，但收到了令牌映射错误或吞吐量低的问题。

如果 VPC 端点连接配置不正确，则可能出现这种情况。

要解决这些问题，请验证以下配置详细信息。要按照 step-by-step 教程学习如何通过接口 VPC 终端节点为 Amazon Keyspaces 配置连接，请参阅。the section called “连接 VPC 终端节点”

1. 确认用于连接 Amazon Keyspaces 的 IAM 实体具有对用户表的读/写权限以及对系统表的读取权限，如以下示例所示。

```
{
   "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. 确认用于连接 Amazon Keyspaces 的 IAM 实体具有访问 Amazon EC2 实例上 VPC 端点信息所需的读取权限，如以下示例所示。

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

> **Note**
>
> 托管式策略 `AmazonKeyspacesReadOnlyAccess_v2` 和 `AmazonKeyspacesFullAccess` 包含允许 Amazon Keyspaces 访问 Amazon EC2 实例以读取可用接口 VPC 端点信息所需的权限。

有关接口 VPC 端点的更多信息，请参阅the section called “将接口 VPC 端点用于 Amazon Keyspaces”

3. 确认 Java 驱动程序的 SSL 配置将主机名验证设置为 false，如本示例所示。

```
hostname-validation = false
```

有关驱动程序配置的更多信息，请参阅the section called “步骤 2：配置驱动程序”。

4. 要确认 VPC 端点的配置是否正确，可以在 VPC 内运行以下语句。

> **Note**
>
> 您不能使用本地开发人员环境或 Amazon Keyspaces CQL 编辑器来确认此配置，因为它们使用的是公共端点。

```
SELECT peer FROM system.peers;
```

输出应与本示例类似，根据您的 VPC 设置和 AWS 区域，返回 2 到 6 个带有私有 IP 地址的节点。

```
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)
```

我无法使用 **cassandra-stress** 进行连接

您尝试使用 **cassandra-stress** 命令连接 Amazon Keyspaces，但收到 **SSL context** 错误。

如果您尝试连接 Amazon Keyspaces 但未正确设置 trustStore，则会发生这种情况。Amazon Keyspaces 要求使用传输层安全性协议 (TLS) 来帮助保护与客户端的连接。

在这种情况下，您看到以下错误。

```
Error creating the initializing the SSL Context
```

要解决此问题，请按照the section called “开始前的准备工作”主题中所示的说明设置 trustStore。

设置 trustStore 后，您应该能够使用以下命令进行连接。

```
./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"
```

我无法使用 IAM 身份进行连接

您尝试使用 IAM 身份连接 Amazon Keyspaces 表，但收到 **Unauthorized** 错误。

如果您尝试使用 IAM 身份（例如 IAM 用户）连接 Amazon Keyspaces 表，但没有执行策略并先为用户提供所需的权限，就会发生这种情况。

在这种情况下，您看到以下错误。

```
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."',)})
```

要解决此问题，请验证 IAM 用户的权限。要使用标准驱动程序进行连接，用户必须至少拥有对系统表的SELECT 访问权限，因为大多数驱动程序在建立连接时都会读取系统键空间/表。

有关授予 Amazon Keyspaces 系统和用户表访问权限的 IAM 策略示例，请参阅the section called “访问 Amazon Keyspaces 表”。

要查看专门针对 IAM 的故障排除部分，请参阅the section called “故障排除”。

我尝试使用 cqlsh 导入数据，但与 Amazon Keyspaces 表的连接断开

您尝试使用 cqlsh 将数据上传到 Amazon Keyspaces，但收到连接错误。

cqlsh 客户端从服务器连续收到三个任意类型的错误后，与 Amazon Keyspaces 的连接失败。cqlsh 客户端失败时显示以下消息。

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

要解决此错误，您需要确保要导入的数据与 Amazon Keyspaces 中的表模式相匹配。查看导入文件中是否存在解析错误。您可以尝试通过 INSERT 语句来使用单行数据，从而隔离错误。

客户端会自动尝试重新建立连接。

Amazon Keyspaces 中的容量管理故障排除

在使用无服务器容量时遇到问题？ 以下介绍一些常见问题以及如何解决这些问题。

无服务器容量错误

本节简要介绍如何识别与无服务器容量管理相关的错误以及如何解决这些错误。例如，如果您的应用程序超出了预置的吞吐能力，则可能会发现容量不足事件。

由于 Apache Cassandra 是基于集群的软件，专为在节点队列上运行而设计，因此它没有与无服务器功能（例如吞吐能力）相关的异常消息。大多数驱动程序只能理解 Apache Cassandra 中的错误代码，因此 Amazon Keyspaces 使用相同的错误代码集来保持兼容性。

要将 Cassandra 错误映射到底层容量事件，您可以使用亚马逊 CloudWatch 监控相关的 Amazon Keyspaces 指标。导致客户端出错的容量不足事件可根据引起事件的资源分为三类：

- 表 - 如果您为表选择预配置容量模式，并且您的应用程序超出了预配置的吞吐量，则可能会出现容量不足错误。有关更多信息，请参阅 the section called “读/写容量模式”。
- 分区 - 如果给定分区的流量超出 3000 RCU 或 1000 WCU，将出现容量不足事件。我们建议最佳做法是在分区之间均匀分配流量。有关更多信息，请参阅the section called “数据建模”。
- 连接 - 如果超过每个连接每秒最大操作次数的限额，可能会出现吞吐量不足的问题。要提高吞吐量，可以在配置与驱动程序的连接时增加默认连接的数量。有关更多信息，请参阅 the section called “CQL 查询吞吐量调整” 和 the section called “负载均衡”。

要确定是哪个资源导致了返回客户端错误的容量不足事件，可以检查 Amazon Keyspaces 控制台中的控制面板。默认情况下，控制台在该表的 “容量” 选项卡的 “容量和相关 CloudWatch 指标” 部分中提供最常见容量和流量相关指标的汇总视图。

要使用亚马逊创建自己的控制面板 CloudWatch，请查看以下 Amazon Keyspaces 指标。

- `PerConnectionRequestRateExceeded` – 向 Amazon Keyspaces 发出的超出了每个连接请求速率的限额。与 Amazon Keyspaces 的每个客户端连接每秒最多可支持 3000 个 CQL 请求。通过创建多个连接，每秒可以执行超过 3000 个请求。
- `ReadThrottleEvents` – 对 Amazon Keyspaces 的请求超过了表的读取容量。
- `StoragePartitionThroughputCapacityExceeded` — 对 Amazon Keyspaces 存储分区的请求超过了该分区的吞吐能力。Amazon Keyspaces 存储分区每秒最多可支持 1000 个 WCU/WRU，每秒最多可支持 3000 个 RCU/RRU。为减少这些异常情况，我们建议您重新审视数据模型，将读/写流量分配到更多分区。
- `WriteThrottleEvents` – 对 Amazon Keyspaces 的请求超过了表的写入容量。

要了解更多信息 CloudWatch，请参阅the section called “使用监控 CloudWatch”。有关 Amazon Keyspaces 的所有可用 CloudWatch 指标的列表，请参阅。the section called “指标与维度”

> **Note**
>
> 要开始使用显示 Amazon Keyspaces 所有常见指标的自定义控制面板，您可以使用示例存储库 GitHub 中提供的预建 CloudWatch 模板。AWS

主题

- 我从客户端驱动程序接收到 NoHostAvailable 容量不足的错误
- 我在数据导入期间收到写入超时错误
- 我看不到键空间或表的实际存储大小

我从客户端驱动程序接收到 **NoHostAvailable** 容量不足的错误

您看到某个表出现 **Read_Timeout** 或 **Write_Timeout** 异常。

重复尝试向容量不足的 Amazon Keyspaces 表写入或读取数据，可能会导致特定于驱动程序的客户端错误。

CloudWatch 用于监控您的预配置吞吐量和实际吞吐量指标，以及表的容量不足事件。例如，读取请求如果没有足够的吞吐容量，就会出现 `Read_Timeout` 异常，并发布到 `ReadThrottleEvents` 指标。写入请求如果没有足够的吞吐容量，就会出现 `Write_Timeout` 异常，并发布到 `WriteThrottleEvents` 指标。有关这些指标的更多信息，请参阅 the section called “指标与维度”。

请考虑使用以下选项之一来解决这些问题：

- 增加表的预置吞吐量，即应用程序可消耗的最大吞吐能力。有关更多信息，请参阅the section called “读取容量单位和写入容量单位”。
- 让该服务通过自动扩展代表您管理吞吐能力。有关更多信息，请参阅the section called “使用 auto scaling 管理吞吐容量”。
- 为表选择按需容量模式。有关更多信息，请参阅the section called “按需容量模式”。

如果您需要增加账户的默认容量限额，请参阅 限额。

您看到与超出分区容量相关的错误。

当临时超出分区容量时，可能会发生分区节流（可能会由自适应容量或按需容量自动处理）。此错误也可能表明您的数据模型存在问题。Amazon Keyspaces 存储分区每秒最多可支持 1000 个 WCU/WRU，每秒最多可支持 3000 个 RCU/RRU。要了解如何改进数据模型以在更多分区之间分布读写流量，请参阅the section called “数据建模”。

`Write_Timeout` 异常也可能是由于在同一逻辑分区中包含静态和非静态数据的并发写入操作速率过高造成的。如果预计流量会在同一逻辑分区中运行多个包含静态和非静态数据的并发写入操作，我们建议将静态和非静态数据分别写入。分别写入数据还有助于优化吞吐量成本。

您看到与超出连接请求速率相关的错误。

连接节流可能由以下原因造成。

- 您可能没有为每个会话配置足够的连接。
- 由于没有正确配置 VPC 端点权限，获得的连接可能少于可用的对等节点。有关 VPC 端点策略的更多信息，请参阅the section called “将接口 VPC 端点用于 Amazon Keyspaces”。
- 如果使用的是 4.x 驱动程序，请检查是否启用了主机名验证。默认情况下，该驱动程序启用 TLS 主机名验证。此配置会导致 Amazon Keyspaces 在驱动程序中显示为单节点集群。我们建议您关闭主机名验证。

我们建议您遵循以下最佳实践，以确保优化连接和吞吐量：

- 配置 CQL 查询吞吐量调整。

  Amazon Keyspace 支持每个 TCP 连接每秒最多 3,000 次 CQL 查询，但对驱动程序可建立的连接数没有限制。

  大多数开源 Cassandra 驱动程序都会建立一个连接到 Cassandra 的连接池，并在该连接池上对查询进行负载均衡。Amazon Keyspaces 向驱动程序公开 9 个对等 IP 地址。大多数驱动程序的默认行为是为每个对等 IP 地址建立一个连接。因此，使用默认设置的驱动程序的最大 CQL 查询吞吐量将是每秒 27,000 次 CQL 查询。

  要增大此数字，我们建议增加驱动程序在其连接池中维护的每个 IP 地址的连接数。例如，如果将每个 IP 地址的最大连接数设置为 2，则将使驱动程序的最大吞吐量增加一倍，达到每秒 54,000 次 CQL 查询。
- 优化您的单节点连接。

  默认情况下，大多数开源 Cassandra 驱动程序在建立会话时会与 `system.peers` 表中公布的每个 IP 地址建立一个或多个连接。但是，某些配置可能会导致驱动程序连接到单个 Amazon Keyspaces

IP 地址。如果驱动程序尝试对对等节点（例如 DataStax Java 驱动程序）进行 SSL 主机名验证，或者通过 VPC 终端节点进行连接，则可能会发生这种情况。

要获得与连接到多个 IP 地址的驱动程序相同的可用性和性能，建议您执行以下操作：

- 根据所需的客户端吞吐量，将每个 IP 的连接数增加到 9 或更高。
- 创建自定义重试策略，确保针对同一个节点运行重试。
- 如果使用 VPC 端点，请向用于连接 Amazon Keyspaces 的 IAM 实体授予访问权限，以查询您的 VPC 的端点和网络接口信息。这样可以改善负载均衡并提高读/写吞吐量。有关更多信息，请参阅???。

我在数据导入期间收到写入超时错误

使用 **`cqlsh COPY`** 命令上传数据时，您收到超时错误。

```
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 使用 `ReadTimeout` 和 `WriteTimeout` 异常指示写入请求何时因吞吐容量不足而失败。为了帮助诊断容量不足异常，Amazon Keyspaces 在亚马逊上发布了以下指标。 CloudWatch

- `WriteThrottleEvents`
- `ReadThrottledEvents`
- `StoragePartitionThroughputCapacityExceeded`

要解决数据加载期间容量不足的错误，请降低每个工作线程的写入速率或总摄取速率，然后重试上传各行。有关更多信息，请参阅the section called “步骤 4：配置 `cqlsh COPY FROM` 设置”。要获得更强大的数据上传选项，可以考虑使用GitHub 存储库中提供的 DSBulk。有关 step-by-step说明，请参阅the section called “使用 DSBulk 加载数据”。

我看不到键空间或表的实际存储大小

您看不到键空间或表的实际存储大小。

您可以通过计算表中的行大小来估算存储大小。有关计算行大小的详细说明，请参阅the section called “计算行大小”。

Amazon Keyspaces 中的数据定义语言故障排除

创建资源时遇到问题？ 以下介绍一些常见问题以及如何解决这些问题。

数据定义语言错误

Amazon Keyspaces 异步执行数据定义语言 (DDL) 操作，例如创建和删除键空间和表。如果应用程序在资源准备就绪之前尝试使用该资源，则操作将失败。

您可以在中监控新密钥空间和表的创建状态 AWS Management Console，这会指示密钥空间或表何时处于待处理状态或处于活动状态。您还可以通过查询系统架构表，以编程方式监控新的键空间或表的创建状态。当键空间或表准备就绪可供使用时，就会在系统架构中变为可见。

> **Note**
>
> 要使用优化密钥空间的创建 AWS CloudFormation，您可以使用此实用程序将 CQL 脚本转换为 CloudFormation 模板。该工具可从GitHub 存储库中获得。

主题

我创建了一个新的键空间，但无法查看或访问它

您在应用程序尝试访问新的键空间时收到错误。

如果您尝试访问新创建的 Amazon Keyspaces 键空间，但该键空间仍在异步创建中，则会出现错误。下面是一个错误示例。

```
InvalidRequest: Error from server: code=2200 [Invalid query] message="unconfigured
 keyspace mykeyspace"
```

要检查新的键空间何时可供使用，推荐的设计模式是轮询 Amazon Keyspaces 系统架构表 (system_schema_mcs.*)。

有关更多信息，请参阅the section called “创建键空间”。

我创建了一个新表，但无法查看或访问它

您在应用程序在尝试访问新表时收到错误。

如果您尝试访问新创建的 Amazon Keyspaces 表，但该表仍在异步创建中，则会出现错误。例如，尝试查询尚不可用的表会失败并看到 `unconfigured table` 错误。

```
InvalidRequest: Error from server: code=2200 [Invalid query] message="unconfigured
 table mykeyspace.mytable"
```

尝试使用 `sync_table()` 查看表时出现 `KeyError`。

```
KeyError: 'mytable'
```

要检查新的表何时可供使用，推荐的设计模式是轮询 Amazon Keyspaces 系统架构表 (system_schema_mcs.*)。

这是正在创建的表的输出示例。

```
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)
```

这是处于活动状态的表的输出示例。

```
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)
```

有关更多信息，请参阅the section called “创建表”。

我正在尝试使用 Amazon Keyspaces point-in-time 恢复 (PITR) 恢复表，但恢复失败了

如果您正在尝试使用恢复 (PITR) point-in-time 恢复 Amazon Keyspaces 表，但看到恢复过程开始但未成功完成，则可能没有为该特定表配置恢复过程所需的所有权限。

除了用户权限外，Amazon Keyspaces 可能还需要权限才能在恢复过程中代表您的主题执行操作。如果表使用客户托管式密钥加密，或者您使用限制传入流量的 IAM 策略，则会出现这种情况。

例如，如果您在 IAM 策略中使用条件密钥将源流量限制为特定端点或 IP 范围，则恢复操作会失败。要允许 Amazon Keyspaces 代表您的主体执行表恢复操作，必须在 IAM 策略中添加 `aws:ViaAWSService` 全局条件键。

有关恢复表所需的权限的更多信息，请参阅the section called “还原权限。”。

我尝试使用 INSERT/UPDATE 来编辑自定义的生存时间 (TTL) 设置，但是操作失败了

如果您试图插入或更新自定义 TTL 值，操作可能会失败，并出现以下错误。

```
TTL is not yet supported.
```

要使用 `INSERT` 或 `UPDATE` 操作为行或列指定自定义 TTL 值，必须先为表启用 TTL。您可以使用 `ttl` 自定义属性为表启用 TTL。

有关为表启用自定义 TTL 设置的更多信息，请参阅the section called “如何使用自定义属性对现有表启用生存时间 (TTL)”。

我尝试将数据上传到 Amazon Keyspaces 表中，但收到超出列数的错误

您正在上传数据，但已超过可以同时更新的列数。

当您的表架构超过 350 KB 的大小上限时，就会发生此错误。有关更多信息，请参阅限额。

我尝试删除 Amazon Keyspaces 表中的数据，但删除范围失败了

您尝试通过分区键删除数据，但收到范围删除错误。

当您试图在一次删除操作中删除 1,000 多行时，就会发生此错误。

```
Range delete requests are limited by the amount of items that can be deleted in a
 single range.
```

有关更多信息，请参阅the section called “范围删除”。

要删除单个分区中的 1,000 多行，请考虑以下选项。

- 按分区删除 - 如果大部分分区的行数低于 1,000 行，则可以尝试按分区删除数据。如果分区包含的行数超过 1,000，则改为尝试按聚类列删除。
- 按聚类列删除 - 如果模型包含多个聚类列，可以使用列层次结构来删除多行。聚类列是一种嵌套结构，通过对顶级列进行操作，可以删除许多行。
- 按单行删除 - 您可以遍历行，并按其完整主键（分区列和聚类列）删除每行。
- 最佳做法是考虑在分区之间拆分行 — 在 Amazon Keyspaces 中，我们建议您在表分区之间分配吞吐量。这样可以将数据和访问平均分配给物理资源，从而提供最佳吞吐量。有关更多信息，请参阅the section called “数据建模”。

在为繁重的工作负载计划删除操作时，请考虑以下建议。

- 使用 Amazon Keyspaces，分区可以包含几乎无限数量的行。这样，您就可以将分区扩展为比传统 Cassandra 指导值 100 MB 更“宽”的范围。随着时间的推移，时间序列或分类账的数据增长超过千兆字节的情况并不少见。
- 有了 Amazon Keyspaces，当您需要对繁重的工作负载执行删除操作时，就无需考虑压缩策略或墓碑。您可以删除任意数量的数据而不会影响读取性能。

Amazon Keyspaces（Apache Cassandra 兼容）中的无服务器资源管理

Amazon Keyspaces（Apache Cassandra 兼容）是一项无服务器服务。Amazon Keyspaces 不是通过集群中的节点为您的工作负载部署、管理和维护存储及计算资源，而是直接向表分配存储和读/写吞吐量资源。

本章详细介绍了 Amazon Keyspaces 中的无服务器资源管理。要了解如何使用 Amazon 监控无服务器资源 CloudWatch，请参阅the section called “使用监控 CloudWatch”。

主题

Amazon Keyspaces 中的存储

Amazon Keyspaces（Apache Cassandra 兼容）会根据表中存储的实际数据自动为表预置存储。您不需要预先为表预置存储。当您的应用程序写入、更新和删除数据时，Amazon Keyspaces 会自动扩展和缩减表的存储。与传统的 Apache Cassandra 集群不同，Amazon Keyspaces 不需要额外的存储来支持压缩等低级系统操作。您仅需为实际使用的存储付费。

默认情况下，Amazon Keyspaces 会将键空间的复制因子设置为 3。您无法修改复制因子。Amazon Keyspaces 在多个 AWS 可用区中自动复制表数据三次，以实现高可用性。Amazon Keyspaces 存储的每 GB 价格已包含复制费用。有关更多信息，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定价。

Amazon Keyspaces 会持续监控表的大小，以确定存储费用。有关 Amazon Keyspaces 如何计算可计费数据大小的更多信息，请参阅 the section called “计算行大小”。

Amazon Keyspaces 中的读/写容量模式

Amazon Keyspaces 具有两个读/写容量模式来处理表的读写：

- 按需（默认）
- 已预置

您选择的读/写容量模式控制对读写吞吐量收费的方式以及管理表吞吐容量的方式。

主题

- 按需容量模式
- 预置的吞吐容量模式
- 管理和查看容量模式
- 更改容量模式时的注意事项

按需容量模式

Amazon Keyspaces（Apache Cassandra 兼容）按需容量模式是一个灵活的计费选项，可以每秒处理数千个请求而不需要进行容量规划。此选项为读取和写入请求提供 pay-per-request 定价，因此您只需为实际用量付费。

选择按需模式时，Amazon Keyspaces 可以将表的吞吐容量立即扩展到以前达到的任何流量级别，然后在应用程序流量减少时调整回来。如果工作负载的流量级别达到新的峰值，则该服务会迅速调整以便为您的表增加吞吐容量。您可以为新表和现有表启用按需容量模式。

如果满足以下任意条件，则按需模式是很好的选项：

- 您创建工作负载未知的新表。
- 您具有不可预测的应用程序流量。
- 您更喜欢只为您使用的容量付费。

要开始使用按需模式，您可以使用控制台或使用几行 Cassandra 查询语言 (CQL) 代码创建新表或更新现有表以使用按需容量模式。有关更多信息，请参阅 the section called “表”。

主题

- 读取请求单位和写入请求单位
- 峰值流量和扩缩属性
- 按需容量模式的最初吞吐量

读取请求单位和写入请求单位

使用按需容量模式表，您无需预先指定您希望应用程序使用多少读取和写入吞吐量。Amazon Keyspaces 会根据读取请求单位 (RRU) 和写入请求单位 (WRU) 向您对表执行的读取和写入操作收费。

- 对于大小不超过 4 KB 的行，一个 RRU 代表一个 LOCAL_QUORUM 读取请求，或两个 LOCAL_ONE 读取请求。如果您需要读取大于 4 KB 的行，则读取操作将使用额外的 RRU。所需的 RRU 总数取决于行大小，以及要使用 LOCAL_QUORUM 还是 LOCAL_ONE 读取一致性。例如，使用 LOCAL_QUORUM 读取一致性读取一个 8 KB 行需要 2 个 RRU，如果选择 LOCAL_ONE 读取一致性，则需要 1 RRU。
- 一个 WRU 代表对大小不超过 1 KB 的行的一次写入。所有写入操作都使用 LOCAL_QUORUM 一致性，使用轻量级事务 (LWT) 不收取额外费用。如果您需要写入大于 1 KB 的行，则写入操作将使用额外的 WRU。所需 WRU 的总数取决于行大小。例如，如果行大小为 2 KB，则需要 2 个 WRU 才能执行一个写入请求。

有关支持的一致性级别的信息，请参阅the section called “支持的 Cassandra 一致性级别”。

峰值流量和扩缩属性

使用按需容量模式的 Amazon Keyspaces 表会自动适应应用程序的流量。按需容量模式会即时在表中承受之前双倍的峰值流量。例如，您的应用程序流量模式可能在每秒 5,000 到 10,000 次 LOCAL_QUORUM 读取之间变化，其中每秒 10,000 次读取是以前的流量峰值。

使用这种模式，按需容量模式可即时容纳最高每秒 20,000 次读取的持续流量。如果应用程序承受每秒 20,000 次读取的流量，则该峰值将成为新的之前峰值，从而使后续流量高达每秒 40,000 次读取。

如果您在一个表上需要的流量是前一个峰值的两倍以上，则随着流量的增加，Amazon Keyspaces 会自动分配更多容量。这有助于确保您的表具有足够的吞吐容量来处理额外的请求。但是，如果您在 30 分钟内超过前一个峰值的两倍，则可能会出现吞吐容量不足错误。

例如，假设您的应用程序流量模式在每秒 5,000 到 10,000 个强一致性读取之间变化，而上一次达到的流量峰值为每秒 20,000 次读取。在这种情况下，服务建议您在将流量推动到每秒 40,000 次读取之前，至少将流量增长的时间间隔 30 分钟。

要了解如何估算表的读取和写入容量消耗，请参阅the section called “估算容量消耗”。

要了解有关账户的默认配额以及如何增加此配额的更多信息，请参阅 限额。

按需容量模式的最初吞吐量

如果您在启用按需容量模式的情况下创建了新表，或者首次将现有表切换为按需容量模式，则该表将具有以下之前峰值设置，即使该表之前尚未使用按需容量模式提供流量也是如此。

- 在按需容量模式下新创建的表：先前峰值为 2000 个 WRU 和 6000 个 RRU。您可以立即将以前的峰值翻倍。这样，新创建的按需表可以提供最高 4000 个 WRU 和 12000 个RRU。
- 现有表切换为按需容量模式：先前峰值是为该表先前预置的 WCU 和 RCU 的一半，或者是按需容量模式的新建表的设置，以较高者为准。

预置的吞吐容量模式

如果您选择预置的吞吐容量 模式，则指定您的应用程序需要的每秒读取和写入次数。这有助于您管理 Amazon Keyspaces 使用量，使其保持在或低于定义的请求速率以优化价格并保持可预测性。要详细了解预置吞吐量的自动扩展，请参阅the section called “使用 auto scaling 管理吞吐容量”。

如果满足以下任意条件，则预置的吞吐容量模式是很好的选项：

- 您具有可预测的应用程序流量。
- 您运行流量比较稳定或逐渐增加的应用程序。
- 您可以预测容量要求以优化价格。

读取容量单位和写入容量单位

对于预置的吞吐容量模式表，您可以按读取容量单位 (RCU) 和写入容量单位 (WCU) 指定吞吐容量：

- 对于大小不超过 4 KB 的行，一个 RCU 表示每秒一个 LOCAL_QUORUM 读取操作或每秒两个 LOCAL_ONE 读取操作。如果您需要读取大于 4 KB 的行，则读取操作将使用额外的 RCU。

  所需的 RCU 总数取决于行大小，以及要使用 LOCAL_QUORUM 还是 LOCAL_ONE 读取。例如，如果您的行大小为 8 KB，则需要 2 个 RCU 来维持每秒一个 LOCAL_QUORUM 读取；如果选择 LOCAL_ONE 读取，则需要 1 个 RCU。
- 一个 WCU 表示每秒对大小不超过 1 KB 的行执行一次写入操作。所有写入操作都使用 LOCAL_QUORUM 一致性，使用轻量级事务 (LWT) 不收取额外费用。如果您需要写入大于 1 KB 的行，则写入操作将使用额外的 WCU。

所需 WCU 的总数取决于行大小。例如，如果您的行大小为 2 KB，则您需要 2 个 WCU 来维持每秒一个写入请求。有关如何估算表的读取和写入容量消耗的更多信息，请参阅the section called “估算容量消耗”。

如果您的应用程序读取或写入较大的行（最大为 1 MB 的 Amazon Keyspaces 行大小上限），它将消耗更多的容量单位。要了解有关如何估算行大小的更多信息，请参阅 the section called “计算行大小”。例如，假设您创建了具有 6 个 RCU 和 6 个 WCU 的预置表。使用这些设置，您的应用程序可以执行以下操作：

- 每秒执行高达 24 KB 的 LOCAL_QUORUM 读取（4 KB × 6 个 RCU）。
- 执行最高每秒 48 KB 的 LOCAL_ONE 读取（读取吞吐量的两倍）。
- 每秒写入高达 6 KB（1 KB × 6 个 WCU）。

预置的吞吐量 是应用程序可以从表消耗的最大吞吐容量。如果您的应用程序超出了预置的吞吐容量，则可能会发现容量不足错误。

例如，读取请求如果没有足够的吞吐容量，就会出现 Read_Timeout 异常，并发布到 ReadThrottleEvents 指标。写入请求如果没有足够的吞吐容量，就会出现 Write_Timeout 异常，并发布到 WriteThrottleEvents 指标。

您可以使用 Amazon CloudWatch 监控您的预配置吞吐量和实际吞吐量指标以及容量不足事件。有关这些指标的更多信息，请参阅 the section called “指标与维度”。

> **Note**
>
> 由于容量不足而反复出现错误可能会导致客户端驱动程序特定的异常，例如， DataStax Java 驱动程序因出现故障。NoHostAvailableException

要更改表的吞吐容量设置，您可以使用 AWS Management Console 或使用 CQL 的 ALTER TABLE 语句，有关更多信息，请参阅 the section called “ALTER TABLE”。

要了解有关账户的默认配额以及如何增加此配额的更多信息，请参阅 限额。

管理和查看容量模式

您可以在 Amazon Keyspaces 系统键空间中查询系统表，以查看有关表的容量模式信息。您还可以查看表使用的是按需吞吐容量模式还是预置吞吐容量模式。如果表配置为预置吞吐量模式，您会看到为表预置的吞吐容量。

示例

```
SELECT * from system_schema_mcs.tables where keyspace_name = 'mykeyspace' and
 table_name = 'mytable';
```

使用按需容量模式配置的表返回以下内容。

```
                    {
                        'capacity_mode': {
                            'last_update_to_pay_per_request_timestamp':
'1579551547603',
                            'throughput_mode': 'PAY_PER_REQUEST'
                         }
                    }
```

使用预置吞吐容量模式配置的表将返回以下内容。

```
                    {
                        '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` 值以毫秒为单位。

要更改表的预置吞吐容量，请使用 the section called “ALTER TABLE”。

更改容量模式时的注意事项

当您将表从预置的容量模式切换到按需容量模式时，Amazon Keyspaces 会对表和分区的结构进行若干更改。此过程可能耗时数分钟。在切换期间，您的表将提供与先前预置的 WCU 和 RCU 数量相一致的吞吐量。

当您从按需容量模式切换回预置的容量模式时，表将提供与表设置为按需容量模式时达到的先前峰值一致的吞吐量。

> **Note**
>
> 每 24 个小时的周期内，您可以在读/写容量模式之间切换两次。

使用 Amazon Keyspaces 自动扩展功能自动管理吞吐容量

许多数据库工作负载本质上是周期性的，或者难以提前进行预测。例如，考虑一个大多数用户在白天处于活跃状态的社交网络应用程序。数据库必须能够处理白天活动，但夜间不需要相同级别的吞吐量。

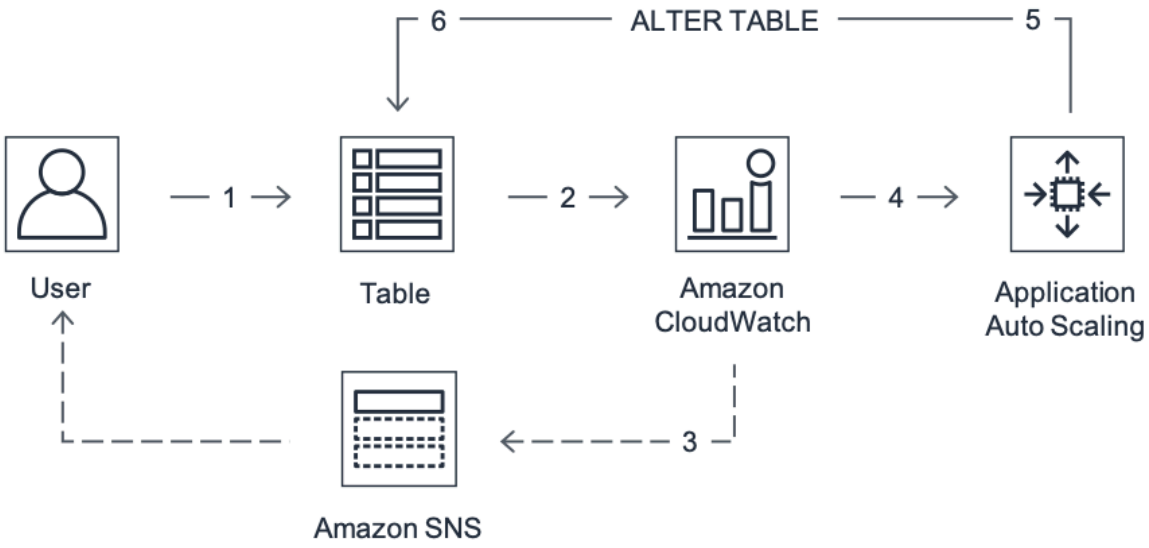
另一个示例是面临快速采用的新移动游戏应用程序。如果此游戏变得极受欢迎，它可能会超出可用的数据库资源，从而导致性能降低并使客户感到不满。这些类型的工作负载通常需要手动干预来扩展或缩减数据库资源，以便响应不断变化的使用量级别。

Amazon Keyspaces（Apache Cassandra 兼容）可以根据实际应用程序流量自动调整吞吐容量，从而帮助您有效地为可变工作负载预置吞吐容量。Amazon Keyspaces 使用 Application Auto Scaling 服务来代表您增加或减少表的读写容量。有关 Application Auto Scaling 的更多信息，请参阅 Application Auto Scaling 用户指南。

> **Note**
>
> 要快速开始使用 Amazon Keyspaces 自动扩缩，请参阅the section called “使用 控制台”。要使用 Cassandra 查询语言 (CQL) 管理 Amazon Keyspaces 扩展策略，请参阅。the section called “使用 CQL”要了解如何使用 CLI 管理 Amazon Keyspaces 扩展策略，请参阅。the section called “使用 CLI”

Amazon Keyspaces 自动扩缩的工作原理

下图简要概述了 Amazon Keyspaces 自动扩缩如何管理表的吞吐容量。



要为表启用自动扩展，请创建扩展策略。扩展策略指定是要扩展读取容量还是写入容量（或二者），并为表指定最小的和最大的预置容量单位设置。

扩展策略还定义了 目标利用率。目标利用率是在某个时间点上使用的容量单位与预置容量单位的比率（以百分比表示）。自动扩展使用目标跟踪 算法向上或向下调整表的预置吞吐量以响应实际工作负载。这样做的目的是使实际容量利用率保持在目标利用率或接近目标利用率。

您可以为读取和写入容量设置介于 20% 和 90% 之间的自动扩展目标利用率值。默认的目标利用率为 70%。如果您的流量快速变化，并且您希望容量能够尽快开始扩展，则可以将目标利用率设置为较低的百分比。如果您的应用程序流量变化较慢，并且您希望降低吞吐量成本，则也可以将目标利用率设置为较高的百分比。

有关扩展策略的更多信息，请参阅《Application Auto Scaling 用户指南》中的 Application Auto Scaling 的目标跟踪扩展策略。

当您创建扩展策略时，Amazon Keyspaces 会代表您创建两对亚马逊 CloudWatch 警报。每对警报均表示预置和使用的吞吐量设置的上限和下限。当表的实际利用率在持续一段时间内偏离目标利用率时，就会触发这些 CloudWatch 警报。要了解有关亚马逊的更多信息 CloudWatch，请参阅亚马逊 CloudWatch 用户指南。

当其中一个 CloudWatch 警报被触发时，亚马逊简单通知服务 (Amazon SNS) Service 会向您发送通知（如果您已启用）。然后， CloudWatch 警报会调用 Application Auto Scaling 来评估您的扩展策略。这进而会向 Amazon Keyspaces 发出 Alter Table 请求，以便根据情况增加或减少表的预置容量。要了解有关 Amazon SNS 通知的更多信息，请参阅设置 Amazon SNS 通知。

Amazon Keyspaces 会处理 Alter Table 请求，方式是动态增加（或减少）表的预置吞吐容量，使它接近目标利用率。

> **Note**
>
> Amazon Keyspaces auto scaling 仅在实际工作负载持续升高（或降低）几分钟时才会修改预配置的吞吐量设置。 目标跟踪算法寻求使目标使用率长期达到或接近选定值。表的内置容量暴增将容纳活动的短时间突增峰值。

auto 缩放对多区域表的工作原理

为确保在预置容量模式下所有多区域表中的所有 AWS 区域 表副本始终有足够的读取和写入容量，我们建议您配置 Amazon Keyspaces 自动扩展。

当您在预配置模式下使用带有 auto Scaling 的多区域表时，您无法为单个表副本禁用自动缩放。但是您可以针对不同区域调整表的读取 auto 缩放设置。例如，您可以为复制表的每个区域指定不同的读取容量和读取 auto scaling 设置。

您在指定区域中为表副本配置的读取自动缩放设置会覆盖表的常规自动缩放设置。但是，写入容量必须在所有表副本之间保持同步，以确保有足够的容量在所有区域中复制写入。

Amazon Keyspaces 自动扩展会 AWS 区域 根据该区域的使用情况独立更新每个表的预配置容量。因此，当 auto scaling 处于活动状态时，每个区域中多区域表的预配置容量可能会有所不同。

您可以使用 Amazon Keyspaces 控制台、API 或 CQL 配置多区域表及其副本的自动扩展设置。 AWS CLI有关如何创建和更新多区域表的 auto Scaling 设置的更多信息，请参阅the section called “如何使用多区域复制”。

> **Note**
>
> 如果您对多区域表使用自动缩放，则必须始终使用 Amazon Keyspaces API 操作来配置自动扩展设置。如果您直接使用 Application Auto Scaling API 操作来配置自动缩放设置，则无法指定多区域表的。 AWS 区域 这可能会导致配置不受支持。

使用说明

在开始使用 Amazon Keyspaces 自动扩缩之前，您应了解以下内容：

- Amazon Keyspaces 自动扩缩会根据您的扩缩策略在必要时增加读取容量或写入容量。所有 Amazon Keyspaces 配额仍将有效，如 限额中所述。
- Amazon Keyspaces 自动扩缩不会阻止您手动修改预置的吞吐量设置。这些手动调整不会影响附加到扩展策略的任何现有 CloudWatch 警报。
- 如果您使用控制台创建预置了吞吐容量的表，则默认情况下将启用 Amazon Keyspaces 自动扩缩。您可以随时修改自动扩展设置。有关更多信息，请参阅 the section called “使用 控制台”。
- 如果您使用 AWS CloudFormation 创建扩展策略，则应从中管理扩展策略， AWS CloudFormation 以便堆栈与堆栈模板同步。如果您更改了 Amazon Keyspaces 的扩展策略，则在重置堆栈时，这些策略将被 AWS CloudFormation 堆栈模板中的原始值覆盖。
- 如果您使用 CloudTrail 监控Amazon Keyspaces的自动扩展，则可能会看到Application Auto Scaling在配置验证过程中拨打的呼叫提醒。您可以使用 `invokedBy` 字段筛选掉这些警报，其中包含用于这些验证检查的 `application-autoscaling.amazonaws.com`。

使用控制台管理 Amazon Keyspaces 自动扩缩策略

您可以使用控制台为新表和现有表启用 Amazon Keyspaces 自动扩缩。您还可以使用该控制台修改自动扩展设置或禁用自动扩展。

> **Note**
>
> 要获得更高级的功能，例如设置缩减和横向扩展冷却时间，请使用 CQL 或 () AWS Command Line Interface 以编程方式AWS CLI管理 Amazon Keyspaces 扩展策略。有关更多信息，请参阅使用 Cassandra 查询语言 (CQL) 管理 Amazon Keyspaces 自动扩展或使用 CLI 管理 Amazon Keyspaces 扩展策略。

主题

- 开始之前：向用户授予 Amazon Keyspaces 自动缩放的权限
- 创建启用了 Amazon Keyspaces 自动扩缩的新表
- 对现有表启用 Amazon Keyspaces 自动扩缩
- 修改或禁用 Amazon Keyspaces 自动扩缩设置
- 在控制台上查看 Amazon Keyspaces 自动扩缩活动

开始之前：向用户授予 Amazon Keyspaces 自动缩放的权限

要开始使用，请确认用户具有创建和管理自动扩展设置所需的适当权限。在 AWS Identity and Access Management (IAM) 中，需要使用 AWS 托管策略来管理 Amazon Keys `AmazonKeyspacesFullAccess` paces 扩展策略。

> **Important**
>
> 需要 `application-autoscaling:*` 权限才能对表禁用自动扩展。必须先关闭表的 auto 缩放功能，然后才能将其删除。

要针对 Amazon Keyspaces 控制台访问和 Amazon Keyspaces 自动扩缩设置 IAM 用户，请添加以下策略。

附加 **`AmazonKeyspacesFullAccess`** 策略

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在 IAM 控制台控制面板上，选择 Users (用户)，然后从列表中选择您的 IAM 用户。
3. 在 Summary (摘要) 页上，选择 Add permissions (添加权限)。
4. 选择直接附加现有策略。
5. 从策略列表中选择 AmazonKeyspacesFullAccess，然后选择下一步：查看。
6. 选择添加权限。

创建启用了 Amazon Keyspaces 自动扩缩的新表

> **Note**
>
> Amazon Keyspaces 自动扩缩需要存在一个代表您执行自动扩缩操作的服务相关角色 (`AWSServiceRoleForApplicationAutoScaling_CassandraTable`)。将自动为您创建此角色。有关更多信息，请参阅the section called “使用服务相关角色”。

创建启用了自动扩展的新表

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 在导航窗格中，选择表，然后选择创建表。
3. 在创建表页面的表详细信息部分中，选择一个键空间并为新表提供一个名称。
4. 在 “列” 部分中，为您的表创建架构。
5. 在主键部分中，定义表的主键并选择可选的聚类列。
6. 在表设置部分，选择自定义设置。
7. 继续读取/写入容量设置。
8. 对于 Capacity mode (容量模式)，选择 Provisioned (预置)。
9. 在 Read capacity (读取容量) 部分中，确认已选择 Scale automatically(自动扩展)。

   在此步骤中，您将选择表的最小和最大读取容量单位以及目标利用率。

   - 最小容量单位：输入表应始终支持的最小吞吐量级别的值。该值必须介于 1 和账户的每秒最大吞吐量配额（默认为 40000）之间。
   - 最大容量单位：输入要为表预置的最大吞吐量。该值必须介于 1 和账户的每秒最大吞吐量配额（默认为 40000）之间。
   - 目标利用率：输入介于 20% 和 90% 之间的目标利用率。当流量超过定义的目标利用率时，容量将自动扩展。当流量低于定义的目标时，容量将自动重新缩减。

   > **Note**
   >
   > 要了解有关账户的默认配额以及如何增加此配额的更多信息，请参阅 限额。

10. 在写入容量部分，选择与上一步中定义的读取容量相同的设置，或者手动配置容量值。
11. 选择创建表。使用指定的自动扩展参数创建表。

对现有表启用 Amazon Keyspaces 自动扩缩

> **Note**
>
> Amazon Keyspaces 自动扩缩需要存在一个代表您执行自动扩缩操作的服务相关角色 (AWSServiceRoleForApplicationAutoScaling_CassandraTable)。将自动为您创建此角色。有关更多信息，请参阅the section called “使用服务相关角色”。

为现有表启用 Amazon Keyspaces 自动扩缩

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 选择要使用的表，然后转到 “容量” 选项卡。
3. 在容量设置部分，选择编辑。
4. 在容量模式下，确保表使用预置容量模式。
5. 选择自动扩展，然后按照创建启用了 Amazon Keyspaces 自动扩缩的新表中的步骤 6 操作来编辑读取和写入容量。
6. 在定义自动扩展设置时，选择 Save (保存)。

修改或禁用 Amazon Keyspaces 自动扩缩设置

您可以使用 AWS Management Console 来修改您的 Amazon Keyspaces 自动缩放设置。为此，请选择要编辑的表，然后转到 “容量” 选项卡。在容量设置部分，选择编辑。现在，您可以修改读取容量或写入容量部分的设置。有关这些设置的更多信息，请参阅 创建启用了 Amazon Keyspaces 自动扩缩的新表。

要禁用 Amazon Keyspaces 的自动缩放，请清除 “自动缩放” 复选框。禁用自动缩放功能会在 Application Auto Scaling 中取消表作为可扩展目标的注册。要删除 Application Auto Scaling 用于访问 Amazon Keyspaces 表的服务相关角色，请按照the section called “删除适用于 Amazon Keyspaces 的服务相关角色”中的步骤操作。

> **Note**
>
> 要删除 Application Auto Scaling 使用的服务相关角色，必须禁用账户中所有表的自动缩放功能。 AWS 区域

在控制台上查看 Amazon Keyspaces 自动扩缩活动

您可以使用亚马逊来监控 Amazon Keyspaces 自动扩展如何使用资源 CloudWatch，亚马逊会生成有关您的使用情况和性能的指标。按照《Application Auto Scaling 用户指南》中的步骤创建 CloudWatch 仪表板。

使用 Cassandra 查询语言 (CQL) 管理 Amazon Keyspaces 自动扩展

要使用 Cassandra 查询语言 (CQL) 为 Amazon Keyspaces 表创建和管理 Amazon Keyspaces 表的自动缩放设置，您可以使用。`cqlsh`本主题概述了您可以使用 CQL 以编程方式管理的 auto Scaling 任务。

有关本主题中描述的 CQL 语句的更多信息，请参阅the section called “DDL 语句”。

主题

- 开始前的准备工作
- 使用 CQL 创建具有自动缩放功能的新表
- 使用 CQL 在现有表上启用自动缩放
- 使用 CQL 查看表的 Amazon Keyspaces 自动扩展配置
- 使用 CQL 关闭表的 Amazon Keyspaces 自动缩放功能

开始前的准备工作

在开始之前，您需要完成以下任务。

配置 权限

如果未完成这些任务，您必须为用户配置相应的权限，以创建和管理自动扩展设置。在 AWS Identity and Access Management (IAM) 中，需要使用 AWS 托管策略来管理 Amazon Keys `AmazonKeyspacesFullAccess` paces 扩展策略。有关详细步骤，请参阅the section called “开始之前：向用户授予 Amazon Keyspaces 自动缩放的权限”。

配置 `cqlsh`

如果您尚未这样做，则必须进行安装和配置`cqlsh`。为此，请按照中的说明进行操作the section called “使用`cqlsh-expansion`”。然后，您可以使用 AWS CloudShell 来运行以下各节中的命令。

使用 CQL 创建具有自动缩放功能的新表

创建新的 Amazon Keyspaces 表时，可以在语句中自动为表的写入或读取容量启用自动缩放。CREATE TABLE这允许 Amazon Keyspaces 代表您联系 Application Auto Scaling，将该表注册为可扩展目标并调整预配置的写入或读取容量。

有关如何创建多区域表以及如何为表副本配置不同的 auto Scaling 设置的更多信息，请参阅。the section called “使用默认设置创建多区域表 (CQL)”

> **Note**
>
> Amazon Keyspaces 自动扩展需要服务相关角色 (AWSServiceRoleForApplicationAutoScaling_CassandraTable) 的存在才能代表您执行自动扩展操作。将自动为您创建此角色。有关更多信息，请参阅the section called “使用服务相关角色”。

要以编程方式为表配置自动缩放设置，您可以使用包含 Amazon Keyspaces 自动缩放参数的AUTOSCALING_SETTINGS语句。这些参数定义了指示 Amazon Keyspaces 调整表的预配置吞吐量的条件，以及要采取哪些其他可选操作。在此示例中，您定义了 mytable 的自动缩放设置。

该策略包含以下元素：

- AUTOSCALING_SETTINGS— 指定是否允许 Amazon Keyspaces 代表您调整吞吐容量。以下值是必需的：
  - provisioned_write_capacity_autoscaling_update:
    - minimum_units
    - maximum_units
  - provisioned_read_capacity_autoscaling_update:
    - minimum_units
    - maximum_units
  - scaling_policy— Amazon Keyspaces 支持目标跟踪政策。要定义目标跟踪策略，请配置以下参数。
    - target_value— Amazon Keyspaces 自动扩展可确保已用容量与预配置容量的比率保持在该值或接近该值。您将 target_value 定义为百分比。

- disableScaleIn:（可选）Aboolean，它指定该表scale-in是禁用还是启用。默认情况下，此参数处于禁用状态。要开启scale-in，请将该boolean值设置为FALSE。这意味着代表您自动缩小餐桌的容量。
- scale_out_cooldown – 横向扩展活动会增加表的预置吞吐量。要为横向扩展活动增加冷却时间，请为 scale_out_cooldown 指定一个值（以秒为单位）。如果未指定值，则默认值为 0。有关目标跟踪和冷却时间的更多信息，请参阅 App lication Auto Scaling 用户指南中的目标跟踪扩展策略。
- scale_in_cooldown – 横向缩减活动会减小表的预置吞吐量。要为缩减活动增加冷却时间，请为 scale_in_cooldown 指定一个值（以秒为单位）。如果未指定值，则默认值为 0。有关目标跟踪和冷却时间的更多信息，请参阅 App lication Auto Scaling 用户指南中的目标跟踪扩展策略。

> **Note**
>
> 为了进一步了解 target_value 的工作原理，假设您的表的预配置吞吐量设置为 200 个写入容量单位。您决定为此表创建扩展策略，并使用 target_value 的 70%。
> 现在假设您开始将写入流量驱动到表，以便实际写入吞吐量为 150 个容量单位。现在的 consumed-to-provisioned 比率是（150/200），或75%。此比率超过了您的目标，因此 auto scaling 会将预配置的写入容量增加到 215，因此该比率为 (150/215)，即 69.77%，target_value尽可能接近您的目标，但不超过该比率。

对于 mytable，您可以将读取和写入容量都设置为 TargetValue 50%。Amazon Keyspaces 自动扩展可在 5-10 个容量单位范围内调整表的预配置吞吐量，使该 consumed-to-provisioned 比率保持在或接近 50%。对于读取容量，您可以将ScaleOutCooldown和的值设置ScaleInCooldown为 60 秒。

您可以使用以下语句创建启用自动扩展功能的新 Amazon Keyspaces 表。

```
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
            }
        }
    }
};
```

使用 CQL 在现有表上启用自动缩放

对于现有的 Amazon Keyspaces 表，您可以使用语句为该表的写入或读取容量启用自动缩放。`ALTER TABLE`如果您要更新当前处于按需容量模式的表，`capacity_mode`则为必填项。如果您的表已处于预置容量模式，则可以省略此字段。

> **Note**
>
> Amazon Keyspaces 自动扩缩需要存在一个代表您执行自动扩缩操作的服务相关角色 (`AWSServiceRoleForApplicationAutoScaling_CassandraTable`)。将自动为您创建此角色。有关更多信息，请参阅the section called “使用服务相关角色”。

在以下示例中，该语句更新了处于按需容量模式的表 my table。该语句将表的容量模式更改为启用了 auto Scaling 的预配置模式。

写入容量配置在 5-10 个容量单位范围内，目标值为 50%。读取容量也配置在 5-10 个容量单位的范围内，目标值为 50%。对于读取容量，您可以将`scale_out_cooldown`和的值设置`scale_in_cooldown`为 60 秒。

```
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
            }
        }
    }
};
```

使用 CQL 查看表的 Amazon Keyspaces 自动扩展配置

要查看表的 auto Scaling 配置的详细信息，请使用以下命令。

```
SELECT * FROM system_schema_mcs.autoscaling WHERE keyspace_name = 'mykeyspace' AND
 table_name = 'mytable';
```

该命令的输出如下所示。

```
 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}}}
```

使用 CQL 关闭表的 Amazon Keyspaces 自动缩放功能

您可以随时关闭表格的 Amazon Keyspaces 自动缩放功能。如果您不再需要扩展表的读取或写入容量，则应考虑关闭自动扩展，这样 Amazon Keyspaces 就不会继续修改表的读取或写入容量设置。您可以使用`ALTER TABLE`语句更新表。

以下语句关闭了表 my table 写入容量的自动缩放。它还会删除以您的名义创建的 CloudWatch 警报。

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

> **Note**
>
> 要删除 Application Auto Scaling 使用的服务相关角色，必须禁用账户中所有表的自动缩放功能。 AWS 区域

使用 CLI 管理 Amazon Keyspaces 扩展策略

要以编程方式更新和管理 Amazon Keyspaces 的自动扩展设置，您可以使用 AWS Command Line Interface (AWS CLI) 或 API。 AWS 要使用 Cassandra 查询语言 (CQL) 管理 Amazon Keyspaces 自动扩展策略，请参阅。the section called “使用 CQL”本主题概述了您可以使用编程方式管理的 auto Scaling 任务。 AWS CLI

有关本主题中描述的 Amazon Key AWS CLI spaces 命令的更多信息，请参阅AWS CLI 命令参考。

主题

开始前的准备工作

在开始之前，您需要完成以下任务。

配置 权限

如果未完成这些任务，您必须为用户配置相应的权限，以创建和管理自动扩展设置。在 AWS Identity and Access Management (IAM) 中，需要使用 AWS 托管策略来管理 Amazon Keys `AmazonKeyspacesFullAccess` paces 扩展策略。有关详细步骤，请参阅the section called “开始之前：向用户授予 Amazon Keyspaces 自动缩放的权限”。

安装 AWS CLI

如果您尚未安装和配置 AWS CLI，则必须先执行此操作。为此，请转到 AWS Command Line Interface 用户指南并按照以下说明进行操作：

- 安装 AWS CLI
- 配置 AWS CLI

使用创建具有自动缩放功能的新表 AWS CLI

创建新的 Amazon Keyspaces 表时，可以在操作中自动为表的写入或读取容量启用自动缩放。`CreateTable`这允许 Amazon Keyspaces 代表您联系 Application Auto Scaling，注册您指定为可扩展目标的表，并调整预配置的写入或读取容量。

有关如何使用 auto Scaling 配置创建多区域表的更多信息，请参阅the section called “使用 auto Scaling (CLI) 在预配置模式下创建新的多区域表”。

> **Note**
>
> Amazon Keyspaces 自动扩展需要服务相关角色 (AWSServiceRoleForApplicationAutoScaling_CassandraTable) 的存在才能代表您执行自动扩展操作。将自动为您创建此角色。有关更多信息，请参阅the section called “使用服务相关角色”。

要以编程方式为表配置自动缩放设置，您可以使用定义 Amazon Keyspaces 自动缩放参数的autoScalingSpecification操作。这些参数定义了指示 Amazon Keyspaces 调整表的预配置吞吐量的条件，以及要采取哪些其他可选操作。在此示例中，您定义了 mytable 的自动缩放设置。

该策略包含以下元素：

- autoScalingSpecification— 指定是否允许 Amazon Keyspaces 代表您调整容量吞吐量。您可以分别为读取容量和写入容量启用 auto Scaling。然后，必须为以下参数指定以下参数autoScalingSpecification：
  - writeCapacityAutoScaling— 最大和最小写入容量单位。
  - readCapacityAutoScaling— 最大和最小读取容量单位。
  - scalingPolicy— Amazon Keyspaces 支持目标跟踪政策。要定义目标跟踪策略，请配置以下参数。
    - targetValue— Amazon Keyspaces 自动扩展可确保已用容量与预配置容量的比率保持在该值或接近该值。您将 targetValue 定义为百分比。
    - disableScaleIn:（可选）Aboolean，它指定该表scale-in是禁用还是启用。默认情况下，此参数处于禁用状态。要开启scale-in，请将该boolean值设置为FALSE。这意味着代表您自动缩小餐桌的容量。
    - scaleOutCooldown – 横向扩展活动会增加表的预置吞吐量。要为横向扩展活动增加冷却时间，请为 ScaleOutCooldown 指定一个值（以秒为单位）。默认值是 0。有关目标跟踪和冷却时间的更多信息，请参阅 App lication Auto Scaling 用户指南中的目标跟踪扩展策略。
    - scaleInCooldown – 横向缩减活动会减小表的预置吞吐量。要为缩减活动增加冷却时间，请为 ScaleInCooldown 指定一个值（以秒为单位）。默认值是 0。有关目标跟踪和冷却时间的更多信息，请参阅 App lication Auto Scaling 用户指南中的目标跟踪扩展策略。

> **Note**
>
> 为了进一步了解 TargetValue 的工作原理，假设您的表的预配置吞吐量设置为 200 个写入容量单位。您决定为此表创建扩展策略，并使用 TargetValue 的 70%。现在假设您开始将写入流量驱动到表，以便实际写入吞吐量为 150 个容量单位。现在的 consumed-to-provisioned 比率是（150/200），或75%。此比率超过了您的目标，因此 auto scaling 会将预配置的写入容量增加到 215，因此该比率为 (150/215)，即 69.77%，TargetValue尽可能接近您的目标，但不超过该比率。

对于 mytable，您可以将读取和写入容量都设置为 TargetValue 50%。Amazon Keyspaces 自动扩展可在 5-10 个容量单位范围内调整表的预配置吞吐量，使该 consumed-to-provisioned 比率保持在或接近 50%。对于读取容量，您可以将ScaleOutCooldown和的值设置ScaleInCooldown为 60 秒。

创建具有复杂自动缩放设置的表时，从 JSON 文件加载自动缩放设置会很有帮助。对于以下示例，您可以从 auto-scaling.zip 下载示例 JSON 文件并进行提取auto-scaling.json，同时记下该文件的路径。在此示例中，JSON 文件位于当前目录中。有关不同的文件路径选项，请参阅如何从文件加载参数。

```
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
```

使用在现有表上启用自动缩放 AWS CLI

对于现有的 Amazon Keyspaces 表，您可以使用操作为表的写入或读取容量启用自动缩放。UpdateTable有关如何更新多区域表的 auto Scaling 设置的更多信息，请参阅the section called “更新多区域表 (CLI) 的预配置容量和 auto Scaling 设置”。

> **Note**
>
> Amazon Keyspaces 自动扩缩需要存在一个代表您执行自动扩缩操作的服务相关角色 (AWSServiceRoleForApplicationAutoScaling_CassandraTable)。将自动为您创建此角色。有关更多信息，请参阅the section called “使用服务相关角色”。

您可以使用以下命令为现有表启用 Amazon Keyspaces 的自动缩放功能。表的 auto 缩放设置是从 JSON 文件加载的。对于以下示例，您可以从 auto-scaling.zip 下载示例 JSON 文件并进行提取`auto-scaling.json`，同时记下该文件的路径。在此示例中，JSON 文件位于当前目录中。有关不同的文件路径选项，请参阅如何从文件加载参数。

有关以下示例中使用的自动缩放设置的更多信息，请参阅the section called “使用创建具有自动缩放功能的新表 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
```

使用查看表的 Amazon Keyspaces 自动缩放配置 AWS CLI

要查看表的 auto Scaling 配置，您可以使用`get-table-auto-scaling-settings`操作。以下 CLI 命令就是一个示例。

```
aws keyspaces get-table-auto-scaling-settings --keyspace-name mykeyspace --table-name
 mytable
```

该命令的输出如下所示。

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

使用关闭表的 Amazon Keyspaces 自动缩放功能 AWS CLI

您可以随时关闭表格的 Amazon Keyspaces 自动缩放功能。如果您不再需要扩展表的读取或写入容量，则应考虑关闭自动扩展，这样 Amazon Keyspaces 就不会继续修改表的读取或写入容量设置。您可以使用`UpdateTable`操作更新表。

以下命令关闭表读取容量的 auto 缩放。它还会删除以您的名义创建的 CloudWatch 警报。

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

> **Note**
>
> 要删除 Application Auto Scaling 使用的服务相关角色，必须禁用账户中所有表的自动缩放功能。 AWS 区域

在 Amazon Keyspaces 中有效使用突发容量

Amazon Keyspaces 提供容量暴增功能，为每个分区吞吐量调配提供一定的灵活性。当您未完全使用分区的吞吐量时，Amazon Keyspaces 为稍后的吞吐量突增情况保留一部分未使用的容量以应对用量峰值。

Amazon Keyspaces 当前保留最多五分钟（300 秒）未使用的读取和写入容量。在读取或写入活动偶尔突增期间，这些额外的容量单元会被快速消耗，甚至超过您为表定义的每秒预置的吞吐能力。

Amazon Keyspaces 还可能在不事先通知的情况下，将容量暴增用于后台维护和其他任务。

请注意，这些暴增容量详细信息未来可能发生变化。

如何估计 Amazon Keyspaces 的容量消耗

当您在 Amazon Keyspaces 中读取或写入数据时，您的查询消耗的读/写请求单元 (RRU/WRU) 或读/写容量单位 (RCU/WCU) 的数量取决于运行查询时亚马逊密钥空间必须处理的数据总量。在某些情况下，返回给客户端的数据可能是 Amazon Keyspaces 为处理查询而必须读取的数据的子集。对于条件写入，即使条件检查失败，Amazon Keyspaces 也会消耗写入容量。

要估算请求处理的总数据量，必须考虑行的编码大小和总行数。本主题涵盖一些常见场景和访问模式的示例，以展示 Amazon Keyspaces 如何处理查询以及这如何影响容量消耗。您可以按照示例估算表的容量需求，并使用 Amazon CloudWatch 观察这些用例的读取和写入容量消耗。

有关如何计算 Amazon Keyspaces 中行的编码大小的信息，请参阅。the section called “计算行大小”

主题

范围查询

要查看范围查询的读取容量消耗，我们使用以下示例表，该表使用按需容量模式。

```
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>
```

现在对该表运行以下查询。

```
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;
```

您从查询中收到以下结果集，Amazon Keyspaces 执行的读取操作在一致模式下消耗 2 个 RRU。LOCAL_QUORUM

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

Amazon Keyspaces 消耗 2 个 RRU 来评估包含这些值的行ck3=60ck3=70并处理查询。但是，Amazon Keyspaces 仅返回查询中指定的WHERE条件为真的那一行，即有值的行。ck3=60为了评估查询中指定的范围，在本例中，Amazon Keyspaces 会读取与该范围上限匹配的行ck3 = 70，但不会在结果中返回该行。读取容量消耗基于处理查询时读取的数据，而不是返回的数据。

限制查询

在处理使用该LIMIT子句的查询时，Amazon Keyspaces 会在尝试匹配查询中指定的条件时读取最大页面大小的行。如果 Amazon Keyspaces 找不到符合第一页LIMIT值的足够匹配数据，则可能需要进行一次或多次分页调用。要继续阅读下一页，您可以使用分页令牌。默认页面大小为 1MB。要在使用LIMIT子句时减少读取容量，可以减小页面大小。有关分页的更多信息，请参阅the section called “对结果进行分页”。

举个例子，让我们看一下以下查询。

```
SELECT * FROM my_table WHERE partition_key=1234 LIMIT 1;”
```

如果您不设置页面大小，Amazon Keyspaces 会读取 1MB 的数据，即使它只向您返回 1 行。要让 Amazon Keyspaces 只读取一行，您可以将此查询的页面大小设置为 1。在这种情况下，Amazon Keyspaces 只能读取一行，前提是您没有基于 T ime-to-live 设置或客户端时间戳的过期行。为了减少读取容量，我们建议将页面大小设置为等于该LIMIT值，以减少 Amazon Keyspaces 读取的数据量。

表格扫描

导致全表扫描的查询（例如使用该ALLOW FILTERING选项的查询）是处理的读取次数多于返回结果的查询的另一个示例。读取容量消耗基于读取的数据，而不是返回的数据。

对于表扫描示例，我们在按需容量模式下使用以下示例表。

```
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>
```

默认情况下，Amazon Keyspaces 在按需容量模式下创建包含四个分区的表。在此示例表中，所有数据都存储在一个分区中，其余三个分区为空。

现在对表运行以下查询。

```
SELECT * from amazon_keyspaces.example_table_2;
```

此查询将生成表扫描操作，其中 Amazon Keyspaces 会扫描表的所有四个分区，并在一致性模式下消耗 6 个 RRU。LOCAL_QUORUM首先，Amazon Keyspaces 消耗 3 个 RRU 来读取这三行。pk='pk'然后，Amazon Keyspaces 会消耗额外的 3 个 RRU 来扫描表的三个空分区。由于此查询会生成表扫描，因此 Amazon Keyspaces 会扫描表中的所有分区，包括没有数据的分区。

轻量级事务

轻量级事务 (LWT) 允许您对表数据执行有条件的写入操作。根据评估当前状态的条件插入、更新和删除记录时，条件更新操作非常有用。

在 Amazon Keyspaces 中，所有写入操作都需要 LOCAL_QUORUM 一致性，并且使用 LWT 不收取额外费用。LWT 的不同之处在于，当 LWT 条件检查结果为 FALSE 时，它会消耗写入容量单位。消耗的写入容量单位数取决于行的大小。如果行大小为 2 KB，则失败的条件写入将消耗两个写入容量单位。如果表中当前不存在该行，则该操作将消耗一个写入容量单位。通过监控中的ConditionalCheckFailed指标， CloudWatch 您可以确定 LWT 条件检查失败所消耗的容量。

估计 Amazon 的读取和写入容量消耗 CloudWatch

要估算和监控读取和写入容量消耗，您可以使用 CloudWatch 控制面板。有关 Amazon Keyspaces 可用指标的更多信息，请参阅。the section called “指标与维度”

要监控特定语句使用的读取和写入容量单位 CloudWatch，可以按照以下步骤操作。

1. 使用示例数据创建新表

2. 为表格配置 Amazon Keyspaces CloudWatch 控制面板。首先，你可以使用 Github 上提供的仪表板模板。
3. 运行 CQL 语句，例如使用`ALLOW FILTERING`选项，然后在控制面板中检查全表扫描所消耗的读取容量单位。

使用 Amazon Keyspaces（Apache Cassandra 兼容）中的键空间、表和行

本章详细介绍如何使用 Amazon Keyspaces（Apache Cassandra 兼容）中的键空间、表和行等。要了解如何使用 Amazon 监控密钥空间和表 CloudWatch，请参阅the section called “使用监控 CloudWatch”。

主题

使用 Amazon Keyspaces 中的键空间

本节详细介绍如何使用 Amazon Keyspaces（Apache Cassandra 兼容）中的键空间。

主题

使用 Amazon Keyspaces 中的系统键空间

Amazon Keyspaces 使用四个系统键空间：

- `system`
- `system_schema`
- `system_schema_mcs`
- `system_multiregion_info`

以下各节详细介绍了 Amazon Keyspaces 支持的系统密钥空间和系统表。

system

这是一个 Cassandra 键空间。Amazon Keyspaces 使用以下表。

表名称	列名称	注释
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	有关本地键空间的信息。
peers	peer, data_center, host_id, preferred_ip, rack, release_version, rpc_address, schema_version, tokens	查询此表可查看可用的端点。例如，如果您通过公共终端节点进行连接，则会看到包含九个可用 IP 地址的列表。如果您通过 FIPS 端点进行连接，则会看到三个 IP 地址的列表。如果您通过 AWS PrivateLink VPC 终端节点进行连接，则会看到已配置的 IP 地址列表。有关更多信息，请参阅the section called “使用接口 VPC 端点信息填充 system.peers 表条目”。

表名称	列名称	注释
size_estimates	keyspace_name, table_name, range_start, range_end, mean_partition_size, partitions_count	此表定义了每个表的每个令牌范围的总大小和分区数。Apache Cassandra Spark Connector 需要此表，它使用估算的分区大小来分配工作。
prepared_statements	prepared_id, logged_keyspace, query_string	此表包含有关已保存查询的信息。

system_schema

这是一个 Cassandra 键空间。Amazon Keyspaces 使用以下表。

表名称	列名称	注释
keyspaces	keyspace_name, durable_writes, replication	有关特定键空间的信息。
tables	keyspace_name, table_name, bloom_filter_fp_chance, caching, comment, compaction, compression, crc_check_chance, dclocal_read_repair_chance, default_time_to_live, extensions, flags, gc_grace_seconds, id, max_index_interval, memtable_flush_period_in_ms, min_index	有关特定表的信息。

表名称	列名称	注释
	_interval, read_repair_chance, speculative_retry	
columns	keyspace_name, table_name, column_name, clustering_order, column_name_bytes, kind, position, type	有关特定列的信息。

system_schema_mcs

这是一个 Amazon Keyspaces 密钥空间，用于存储有关或 AWS 亚马逊密钥空间特定设置的信息。

表名称	列名称	注释
keyspaces	keyspace_name, durable_writes, replication	查询此表可以编程方式了解是否已创建键空间。有关更多信息，请参阅the section called “创建键空间”。
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_li	查询此表可了解特定表的状态。有关更多信息，请参阅the section called “创建表”。 您也可以查询此表，列出特定于 Amazon Keyspaces 且存储为的设置。custom_properties 例如： • capacity_mode • client_side_timestamps

表名称	列名称	注释
	ve, read_repair_chance, max_index_interval, extensions, compaction, comment, id, compression, memtable_flush_period_in_ms, status	• encryption_specification • point_in_time_recovery • ttl
tables_history	keyspace_name, table_name, event_time, creation_time, custom_properties, event	查询此表可了解特定表的模式更改。
columns	keyspace_name, table_name, column_name, clustering_order, column_name_bytes, kind, position, type	此表与 system_schema 键空间中的 Cassandra 表相同。
tags	resource_id, keyspace_name, resource_name, resource_type, tags	查询此表可了解键空间是否具有标签。有关更多信息，请参阅the section called “使用 CQL 向新的或现有的键空间和表添加标签”。

表名称	列名称	注释
autoscaling	keyspace_name, table_name, provisioned_read_capacity_autoscaling_update, provisioned_write_capacity_autoscaling_update	查询此表以获取已配置表的 auto Scaling 设置。请注意，在表格处于活动状态之前，这些设置才可用。要查询此表，必须在 WHERE 子句中指定 keyspace_name 和 table_name 。有关更多信息，请参阅the section called “使用 CQL”。

system_multiregion_info

这是一个 Amazon Keyspaces 键空间，用于存储有关多区域复制的信息。

表名称	列名称	注释
tables	keyspace_name, table_name, region, status	此表包含有关多区域表的信息，例如，复制 AWS 区域 该表的位置和表的状态。您也可以查询此表，列出存储为 Amazon Keyspaces 的特定设置。custom_properties 例如： • capacity_mode 要查询此表，必须在 WHERE 子句中指定 keyspace_name 和 table_name 。有关更多信息，请参阅the section called “创建多区域键空间 (CQL)”。

表名称	列名称	注释
autoscaling	keyspace_name, table_name, provisioned_read_capacity_autoscaling_update, provisioned_write_capacity_autoscaling_update, region	查询此表以获取多区域预配置表的 auto scaling 设置。请注意，在表格处于活动状态之前，这些设置才可用。要查询此表，必须在 WHERE 子句中指定 keyspace_name 和 table_name 。有关更多信息，请参阅the section called “使用 CQL”。

在 Amazon Keyspaces 中创建键空间

Amazon Keyspaces 以异步方式执行数据定义语言 (DDL) 操作，例如创建和删除键空间。

您可以在中监控新密钥空间的创建状态 AWS Management Console，该状态会指示密钥空间何时处于待处理状态或处于活动状态。您还可以使用 system_schema_mcs 键空间以编程方式监控新键空间的创建状态。当密钥空间准备就绪可供使用时，system_schema_mcskeyspaces表中就会变为可见。

要检查新键空间何时可供使用，推荐的设计模式是轮询 Amazon Keyspaces system_schema_mcs keyspaces 表 (system_schema_mcs.*)。有关键空间的 DDL 语句列表，请参阅 CQL 语言参考中的the section called “Keyspaces”部分。

以下查询显示是否已成功创建键空间。

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

对于已成功创建的密钥空间，查询的输出如下所示。

```
keyspace_name | durable_writes  | replication
--------------+-----------------+--------------
   mykeyspace | true            |{...} 1 item
```

使用 Amazon Keyspaces 中的表

本节详细介绍如何使用 Amazon Keyspaces（Apache Cassandra 兼容）中的表。

主题

在 Amazon Keyspaces 中创建表

Amazon Keyspaces 以异步方式执行数据定义语言 (DDL) 操作，例如创建和删除表。您可以在中监控新表的创建状态 AWS Management Console，该状态会指示表何时处于待处理状态或活动状态。您还可以使用系统模式表以编程方式监控新表的创建状态。

当表可供使用时，会在系统模式中显示为活动状态。要检查新的表何时可供使用，推荐的设计模式是轮询 Amazon Keyspaces 系统架构表 (`system_schema_mcs.*`)。有关表的 DDL 语句列表，请参阅 CQL 语言参考中的the section called “表”部分。

以下查询显示表的状态。

```
SELECT keyspace_name, table_name, status FROM system_schema_mcs.tables WHERE
 keyspace_name = 'mykeyspace' AND table_name = 'mytable';
```

对于仍在创建且处于待处理状态的表，此查询的输出如下所示。

```
keyspace_name | table_name | status
--------------+------------+--------
   mykeyspace |    mytable | CREATING
```

对于已成功创建且处于活动状态的表，此查询的输出如下所示。

```
keyspace_name | table_name | status
--------------+------------+--------
   mykeyspace |    mytable | ACTIVE
```

使用 Amazon Keyspaces 中的多区域表

多区域表必须通过以下两种方式之一配置写入吞吐量容量：

- 按需容量模式，以写入请求单位 (WRU) 衡量
- 带有 auto Scaling 功能的预置容量模式，以写入容量单位 (WCU) 衡量

您可以将预置容量模式与 auto scaling 或按需容量模式配合使用，以帮助确保多区域表具有足够的容量来执行对所有表的复制写入。 AWS 区域

> **Note**
>
> 在其中一个区域中更改表的容量模式会更改所有副本的容量模式。

默认情况下，Amazon Keyspaces 对多区域表使用按需模式。在按需模式下，您无需指定您期望应用程序执行的读取和写入吞吐量。Amazon Keyspaces 可在您的工作负载上升或下降到之前达到的任何流量水平时立即适应您的工作负载。如果工作负载的流量水平达到新的峰值，Amazon Keyspaces 会迅速调整以适应工作负载。

如果您为表选择预置容量模式，则必须配置应用程序所需的每秒读取容量单位 (RCU) 和写入容量单位 (WCU) 的数量。

要规划多区域表的吞吐容量需求，应首先估计每个区域每秒所需的 WCU 数量。然后，将表复制到的所有区域的写入数据相加，然后使用该总和为每个区域配置容量。这是必需的，因为在一个区域中执行的每一次写入也必须在每个副本区域中重复。

如果该表没有足够的容量来处理来自所有区域的写入，则会出现容量异常。此外，区域间复制的等待时间将会增加。

例如，如果您有一个多区域表，预计美国东部（弗吉尼亚北部）每秒 5 次写入，美国东部（俄亥俄州）每秒 10 次写入，欧洲（爱尔兰）每秒 5 次写入，则应预计该表在每个区域消耗 20 个 WCU：美国东部（弗吉尼亚北部）、美国东部（俄亥俄州）和欧洲（爱尔兰）。这意味着在本示例中，您需要为表的每个副本预配置 20 个 WCU。您可以使用 Amazon 监控表的容量消耗 CloudWatch。有关更多信息，请参阅the section called “使用监控 CloudWatch”。

由于每次多区域写入按照 WCU 的 1.25 倍计费，因此在本示例中，您总共会看到 75 个 WCU 计费。有关定价的更多信息，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定价。

有关使用 Amazon Keyspaces 自动扩展the section called “使用 auto scaling 管理吞吐容量”的预配置容量的更多信息，请参阅。

> **Note**
>
> 如果表在带有 auto Scaling 的预配置容量模式下运行，则允许每个区域的预配置写入容量在这些自动扩展设置内浮动。

Amazon Keyspaces 中的静态列

在包含聚类列的 Amazon Keyspaces 表中，您可以使用 `STATIC` 关键字来创建静态列。存储在静态列中的值在逻辑分区的所有行之间共享。当您更新此列的值时，Amazon Keyspaces 会自动将更改应用于该分区中的所有行。

本节介绍在写入静态列时如何计算编码数据大小。此过程与将数据写入行中的非静态列的过程是分开处理的。除了静态数据的大小限额外，对静态列的读取和写入操作也会单独影响表的计量和吞吐容量。

计算 Amazon Keyspaces 中每个逻辑分区的静态列大小

本节详细介绍如何估算 Amazon Keyspaces 中的编码静态列大小。在计算账单和限额使用量时，应使用编码大小。在计算表的预置吞吐容量需求时，也应使用编码大小。要计算 Amazon Keyspaces 中的编码静态列大小，可以遵循以下准则。

- 分区键最多可包含 2048 个字节的数据。分区键中的每个键列最多需要 3 个字节的元数据。这些元数据字节计入每个分区 1MB 的静态数据大小限额。计算静态数据大小时，应假设每个分区键列使用全部 3 个字节的元数据。
- 根据数据类型使用静态列数据值的原始大小。有关数据类型的更多信息，请参阅 the section called “数据类型”。
- 在元数据的静态数据大小上增加 104 个字节。
- 聚类列和常规非主键列不计入静态数据的大小。要了解如何估算行中的非静态数据大小，请参阅the section called “计算行大小”。

编码静态列的总大小基于以下公式：

```
partition key columns + static columns + metadata = total encoded size of static data
```

考虑以下表示例，其中所有列均为整数类型。此表包含两个分区键列、两个聚类列、一个常规列和一个静态列。

```
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));
```

在此示例中，我们计算以下语句的静态数据大小：

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, static_col1) values(1,2,6);
```

要估算此写入操作所需的总字节数，可以按照以下步骤操作。

1. 通过将存储在一个分区键列中的数据类型的字节和元数据字节相加，计算该列的大小。针对所有分区键列重复此操作。

   a. 计算分区键第一列 (pk_col1) 的大小：

   ```
   4 bytes for the integer data type + 3 bytes for partition key metadata = 7
    bytes
   ```

   b. 计算分区键第二列 (pk_col2) 的大小：

   ```
   4 bytes for the integer data type + 3 bytes for partition key metadata = 7
    bytes
   ```

   c. 将两列相加，得出分区键列的估算总大小：

   ```
   7 bytes + 7 bytes = 14 bytes for the partition key columns
   ```

2. 加上静态列的大小。在此示例中，我们只有一个静态列用于存储一个整数（需要 4 个字节）。
3. 最后，要得出编码静态列数据的总大小，请将主键列和静态列的字节相加，再加上元数据所需的 104 个字节：

   ```
   14 bytes for the partition key columns + 4 bytes for the static column + 104 bytes
    for metadata = 122 bytes.
   ```

您也可以使用相同的语句更新静态数据和非静态数据。要估算写入操作的总大小，必须先计算非静态数据更新的大小。然后计算行更新的大小（如the section called “计算行大小”中的示例所示），并将结果相加。

在此示例中，您总共可以写入 2MB，1MB 是最大行大小限额，1MB 是每个逻辑分区的最大静态数据大小限额。

要计算同一语句中静态数据和非静态数据更新的总大小，可以使用以下公式：

```
(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
```

考虑以下表示例，其中所有列均为整数类型。此表包含两个分区键列、两个聚类列、一个常规列和一个静态列。

```
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));
```

在此示例中，我们在向表中写入一行时计算数据的大小，如以下语句所示：

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, ck_col1, ck_col2, reg_col1,
 static_col1) values(2,3,4,5,6,7);
```

要估算此写入操作所需的总字节数，可以按照以下步骤操作。

1. 按照之前的步骤计算编码静态数据的总大小。在此示例中，它是 122 个字节。
2. 按照the section called “计算行大小”中的步骤，根据非静态数据的更新，加上编码行总大小。在此示例中，行更新的总大小为 134 个字节。

   ```
   122 bytes for static data + 134 bytes for nonstatic data = 256 bytes.
   ```

计量 Amazon Keyspaces 中的静态数据的读取/写入操作

静态数据与 Cassandra 中的逻辑分区相关联，而不是与单个行相关联。Amazon Keyspaces 中的逻辑分区跨越多个物理存储分区，实际上可以不受大小限制。因此，Amazon Keyspaces 分别计量静态数

据和非静态数据的写入操作。此外，同时包含静态数据和非静态数据的写入操作需要额外的底层操作来提供数据一致性。

如果您执行包含静态数据和非静态数据的混合写入操作，则会产生两个不同的写入操作，一个用于非静态数据，另一个用于静态数据。这适用于按需和预置读取/写入容量模式。

以下示例详细介绍了在计算 Amazon Keyspaces 中包含静态列的表的预置吞吐容量需求时，如何估算所需的读取容量单位 (RCU) 和写入容量单位 (WCU)。您可以使用以下公式估算表处理同时包含静态数据和非静态数据的写入操作所需的容量：

```
2 x WCUs required for nonstatic data + 2 x WCUs required for static data
```

例如，如果您的应用程序每秒写入 27KB 的数据，并且每次写入包括 25.5KB 的非静态数据和 1.5KB 的静态数据，则您的表需要 56 个 WCU（2 x 26 个 WCU + 2 x 2 个 WCU）。

Amazon Keyspaces 计量静态数据和非静态数据的读取方式与计量多行读取的方式相同。因此，在同一操作中读取静态数据和非静态数据的价格取决于为执行读取而处理的数据的总大小。

要了解如何使用 Amazon 监控无服务器资源 CloudWatch，请参阅the section called “使用监控 CloudWatch”。

使用 Amazon Keyspaces 中的行

本节详细介绍如何使用 Amazon Keyspaces（Apache Cassandra 兼容）中的行。表是 Amazon Keyspaces 中的主要数据结构，而表中的数据则按列和行进行组织。

主题

计算 Amazon Keyspaces 中的行大小

Amazon Keyspaces 提供完全托管式存储，可提供个位数毫秒级的读取和写入性能，还能在多个 AWS 可用区中持久存储数据。Amazon Keyspaces 将元数据附加到所有行和主键列，以支持高效的数据访问和高可用性。

本节详细介绍如何估算 Amazon Keyspaces 中的编码行大小。在计算账单和限额使用量时，应使用编码行大小。在计算表的预置吞吐容量需求时，也应使用编码行大小。要计算 Amazon Keyspaces 中的编码行大小，可以遵循以下准则。

- 对于常规列（即不是主键的列）、聚类列或 STATIC 列，根据数据类型使用单元格数据的原始大小并加上所需的元数据。有关 Amazon Keyspaces 中支持的数据类型的更多信息，请参阅the section called “数据类型”。下面列出了 Amazon Keyspaces 存储数据类型值和元数据的方式的一些主要区别。
- 每个列名称所需的空间使用列标识符进行存储，并添加到存储在列中的每个数据值。列标识符的存储值取决于表中列的总数：
  - 1–62 个列：1 个字节
  - 63–124 个列：2 个字节
  - 125–186 个列：3 个字节

  每增加 62 个列，添加 1 个字节。请注意，在 Amazon Keyspaces 中，使用单个 INSERT 或 UPDATE 语句最多可修改 225 个常规列。有关更多信息，请参阅the section called “Amazon Keyspaces 服务限额”。
- 分区键最多可包含 2048 个字节的数据。分区键中的每个键列最多需要 3 个字节的元数据。计算行大小时，应假设每个分区键列使用全部 3 个字节的元数据。
- 聚类列最多可存储 850 个字节的数据。除了数据值的大小外，每个聚类列还要求元数据占数据值大小的 20%。计算行大小时，应为每 5 个字节的聚类列数据值添加 1 个字节的元数据。
- Amazon Keyspaces 将每个分区键和聚类键列的数据值存储两次。额外的开销用于高效查询和内置索引。
- Cassandra ASCII、TEXT 和 VARCHAR 字符串数据类型均使用采用 UTF-8 二进制编码的统一码存储在 Amazon Keyspaces 中。Amazon Keyspaces 中的字符串大小等于 UTF-8 编码的字节数。
- Cassandra INT、BIGINT、SMALLINT 和 TINYINT 数据类型以长度可变的数据值（最多可包含 38 位有效数字）的形式存储在 Amazon Keyspaces 中。系统会删减开头和结尾的 0。其中任何一种数据类型的大小约为每两个有效数字 1 个字节 + 1 个字节。
- Amazon Keyspaces 中的 BLOB 以该值的原始字节长度进行存储。
- Null 值或 Boolean 值的大小为 1 个字节。
- 存储集合数据类型（例如 LIST 或 MAP）的列需要 3 个字节的元数据，无论其内容如何。LIST 或 MAP 的大小为（列 ID）+ 总和（嵌套元素的大小）+（3 个字节）。空 LIST 或 MAP 的大小为（列 ID）+（3 个字节）。每个单个 LIST 或 MAP 元素还需要 1 个字节的元数据。
- STATIC 列数据不计入 1MB 的最大行大小。要计算静态列的数据大小，请参阅the section called “计算每个逻辑分区的静态列大小”。
- 启用该功能后，系统会为每行中的每个列存储客户端时间戳。这些时间戳大约占用 20-40 字节（取决于您的数据），并会增加该行的存储和吞吐量成本。有关更多信息，请参阅the section called “Amazon Keyspaces 中的客户端时间戳”。

- 在行元数据的每行大小上增加 100 个字节。

编码数据行的总大小基于以下公式：

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

> **Important**
>
> 所有列元数据（例如列 ID、分区键元数据、聚类列元数据以及客户端时间戳和行元数据）均计入 1MB 的最大行大小。

考虑以下表示例，其中所有列均为整数类型。此表包含两个分区键列、两个聚类列和一个常规列。由于此表包含五列，因此列名称标识符所需的空间为 1 个字节。

```
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));
```

在此示例中，我们在向表中写入一行时计算数据的大小，如以下语句所示：

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, ck_col1, ck_col2, reg_col1)
 values(1,2,3,4,5);
```

要估算此写入操作所需的总字节数，可以按照以下步骤操作。

1. 通过将存储在一个分区键列中的数据类型的字节和元数据字节相加，计算该列的大小。针对所有分区键列重复此操作。

   a. 计算分区键第一列 (pk_col1) 的大小：

   ```
   (2 bytes for the integer data type) x 2 + 1 byte for the column id + 3 bytes
    for partition key metadata = 8 bytes
   ```

   b. 计算分区键第二列 (pk_col2) 的大小：

   ```
   (2 bytes for the integer data type) x 2 + 1 byte for the column id + 3 bytes
    for partition key metadata = 8 bytes
   ```

c. 将两列相加，得出分区键列的估算总大小：

```
8 bytes + 8 bytes = 16 bytes for the partition key columns
```

2. 通过将存储在一个聚类列中的数据类型的字节和元数据字节相加，计算该列的大小。针对所有聚类列重复此操作。

a. 计算聚类列第一列 (ck_col1) 的大小：

```
(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. 计算聚类列第二列 (ck_col2) 的大小：

```
(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. 将两列相加，得出聚类列的估算总大小：

```
6 bytes + 6 bytes = 12 bytes for the clustering columns
```

3. 加上常规列的大小。在此示例中，我们只有一个列用于存储一个个位数整数，这需要 2 个字节，另外列 ID 还需要 1 个字节。
4. 最后，要获得编码行总大小，请将所有列的字节相加，再加上行元数据所需的 100 个字节：

```
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.
```

要了解如何使用 Amazon CloudWatch 监控无服务器资源，请参阅the section called “使用监控 CloudWatch”。

使用 Amazon Keyspaces 中的查询

本节介绍如何使用 Amazon Keyspaces（Apache Cassandra 兼容）中的查询。可用于查询、转换和管理数据的 CQL 语句是 SELECT、INSERT、UPDATE 和 DELETE。以下主题概述了使用查询时可用的一些比较复杂的选项。有关完整的语言语法及其示例，请参阅 the section called “DML 语句”。

主题

- 对 Amazon Keyspaces 中的结果排序
- 对 Amazon Keyspaces 中的结果进行分页

在 Amazon Keyspaces 中将 **IN** 运算符与 **SELECT** 语句配合使用

SELECT IN

您可以使用 SELECT 语句查询表中的数据，这将读取表中一行或多行的一列或多列，并返回包含与请求匹配的行的结果集。SELECT 语句包含一个 select_clause，该子句用于确定读取并在结果集中返回哪些列。该子句可包含在返回数据之前对数据进行转换的指令。可选的 WHERE 子句指定必须查询哪些行，该子句由作为主键一部分的列上的关系组成。Amazon Keyspaces 支持在 IN 子句中使用 WHERE 关键字。本节通过示例来说明 Amazon Keyspaces 如何处理包含 IN 关键字的 SELECT 语句。

以下示例展示了 Amazon Keyspaces 如何将包含 IN 关键字的 SELECT 语句分解为子查询。在此示例中，我们使用名为 my_keyspace.customers 的表。此表包含一个主键列 department_id、两个聚类列 sales_region_id 和 sales_representative_id，以及在 customer_name 列中包含客户姓名的一个列。

```
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
```

使用此表，您可以通过在 WHERE 子句中使用 IN 关键字，运行以下 SELECT 语句来查找您感兴趣的部门和销售区域中的客户。下面是一个示例语句。

```
SELECT * FROM my_keyspace.customers WHERE department_id IN (0, 1) AND sales_region_id
 IN (0, 1);
```

Amazon Keyspaces 将此语句分为四个子查询，如以下输出所示。

```
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
```

使用 IN 关键字时，在以下任一情况下，Amazon Keyspaces 都会自动对结果进行分页：

- 每处理 10 个子查询之后。
- 处理 1MB 的逻辑 IO 之后。
- 如果您配置了 PAGE SIZE，则 Amazon Keyspaces 会在根据设置的 PAGE SIZE 读取要处理的查询数量后进行分页。
- 如果您使用 LIMIT 关键字来减少返回的行数，则 Amazon Keyspaces 会在根据设置的 LIMIT 读取要处理的查询数量后进行分页。

下表通过一个示例来具体说明。

有关分页的更多信息，请参阅the section called “对结果进行分页”。

```
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
```

您可以对此表运行以下语句来查看分页的运作方式。

```
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 将此语句处理为 24 个子查询，因为此查询中包含的所有 IN 词语的笛卡尔乘积的基数为 24。

```
 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
```

以下示例展示了如何在包含 IN 关键字的 SELECT 语句中使用 ORDER BY 子句。

```
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
```

子查询按照查询中显示分区键和聚类键列的顺序进行处理。在下面的示例中，首先处理分区键值为“2”的子查询，然后处理分区键值为“3”和“1”的子查询。给定子查询的结果根据查询的排序子句（如有）或表创建期间定义的表的聚类顺序进行排序。

```
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
```

对 Amazon Keyspaces 中的结果排序

ORDER BY 子句指定 SELECT 语句中返回的结果的排序顺序。此语句将一系列列名称作为参数，您可以为每个列指定数据的排序顺序。您只能在排序子句中指定聚类列，不能指定非聚类列。

可用于返回的结果的两个排序顺序选项是 ASC（用于升序排序顺序）和 DESC（用于降序排序顺序）。

```
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
```

如果您未在查询语句中指定排序顺序，则使用聚类列的默认排序顺序。

您可以在排序子句中使用的可能排序顺序取决于创建表时为每个聚类列分配的排序顺序。查询结果只能按照创建表时为所有聚类列定义的顺序或与定义的排序顺序相反的顺序进行排序。不允许使用其他可能的组合。

例如，如果表的 CLUSTERING ORDER 是 (col1 ASC, col2 DESC, col3 ASC)，则 ORDER BY 的有效参数是 (col1 ASC, col2 DESC, col3 ASC) 或 (col1 DESC, col2 ASC, col3 DESC)。有关 CLUSTERING ORDER 的更多信息，请参阅 the section called “CREATE TABLE” 下的 table_options。

对 Amazon Keyspaces 中的结果进行分页

当为处理 SELECT 语句而读取的数据超过 1MB 时，Amazon Keyspaces 会自动对 SELECT 语句中的结果进行分页。利用分页，SELECT 语句中的结果将分成若干“页”大小为 1MB（或更小）的数据。应用程序可以先处理第一页结果，然后处理第二页结果，依此类推。在处理返回多行的 SELECT 查询时，客户应始终检查分页令牌。

如果客户提供的 PAGE SIZE 需要读取超过 1MB 的数据，则 Amazon Keyspaces 会根据 1MB 的数据读取增量自动将结果分成多页。

例如，如果一行的平均大小为 100KB，而您将 PAGE SIZE 指定为 20，则 Amazon Keyspaces 会在读取 10 行（读取 1000KB 的数据）后自动对数据进行分页。

由于 Amazon Keyspaces 根据为处理请求而读取的行数（而不是结果集中返回的行数）对结果进行分页，因此如果您运行过滤查询，某些页面可能不包含任何行。

例如，如果您将 PAGE SIZE 设置为 10，Keyspaces 评估需要读取 30 行才能处理您的 SELECT 查询，则 Amazon Keyspaces 将返回三页。如果只有一部分行与您的查询相匹配，则某些页面的行数可能少于 10。有关 of LIMIT 查询如何影响读取容量的示例，请参阅the section called “限制查询”。PAGE SIZE

使用 Amazon Keyspaces 中的分区程序

在 Apache Cassandra 中，分区程序控制将数据存储在集群中的哪些节点上。分区程序使用分区键的哈希值创建数字令牌。Cassandra 使用此令牌在节点之间分配数据。客户还可以在 SELECT 操作和 WHERE 子句中使用这些令牌来优化读取和写入操作。例如，客户可以通过在每个并行作业中指定要查询的不同令牌范围，高效地对大型表执行并行查询。

Amazon Keyspaces 提供三种不同的分区程序。

Murmur3Partitioner（默认）

与 Apache Cassandra 兼容的 Murmur3Partitioner。Murmur3Partitioner 是 Amazon Keyspaces 和 Cassandra 1.2 及更高版本中的默认 Cassandra 分区程序。

RandomPartitioner

与 Apache Cassandra 兼容的 RandomPartitioner。RandomPartitioner 是 Cassandra 1.2 之前版本的默认 Cassandra 分区程序。

Keyspaces 默认分区程序

DefaultPartitioner 返回与 RandomPartitioner 相同的 token 函数结果。

分区程序设置在账户级别按区域应用。例如，如果您更改了美国东部（弗吉尼亚州北部）的分区程序，则更改将应用于该区域内同一账户中的所有表。您可以随时安全地更改分区程序。请注意，配置更改需要大约 10 分钟才能完成。更改分区程序设置时，您无需重新加载 Amazon Keyspaces 数据。客户将在下次连接时自动使用新的分区程序设置。

您可以使用 AWS Management Console或 Cassandra 查询语言 (CQL) 更改分区程序。

AWS Management Console

使用 Amazon Keyspaces 控制台更改分区程序

1. 登录 https://console.aws.amazon.com/keyspaces/home，然后打开 AWS Management Console Amazon Keyspaces 控制台。
2. 在导航窗格中，选择 Configuration（配置）。
3. 在配置页面上，转到编辑分区程序。
4. 选择与您的 Cassandra 版本兼容的分区程序。分区程序更改需要大约 10 分钟才能应用。

> **Note**
>
> 配置更改完成后，您必须断开连接并重新连接到 Amazon Keyspaces，这样请求才能使用新的分区程序。

Cassandra Query Language (CQL)

1. 要查看账户配置了哪个分区程序，可以使用以下查询。

```
SELECT partitioner from system.local;
```

如果未更改分区程序，则查询将显示以下输出。

```
partitioner
-------------------------------------------
com.amazonaws.cassandra.DefaultPartitioner
```

2. 要将分区程序更新为 Murmur3 分区程序，可以使用以下语句。

```
UPDATE system.local set
 partitioner='org.apache.cassandra.dht.Murmur3Partitioner' where key='local';
```

3. 请注意，此配置更改需要大约 10 分钟才能完成。要确认是否已设置分区程序，可以再次运行 SELECT 查询。请注意，由于最终读取一致性，响应反映的可能还不是最近完成的分区程序更改的结果。如果您稍等片刻后再次重复 SELECT 操作，响应应会返回最新数据。

```
SELECT partitioner from system.local;
```

> **Note**
>
> 您必须断开连接并重新连接到 Amazon Keyspaces，这样请求才能使用新的分区程序。

使用 Amazon Keyspaces 资源的标签和标签

您可以使用标签来标记 Amazon Keyspaces（Apache Cassandra 兼容）资源。标签可让您按各种方法对资源进行分类，例如，按用途、所有者、环境或其他标准。标签可帮助您：

- 根据您分配到资源的标签来快速识别资源。
- 查看按标签细分的 AWS 账单。
- 根据标签控制对 Amazon Keyspaces 资源的访问。有关使用标签的 IAM 策略示例，请参阅 the section called “基于 Amazon Keyspaces 标签的授权”。

亚马逊弹性计算云 (Amazon EC2)、亚马逊简单存储 AWS 服务 (Amazon S3)、亚马逊密钥空间等服务支持标记。有效的标签让您可对具有特定标签的服务创建报告，从而提供成本分析。

要开始使用标签，请执行以下操作：

1. 了解 Amazon Keyspaces 标签限制。
2. 使用 Amazon Keyspaces 的标记操作 创建标签。
3. Amazon Keyspaces 成本分配报告用于跟踪每个有效标签的 AWS 费用。

最后，最佳实践是遵循最佳标签策略。有关信息，请参阅 AWS 标记策略。

Amazon Keyspaces 标签限制

每个标签都由键 和值组成，这两个参数都由您定义。以下限制适用：

- 每个 Amazon Keyspaces 键空间或表的同一个键只能有一个标签。如果您尝试添加现有标签（相同键），现有标签值会更新为新值。

- 应用于键空间的标签不会自动应用于该键空间内的表。要将相同的标签应用于键空间及其所有表，必须为每项资源添加单独的标签。
- 创建多区域键空间或表时，您在创建过程中定义的任何标签都会自动应用于所有区域中的所有键空间和表。当您使用 `ALTER KEYSPACE` 或 `ALTER TABLE` 更改现有标签时，更新仅应用于您进行更改的区域中的键空间或表。
- 值充当标签类别（键）中的描述符。在 Amazon Keyspaces 中，值不能为空或为 null。
- 标签键和值区分大小写。
- 最大键长度为 128 个 Unicode 字符。
- 最大值长度为 256 个 Unicode 字符。
- 允许的字符包括字母、空格和数字，以及以下特殊字符：`+ - = . _ : /`
- 每个资源的最大标签数是 50。
- AWS 分配的标签名称和值将自动被分配 `aws:` 前缀，您无法分配该前缀。AWS 分配的标签名称不计入标签限制 50。用户分配的标签名称在成本分配报告中具有 `user:` 前缀。
- 您不能回溯标签的应用日期。

Amazon Keyspaces 的标记操作

您可以使用 Amazon Keyspaces（Apache Cassandra 兼容）控制台、AWS CLI 或 Cassandra 查询语言 (CQL) 添加、列出、编辑或删除键空间和表的标签。然后，您可以激活这些用户定义的标签，以便在 AWS Billing and Cost Management 控制台上显示这些标签以进行成本分配跟踪。有关更多信息，请参阅 Amazon Keyspaces 成本分配报告。

对于批量编辑，您还可以使用控制台中的标签编辑器。有关更多信息，请参阅 AWS Resource Groups 用户指南中的使用标签编辑器。

主题

- 使用控制台向新的或现有的键空间和表添加标签
- 使用 AWS CLI 向新的或现有的键空间和表添加标签
- 使用 CQL 向新的或现有的键空间和表添加标签

使用控制台向新的或现有的键空间和表添加标签

您可以在创建新的键空间和表时使用 Amazon Keyspaces 控制台向它们添加标签。您还可以添加、编辑或删除现有表的标签。

在创建键空间表时对键空间进行标记（控制台）

1. 登录 https://console.aws.amazon.com/keyspaces/home，然后打开 AWS Management Console Amazon Keyspaces 控制台。
2. 在导航窗格中，选择 Keyspaces (键空间)，然后选择 Create keyspace (创建键空间)。
3. 在 Create keyspace (创建键空间) 页面上，提供键空间的名称。输入标签的键和值，然后选择 Add new tag (添加新标签)。
4. 选择 Create keyspace (创建键空间)。

在创建表时对表进行标记（控制台）

1. 登录 https://console.aws.amazon.com/keyspaces/home，然后打开 AWS Management Console Amazon Keyspaces 控制台。
2. 在导航窗格中，选择 Tables (表)，然后选择 Create table (创建表)。
3. 在创建表页面的表详细信息部分中，选择一个键空间，并为表提供名称。
4. 在架构部分，为您的表创建架构。
5. 在表设置部分，选择自定义设置。
6. 继续前往表标签 - 可选部分，然后选择添加新标签，以创建新标签。
7. 选择 Create Table（创建表）。

标记现有资源（控制台）

1. 登录 https://console.aws.amazon.com/keyspaces/home，然后打开 AWS Management Console Amazon Keyspaces 控制台。
2. 在导航窗格中，选择 Keyspaces (键空间) 或 Tables (表)。
3. 在列表中选择键空间或表。然后选择 Manage tags (管理标签) 以添加、编辑或删除标签。

有关标签结构的信息，请参阅 Amazon Keyspaces 标签限制。

使用 AWS CLI 向新的或现有的键空间和表添加标签

本节中的示例演示了如何在创建键空间和表时使用 AWS CLI 指定标签、如何为现有资源添加或删除标签以及如何列出标签。

以下示例说明了如何创建带有标签的新表。该命令在已存在的键空间 myKeyspace 中创建了表 myTable。请注意，为了便于阅读，该命令已分成不同的行。

```
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'
```

以下示例说明了如何向现有表添加新标签。

```
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'
```

下一个示例说明了如何列出指定资源的标签。

```
aws keyspaces list-tags-for-resource --resource-arn 'arn:aws:cassandra:us-
east-1:111222333444:/keyspace/myKeyspace/table/myTable'
```

最后一个命令的输出如下所示。

```
{
    "tags": [
        {
            "key": "key1",
            "value": "val1"
        },
        {
            "key": "key2",
            "value": "val2"
        },
        {
            "key": "key3",
            "value": "val3"
        },
        {
            "key": "key4",
            "value": "val4"
        }
    ]
}
```

使用 CQL 向新的或现有的键空间和表添加标签

以下示例演示了在创建键空间和表时如何使用 CQL 指定标签、如何标记现有资源以及如何读取标签。

以下示例创建一个带有标签的新键空间：

```
CREATE KEYSPACE mykeyspace WITH TAGS = {'key1':'val1', 'key2':'val2'} ;
```

以下示例创建带有标签的新表。

```
CREATE TABLE mytable(...) WITH TAGS = {'key1':'val1', 'key2':'val2'};
```

在语句中使用其他命令标记资源。

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

以下示例说明如何添加或删除现有键空间和表上的标签。

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

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

要读取附加到资源的标签，请使用以下 CQL 语句。

```
SELECT * FROM system_schema_mcs.tags WHERE valid_where_clause;
```

WHERE 子句必需，且必须为以下格式之一：

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

示例：

以下查询显示键空间是否具有标签。

```
```

```
SELECT * FROM system_schema_mcs.tags WHERE keyspace_name = 'mykeyspace' AND
 resource_type = 'keyspace';
```

查询的输出如下所示。

```
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'}
```

以下查询显示表的标签。

```
SELECT * FROM system_schema_mcs.tags WHERE keyspace_name = 'mykeyspace' AND
 resource_name = 'mytable';
```

该查询的输出如下所示。

```
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 成本分配报告

AWS 使用标签组织成本分配报告上的资源成本。AWS 提供了两种类型的成本分配标签：

- AWS 生成的标签 AWS 为您定义、创建和应用此标签。
- 用户定义的标签。您定义、创建和应用这些标签。

您必须先分别激活两种类型的标签，然后这些标签才能显示在 Cost Explorer 中或成本分配报告上。

要激活 AWS 生成的标签：

1. 登录 AWS Management Console，打开 Billing and Cost Management 控制台：https://console.aws.amazon.com/billing/home#/。
2. 在导航窗格中，选择 Cost Allocation Tags (成本分配标签)。
3. 在 AWS 生成的成本分配标签下，选择激活。

激活用户定义的标签：

1. 登录 AWS Management Console，打开 Billing and Cost Management 控制台：https://console.aws.amazon.com/billing/home#/。
2. 在导航窗格中，选择 Cost Allocation Tags (成本分配标签)。
3. 在 User-Defined Cost Allocation Tags (用户生成的成本分配标签) 下，选择 Activate (激活)。

创建并激活标签后，AWS 生成成本分配报告，其中按活动标签分组了使用情况和成本。成本分配报告包括您每个账单周期的所有 AWS 成本。该报告包括标记资源和未标记资源，因此您可以清晰地排列资源费用。

> **Note**
>
> 目前，从 Amazon Keyspaces 传出的所有数据不会在成本分配报告上按标签细分。

有关更多信息，请参阅使用成本分配标签。

使用 Amazon Keyspaces 进行设计和架构的最佳实践

通过本节可快速查找使用 Amazon Keyspaces 尽可能提高性能和降低吞吐量成本的建议。

目录

    - 了解 Application Auto Scaling 设置
    - 如何识别具有低目标利用率 (<=50%) 的表
    - 如何处理具有季节性差异的工作负载
    - 如何处理具有未知模式的尖峰工作负载
    - 如何处理具有关联应用程序的工作负载
- 确定未使用的资源
    - 如何确定未使用的资源
    - 确定未使用的表资源
    - 清理未使用的表资源
    - 清理未使用的 point-in-time 恢复 (PITR) 备份
- 评估您的表使用模式
    - 执行更少的强一致性读取操作
    - 启用生存时间（TTL）
- 评估预置容量大小是否合适
    - 如何从 Amazon Keyspaces 表中检索使用指标
    - 如何识别预置不足的 Amazon Keyspaces 表
    - 如何识别过度预置的 Amazon Keyspaces 表

适用于 Amazon Keyspaces 的 NoSQL 设计

NoSQL 数据库系统（如 Amazon Keyspaces）使用替代数据管理模型，如键值对或文档存储。从关系数据库管理系统转向 NoSQL 数据库系统（如 Amazon Keyspaces）时，一定要了解主要区别和特定设计方法。

主题

- 关系数据设计和 NoSQL 之间的区别
- NoSQL 设计的两个关键概念
- 了解 NoSQL 设计

关系数据设计和 NoSQL 之间的区别

关系数据库系统 (RDBMS) 和 NoSQL 数据库各有优劣：

- 在 RDBMS 中，可以灵活查询数据，但查询成本相对较高，不能很好地扩展适应高流量情况（参见 the section called “数据建模”）。
- 在 NoSQL 数据库（如 Amazon Keyspaces）中，高效查询数据的方式有限，此外查询成本高且速度慢。

这些区别使得这两个系统的数据库设计不同：

- 在 RDBMS 中，针对灵活性设计，不必担心实现细节或性能。查询优化通常不会影响架构设计，但标准化很重要。
- 在 Amazon Keyspaces 中，对模式进行专门设计，以尽可能地加快最常见和最重要的查询的速度并尽可能降低其成本。根据业务使用案例的具体要求定制数据结构。

NoSQL 设计的两个关键概念

NoSQL 设计需要不同于 RDBMS 设计的思维模式。对于 RDBMS，可以创建标准化数据模型，不考虑访问模式。以后出现新问题和查询要求后，进行扩展。可以将每种类型的数据整理到各自的表中。

NoSQL 设计的不同之处

- 相比之下，在了解需要解答的问题之前，您不应开始设计 Amazon Keyspaces 的模式。事先了解业务问题和应用程序使用案例十分重要。
- 应在 Amazon Keyspaces 应用程序中保留尽可能少的表。使用较少的表可以提高事物的可扩展性、减少权限管理需求以及降低 Amazon Keyspaces 应用程序的开销。此外还可以帮助降低总体备份成本。

了解 NoSQL 设计

设计 Amazon Keyspaces 应用程序的第一步是确定系统必须满足的具体查询模式。

具体来说，开始前务必了解应用程序访问模式的三个基本属性：

- 数据大小：了解一次存储和请求的数据量将有助于确定对数据进行分区的最有效方法。
- 数据形状：NoSQL 数据库处理查询时不会改变数据形状（RDBMS 系统会这样做），而是整理数据，使数据库中的数据形状与查询内容对应。这是加快速度并增强可扩展性的一个关键因素。
- 数据速度：Amazon Keyspaces 通过增加可用于处理查询的物理分区数量，并在这些分区高效分配数据来进行扩展。事先了解峰值查询负载可能有助于确定数据分区方式，从而最高效地使用 I/O 容量。

确定具体查询要求后，可以根据管理性能的一般原则整理数据：

- 将相关数据放在一起。 对 20 年前的路由表优化研究发现，“参考局部性”是加速响应时间的最重要因素：将相关数据集中放置到一个位置。这一点对于今天的 NoSQL 系统同样成立，将相关数据保留在最接近的位置，将对成本和性能产生重大影响。不应在多个表分布相关数据项目，而应保持 NoSQL 系统中的相关项目尽可能靠近。

  作为一般规则，应在 Amazon Keyspaces 应用程序中保留尽可能少的表。

  例外情况是指涉及大量时间序列数据，或数据集具有明显不同访问模式的情况。具有反向索引的单个表通常支持简单查询创建和检索应用程序所需的复杂层次数据结构。
- 使用排序顺序。 如果键设计能够一起排序，可以将相关项目组织在一起，高效查询。这是一个重要的 NoSQL 设计策略。
- 分发查询。 大量查询不能集中在数据库的某个部分，这样可能超出 I/O 容量，这一点也很重要。应设计数据键，在尽可能多的分区均匀分布流量，从而避免“热点”。

这些一般准则将转化为可用于在 Amazon Keyspaces 中为数据高效建模的一些常见设计模式。

客户端驱动程序与 Amazon Keyspaces（Apache Cassandra 兼容）的连接

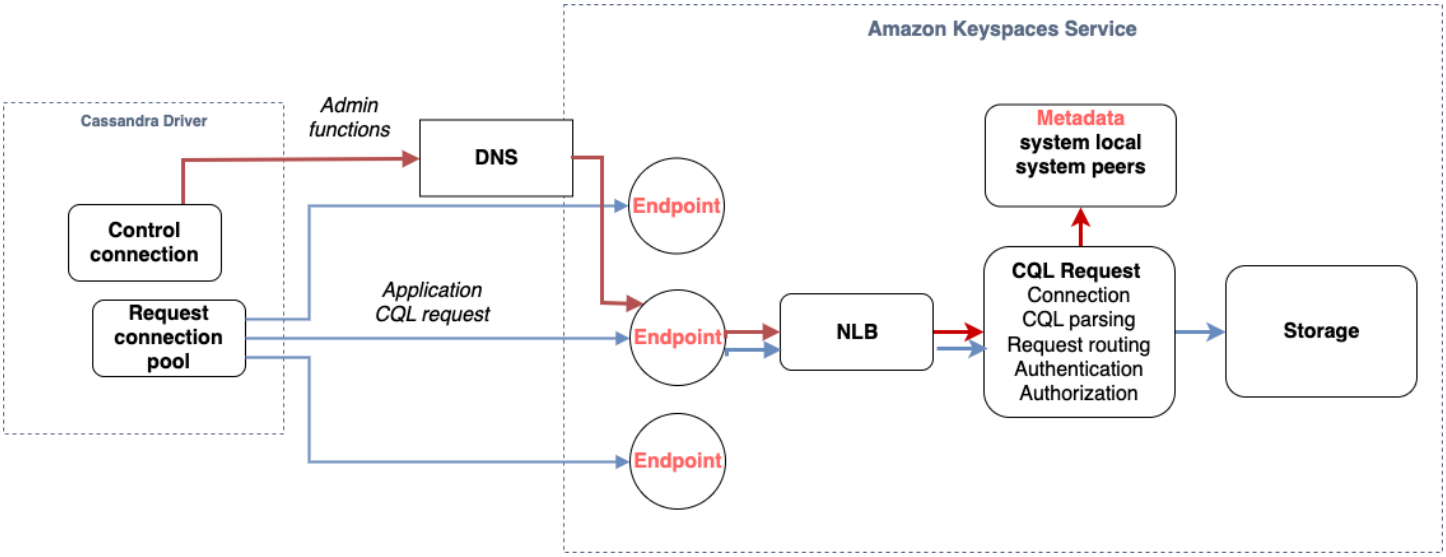
要与 Amazon Keyspaces 通信，您可以使用自己选择的任何现有 Apache Cassandra 客户端驱动程序。由于 Amazon Keyspaces 是一项无服务器服务，我们建议您针对应用程序的吞吐量需求优化客户端驱动程序的连接配置。本主题介绍一些最佳实践，包括如何计算应用程序需要的连接数，以及连接的监控和错误处理。

主题

连接如何在 Amazon Keyspaces 中运作

本节概述客户端驱动程序连接在 Amazon Keyspaces 中的运作方式。由于 Cassandra 客户端驱动程序配置错误可能会导致 Amazon Keyspaces 中出现 `PerConnectionRequestExceeded` 事件，因此需要在客户端驱动程序配置中配置适当数量的连接，以避免这类连接错误和类似的连接错误。

连接到 Amazon Keyspaces 时，驱动程序需要种子端点来建立初始连接。Amazon Keyspaces 使用 DNS 将初始连接路由到众多可用端点之一。系统会将端点附加到网络负载均衡器，网络负载均衡器转而与实例集中的一个请求处理程序建立连接。建立初始连接后，客户端驱动程序将从 `system.peers` 表收集有关所有可用端点的信息。利用这些信息，客户端驱动程序可以创建与所列端点的其他连接。客户端驱动程序可以创建的连接数受客户端驱动程序设置中指定的本地连接数的限制。默认情况下，大多数客户端驱动程序会为每个端点建立一个连接并建立与 Cassandra 的连接池，然后对连接池查询进行负载均衡。尽管可以与同一个端点建立多个连接，但在网络负载均衡器后面，它们可能连接到许多不同的请求处理程序。通过公有端点进行连接时，如果与 `system.peers` 表中列出的 9 个端点中的每个端点建立了 1 个连接，则会导致与不同请求处理程序建立 9 个连接。



如何在 Amazon Keyspaces 中配置连接

Amazon Keyspaces 支持每个 TCP 连接每秒最多 3000 次 CQL 查询。由于驱动程序可以建立的连接数没有限制，因此我们建议每个连接每秒仅定向 500 个 CQL 请求，为开销、流量暴增和负载均衡改进留出余地。请按照以下步骤操作，确保驱动程序的连接已根据应用程序的需求正确配置。

增加驱动程序在其连接池中维护的每个 IP 地址的连接数。

- 大多数 Cassandra 驱动程序都会建立与 Cassandra 的连接池，并对连接池查询进行负载均衡。大多数驱动程序的默认行为是与每个端点建立单个连接。Amazon Keyspaces 向驱动程序公开

了 9 个对等 IP 地址，因此根据大多数驱动程序的默认行为，这会导致建立 9 个连接。Amazon Keyspaces 支持每个 TCP 连接每秒最多 3000 次 CQL 查询，因此，使用默认设置的驱动程序的最大 CQL 查询吞吐量为每秒 27000 次 CQL 查询。如果使用了驱动程序的默认设置，那么单个连接可能必须处理超过最大 CQL 查询吞吐量“每秒 3000 次 CQL 查询”的查询量。这可能会导致 `PerConnectionRequestExceeded` 事件。

- 为避免 `PerConnectionRequestExceeded` 事件，您必须将驱动程序配置为为每个端点创建额外连接以分配吞吐量。
- 作为 Amazon Keyspaces 中的最佳实践，应假设每个连接可以支持每秒 500 次 CQL 查询。
- 这意味着，对于需要支持分布在 9 个可用端点上的预计每秒 27000 次 CQL 查询的生产应用程序，您必须为每个端点配置 6 个连接。这样可以确保每个连接每秒处理的请求不超过 500 个。

根据应用程序的需求，计算您需要为驱动程序配置的每个 IP 地址的连接数。

要确定需要为应用程序配置的每个端点的连接数，请考虑以下示例。您有一个应用程序需要支持每秒 20000 次 CQL 查询，其中包括 10000 个 `INSERT` 操作、5000 个 `SELECT` 操作和 5000 个 `DELETE` 操作。Java 应用程序在 Amazon Elastic Container Service (Amazon ECS) 上的三个实例上运行，其中每个实例都与 Amazon Keyspaces 建立了一个会话。可用于估算您需要为驱动程序配置的连接数的计算使用以下输入。

1. 应用程序需要支持的每秒请求数。
2. 可用实例的数量减去 1（为维护或故障做准备）。
3. 可用端点的数量。如果您通过公有端点进行连接，则有 9 个可用端点。如果您使用 VPC 端点，则有 2 到 5 个可用端点，具体取决于区域。
4. 使用每个连接每秒 500 次 CQL 查询作为 Amazon Keyspaces 的最佳实践。
5. 将结果四舍五入。

在此示例中，公式如下所示。

```
20,000 CQL queries / (3 instances - 1 failure) / 9 public endpoints / 500 CQL queries
 per second = ROUND(2.22) = 3
```

根据此计算，您需要在驱动程序配置中为每个端点指定 3 个本地连接。对于远程连接，为每个端点仅配置 1 个连接。

如何在 Amazon Keyspaces 中通过 VPC 端点配置连接

通过私有 VPC 端点进行连接时，您的可用端点很可能少于 9 个。此外，每个区域的 VPC 端点数量可能会有所不同，具体取决于可用区的数量和分配的 VPC 中的子网数量。美国东部（弗吉尼亚州北部）区域有五个可用区，您最多可以拥有五个 Amazon Keyspaces 端点。美国西部（北加利福尼亚）区域有两个可用区，您最多可以拥有两个 Amazon Keyspaces 端点。端点数不会影响规模，但会增加您需要在驱动程序配置中建立的连接数。考虑以下示例。您的应用程序需要支持 20000 个 CQL 查询，并且在 Amazon ECS 上的三个实例上运行，其中每个实例都与 Amazon Keyspaces 建立了一个会话。唯一的区别是不同 AWS 区域中可用的端点数。

美国东部（弗吉尼亚州北部）区域所需的连接数：

```
20,000 CQL queries / (3 instances - 1 failure) / 5 private VPC endpoints / 500 CQL
 queries per second = 4 local connections
```

美国西部（北加利福尼亚）区域所需的连接数：

```
20,000 CQL queries / (3 instances - 1 failure) / 2 private VPC endpoints / 500 CQL
 queries per second = 10 local connections
```

> **Important**
>
> 使用私有 VPC 端点时，Amazon Keyspaces 需要额外的权限来动态发现可用的 VPC 端点和填充 `system.peers` 表。有关更多信息，请参阅 the section called “使用接口 VPC 端点信息填充 `system.peers` 表条目”。

使用其他终端节点通过私有 VPC 终端节点访问亚马逊密钥空间时 AWS 账户，您可能只能看到一个 Amazon Keyspaces 终端节点。再说一次，这不会影响 Amazon Keyspaces 的可能吞吐量的规模，但可能需要您增加驱动程序配置中的连接数。此示例显示了单个可用端点的相同计算。

```
20,000 CQL queries / (3 instances - 1 failure) / 1 private VPC endpoints / 500 CQL
 queries per second = 20 local connections
```

要了解有关使用共享 VPC 跨账户访问 Amazon Keyspaces 的更多信息，请参阅the section called “共享 VPC 中的跨账户访问”。

如何监控 Amazon Keyspaces 中的连接

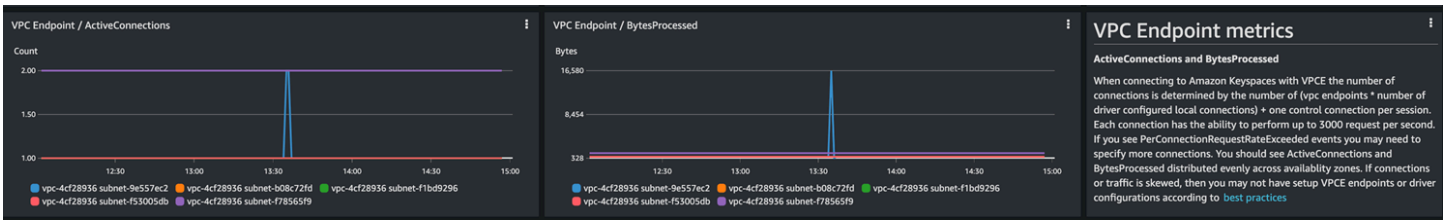
为了帮助确定应用程序连接到的端点数，您可以记录在 system.peers 表中发现的对等节点数。以下示例是 Java 代码的示例，该代码将在连接建立后打印对等节点的数量。

```
ResultSet result = session.execute(new SimpleStatement("SELECT * FROM system.peers"));

logger.info("number of Amazon Keyspaces endpoints:" + result.all().stream().count());
```

> **Note**
>
> CQL 控制台或 AWS 控制台未部署在 VPC 内，因此使用公有终端节点。因此，从位于 VPCE 外部的应用程序运行 system.peers 查询通常会导致产生 9 个对等节点。打印每个对等节点的 IP 地址也可能很有帮助。

您还可以通过设置 VPCE Ama CloudWatch zon 指标来观察使用 VPC 终端节点时的对等节点数量。在中 CloudWatch，您可以看到与 VPC 终端节点建立的连接数量。Cassandra 驱动程序将为每个端点建立连接以发送 CQL 查询，并建立控制连接以收集系统表信息。下图显示了在驱动程序设置中配置了 1 个连接的情况下连接到 Amazon Keyspaces 后的 VPC 终端节点 CloudWatch 指标。该指标显示了六个活动连接，包括一个控制连接和五个常规连接（在可用区中每个端点 1 个连接）。



要开始使用 CloudWatch 图表监控连接数，您可以在 Amazon Keyspaces AWS CloudFormation 模板存储库 GitHub 中部署此模板。

如何处理 Amazon Keyspaces 中的连接错误

当超出每个连接 3000 个请求的限额时，Amazon Keyspaces 会返回一个 PerConnectionRequestExceeded 事件，Cassandra 驱动程序会收到一个 WriteTimeout 或 ReadTimeout 异常。您应该在 Cassandra 重试策略或应用程序中使用指数回退重试此异常。您应该提供指数回退以避免发送额外请求。

默认重试策略会在查询计划中尝试 try next host 操作。由于 Amazon Keyspaces 在连接到 VPC 端点时可能有一到三个可用端点，因此除了 WriteTimeout 和 ReadTimeout 异常之外，您还可能在

应用程序日志中看到 NoHostAvailableException 异常。您可以使用 Amazon Keyspaces 提供的重试策略，这些策略在同一端点上重试，但会跨越不同的连接。

你可以在 Amazon Keyspaces Java 代码示例存储库 GitHub 中找到 Java 指数重试策略的示例。您可以在 Github 上的 Amazon Keyspaces 代码示例存储库中找到其他语言示例。

Amazon Keyspaces（Apache Cassandra 兼容）中的数据建模

本主题介绍 Amazon Keyspaces（Apache Cassandra 兼容）中的数据建模概念。通过本节可查找设计与应用程序数据访问模式一致的数据模型的建议。在使用 Amazon Keyspaces 时，实施数据建模最佳实践可提高性能并尽可能降低吞吐量成本。

要更轻松地可视化和设计数据模型，可以使用 NoSQL Workbench。

主题

- 如何在 Amazon Keyspaces 中高效使用分区键

如何在 Amazon Keyspaces 中高效使用分区键

唯一标识 Amazon Keyspaces 表中的每一行的主键可以包含一个或多个分区键列（用于确定存储数据的分区）以及一个或多个可选的聚类列（用于定义数据在分区内的聚类和排序方式）。

由于分区键确定存储数据的分区数量以及数据在这些分区中的分布方式，因此选择分区键的方式可能会对查询的性能产生很大影响。一般来说，设计应用程序时应确保磁盘上所有分区的活动一致。

将应用程序的读取和写入活动均匀分配到所有分区有助于尽可能降低吞吐量成本，这适用于按需和预置读取/写入容量模式。例如，如果您使用预置容量模式，则可以确定应用程序所需的访问模式，并估算每个表所需的总读取容量单位 (RCU) 和写入容量单位 (WCU)。Amazon Keyspaces 使用您预置的吞吐量支持您的访问模式，前提是针对给定分区的流量不超过 3000 个 RCU 和 1000 个 WCU。

Amazon Keyspaces 通过提供容量暴增为每个分区的吞吐量预置提供更大的灵活性，有关更多信息，请参阅the section called “容量爆增”。

主题

- 在 Amazon Keyspaces 中使用写入分片均匀分配工作负载

在 Amazon Keyspaces 中使用写入分片均匀分配工作负载

在 Amazon Keyspaces 中跨分区更好地分配写入的一种方式是扩展空间。可以通过多种不同方式来进行。您可以添加一个额外的分区键列，将随机数写入到该列，以便在分区间分配行。或者，可以使用基于查询内容计算的数字。

使用复合分区键和随机值进行分片

在分区中更均匀地分配负载的一种策略是添加一个额外的分区键列并在其中写入随机数。然后在更大空间内随机写入。

例如，请考虑下表，该表包含一个代表日期的分区键。

```
CREATE TABLE IF NOT EXISTS tracker.blogs (
   publish_date date,
   title text,
   description int,
   PRIMARY KEY (publish_date));
```

为了在分区间更均匀地分配此表，可以添加一个用于存储随机数的额外分区键列 shard。例如：

```
CREATE TABLE IF NOT EXISTS tracker.blogs (
   publish_date date,
   shard int,
   title text,
   description int,
   PRIMARY KEY ((publish_date, shard)));
```

插入数据时，您可以为 shard 列选择一个介于 1 和 200 之间的随机数。这将生成复合分区键值，比如 (2020-07-09, 1)、(2020-07-09, 2)，以此类推，直到 (2020-07-09, 200)。由于随机化分区键，每天表的写入将均匀分布在多个分区。这样可以提高并行度和总体吞吐量。

但是，要读取给定日期的所有行，必须针对所有分片查询这些行，然后合并结果。例如，先对分区键值 (2020-07-09, 1) 发出 SELECT 语句。然后，对 (2020-07-09, 2) 发出另一个 SELECT 语句，以此类推，直到 (2020-07-09, 200)。最后，应用程序必须合并所有这些 SELECT 语句的结果。

使用复合分区键和计算值进行分片

随机化策略可以显著改善写入吞吐量，但要读取特定行却很困难，因为您不知道在写入该行时，将哪个值写入 shard 列。要让读取单个行更轻松，可使用其他策略。不使用随机数在分区间分配行，而是使用可根据查询内容计算的数字。

考虑上一个示例，表在分区键中使用当天日期。现在假设每个行有一个可访问的 `title` 列，并且除了日期之外，您最常需要按标题查找行。在应用程序将行写入表之前，它可以根据标题计算哈希值，并使用该值来填充 `shard` 列。计算可能产生一个介于 1 和 200 之间非常均匀分布的数字，类似于随机策略所生成的数字。

简单的计算足够了，例如标题字符的 UTF-8 码位值的乘积，取模 200，加 1。复合分区键值是日期和计算结果的组合。

利用此策略，写入均匀分布在分区键值之间，从而均匀分布在物理分区之间。您可以对特定的行和日期轻松执行 `SELECT` 语句，因为您可以为特定的 `title` 值计算分区键值。

要读取指定日期的所有行，您仍然必须对每个（`2020-07-09, N`）键（其中，`N` 为 1-200）执行 `SELECT` 语句，然后应用程序必须合并所有结果。好处是避免出现单个“热门”分区键值，承担所有工作负载。

优化 Amazon Keyspaces 表的成本

本节介绍有关如何优化现有 Amazon Keyspaces 表的成本的最佳实践。您应该查看以下策略，看看哪种成本优化策略最适合您的需求，并以迭代方式处理它们。每个策略都概述了可能影响成本的因素、如何寻找优化成本的机会，以及有关如何实施这些最佳实践以帮助您节省成本的规范性指导。

主题

- 在表级别评估您的成本
- 评估表的容量模式
- 评估表的 Application Auto Scaling 设置
- 确定未使用的资源
- 评估您的表使用模式
- 评估预置容量大小是否合适

在表级别评估您的成本

中的Cost Explorer工具 AWS Management Console 允许您查看按类型细分的成本，例如读取、写入、存储和备份费用。您还可以查看按时段（例如月或日）汇总的这些成本。

Cost Explorer 的一个常见挑战是，您无法轻松地只查看一张特定表的成本，因为 Cost Explorer 不允许您按特定表的成本进行筛选或分组。您可以在 Amazon Keyspac es 控制台的 “监控” 选项卡上查看每个

表的可计费表大小（字节）指标。如果您需要更多与每个表的成本相关的信息，本节将向您介绍如何在 Cost Explorer 中使用标签来执行单个表的成本分析。

主题

- 如何查看单个 Amazon Keyspaces 表的成本
- Cost Explorer 的默认视图
- 如何在 Cost Explorer 中使用和应用表标签

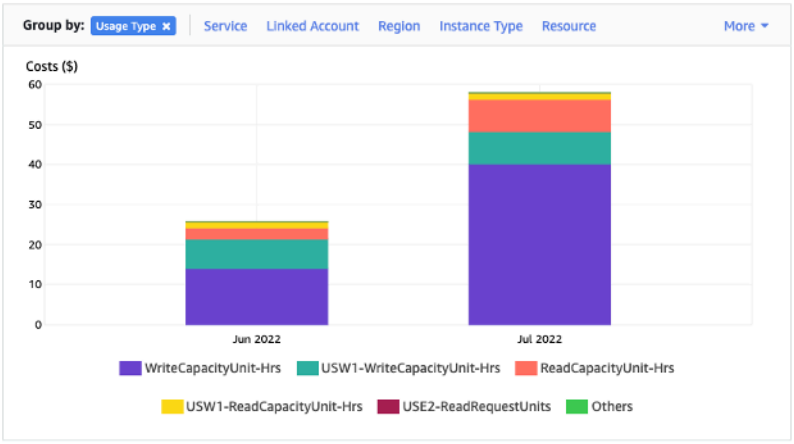
如何查看单个 Amazon Keyspaces 表的成本

您可以在控制台中查看有关 Amazon Keyspaces 表的基本信息，包括主键架构、计费表大小和与容量相关的指标。您可以使用表的大小来计算表的每月存储成本。例如，美国东部（弗吉尼亚北部）AWS 区域每 GB 0.25 美元。

如果表使用预置容量模式，则还会返回当前读取容量单位 (RCU) 和写入容量单位 (WCU) 设置。您可以使用此信息来计算表的当前读取和写入成本。请注意，这些成本可能会发生变化，特别是如果您已将表配置为 Amazon Keyspaces 自动缩放。

Cost Explorer 的默认视图

Cost Explorer 的默认视图提供显示已用资源（例如吞吐量和存储）的成本的图表。您可以选择按时段对这些成本进行分组，例如按月或按日列出总值。存储、读取、写入和其他类别的成本也可以进行细分和比较。



如何在 Cost Explorer 中使用和应用表标签

默认情况下，Cost Explorer 不会提供任何一个特定表的成本汇总，因为它会将多个表的成本合并为一个总额。不过，您可以使用 AWS 资源标记，通过元数据标签来标识各个表。标签是可用于多种用途的

键值对，例如标识属于某个项目或部门的所有资源。有关更多信息，请参阅 the section called “使用标签”。

在这个例子中，我们使用一个名为的表MyTable。

1. 使用密钥为 t able_name 和值设置标签。MyTable
2. 在 Cost Explorer 中激活标签，然后筛选标签值，以便更清楚地了解每个表的成本。

> **Note**
>
> 标签可能需要一两天的时间才会开始显示在 Cost Explorer 中

您可以在控制台中自己设置元数据标签，也可以使用 CQL、或 SDK 以编程方式设置元数据标签。AWS CLI AWS 考虑一下，在您组织的新表创建流程中要求设置 table_name 标签。有关更多信息，请参阅 the section called “Amazon Keyspaces 成本分配报告”。

评估表的容量模式

本节概述如何为 Amazon Keyspaces 表选择合适的容量模式。每种模式都经过优化，以满足不同工作负载对吞吐量变化的响应能力以及使用量计费方式的需求。在制定决策时，您必须平衡这些因素。

主题

- 有哪些表容量模式可用
- 何时选择按需容量模式
- 何时选择预置容量模式
- 选择表容量模式时需要考虑的其他因素

有哪些表容量模式可用

创建 Amazon Keyspaces 表时，您必须选择按需容量模式或预置容量模式。有关更多信息，请参阅 the section called “读/写容量模式”。

按需容量模式

按需容量模式用于消除规划或预置 Amazon Keyspaces 表容量的需求。在此模式下，您的表会即时适应请求数，无需扩展或缩减任何资源（最高为表之前峰值吞吐量的两倍）。

按需容量表通过计算表的实际请求数进行计费，因此您只需按实际使用量付费，而不是按预置容量付费。

预置容量模式

预置容量模式是一种更为传统的模式，在这种模式下，您可以直接定义可供表用于处理请求的容量，也可以借助 Application Auto Scaling 来定义。由于在任何给定时间表都预置了具体容量，因此基于预置容量而不是请求数量进行计费。而且，如果请求数超出分配的容量，就会导致表拒绝请求，降低应用程序用户的体验。

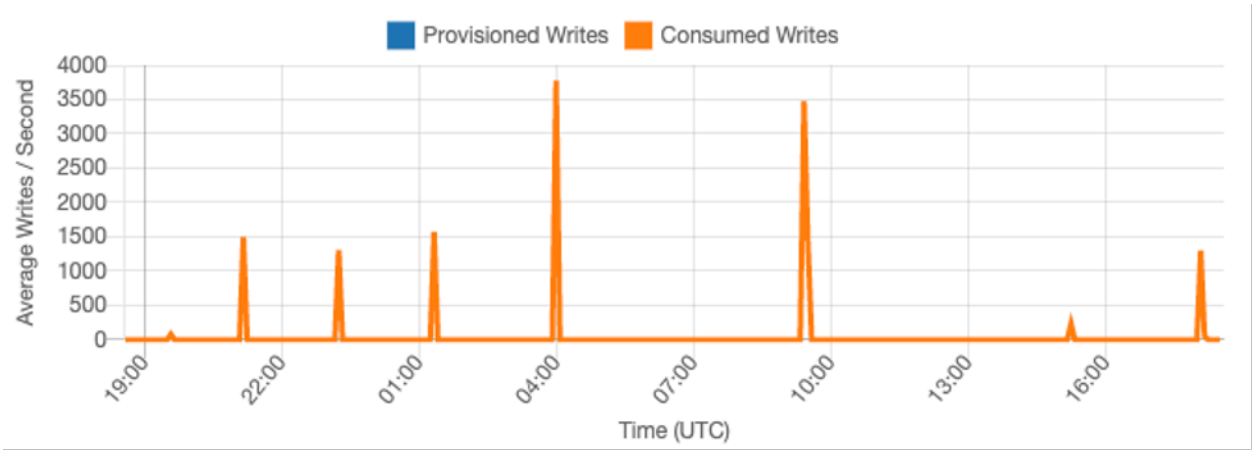
预置容量模式要求既不过度预置表容量，也不让表容量预置不足，以降低吞出容量不足错误出现的可能性和优化成本。

何时选择按需容量模式

在优化成本时，如果您有类似于下图的不可预测工作负载，则按需模式是您的最佳选择。

以下因素会导致此类工作负载：

- 不可预测的请求时机（导致流量峰值）
- 请求数量变动（由批量工作负载导致）
- 某个时段内降至零或低于峰值的 18%（由开发或测试环境导致）



对于具有上述特征的工作负载，使用 Application Auto Scaling 来保持足够的容量以供表应对流量激增可能会导致不良后果。要么表过度预置，成本超出必要范围，要么表预置不足，请求导致了不必要的吞出容量不足错误。在这种情况下，按需容量表是更好的选择。

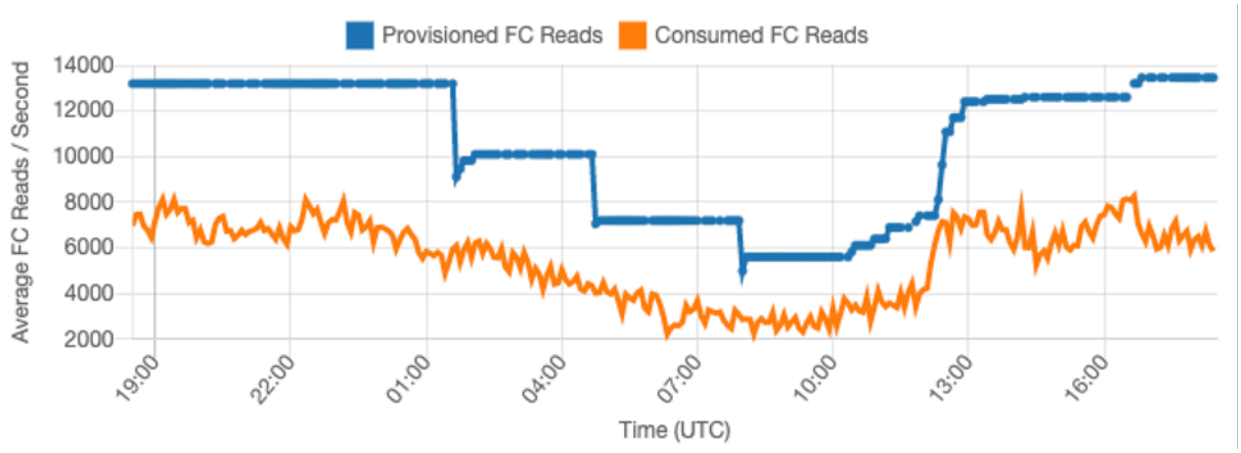
由于按需容量表按请求进行计费，您无需在表级别执行任何其他操作来优化成本。您应该定期评估按需容量表，以验证工作负载是否仍然具备上述特征。如果工作负载已经稳定，则可考虑改为预置模式以保持成本优化。

何时选择预置容量模式

适合预置容量模式的工作负载应该是具有可预测使用模式的工作负载，如下图所示。

以下因素有助于实现可预测的工作负载：

- 给定时段或一天内的可预测/周期性流量
- 短期内流量爆发有限



由于给定时段或一天内的流量更加稳定，您可以将预置容量设置为相对接近于表的实际使用量。预置容量表的成本优化过程归根到底是一个练习的过程，即在不增加表的 `ThrottledRequests` 事件的情况下，让预置容量（蓝线）尽可能靠近使用容量（橙线）。两条线之间的空间既是容量浪费，又是避免吞出容量不足错误导致用户体验欠佳的保障。

Amazon Keyspaces 为预置容量表提供 Application Auto Scaling，该功能会自动为您实现均衡。您可以跟踪全天使用的容量，并根据一些变量来配置表的预置容量。

最小容量单位

您可以设置表的最小容量来限制吞出容量不足错误的发生，但这不会降低表的成本。如果您的表有一段时间使用量较低，然后突然出现高使用量，那么设置最小值可以防止 Application Auto Scaling 将表容量设置得过低。

最大容量单位

您可以设置表的最大容量，以限制将表扩大到超出预期值。对于无需大规模负载测试的开发或测试表，可考虑应用最大值。您可以为任意表设置最大值，但在生产环境中使用该表时，请务必定期根据表的使用基准评估此设置，以防止意外出现吞出容量不足错误。

目标利用率

对于预置容量表，设置表的目标利用率是优化其成本的主要方法。将此值设置为较低的百分比会提高表的过度预置程度，进而增加成本，但可以降低出现吞出容量不足错误的风险。将此值设置为较高的百分比可以降低表的过度预置程度，但会提高出现吞出容量不足错误的风险。

选择表容量模式时需要考虑的其他因素

在两种容量模式之间做出选择时，还需要考虑另外一些因素。

在两种表模式之间做出选择时，请考虑这种额外折扣对表的成本有多大的影响。在许多情况下，在过度预置的预置容量表上，即使是相对不可预测的工作负载，采用预留容量也可能更经济。

提高工作负载的可预测性

在某些情况下，工作负载可能会同时具有可预测和不可预测的模式。虽然按需容量表可以轻松支持这种模式，但如果能够改善工作负载中不可预测的模式，则可能会降低成本。

造成这种模式的最常见原因之一是批量导入。这种类型的流量通常会超过表的基准容量，以至于在运行表时出现吞出容量不足错误。要在预置容量表上确保此类工作负载的运行，请考虑以下选项：

- 如果批处理按照预定时间进行，则可以在运行之前，计划增加应用程序自动扩缩容量。
- 如果批处理随机进行，则可以考虑尝试延长运行所需的时间，而不是尽快执行。
- 为导入操作添加一个加速期，在此期间，开始时导入的速度较慢，但会在几分钟内缓慢加快，直到 Application Auto Scaling 有机会开始调整表容量。

评估表的 Application Auto Scaling 设置

本节概述如何评估 Amazon Keyspaces 表上的 Application Auto Scaling 设置。Amazon Keyspaces Application Auto Scaling 是一项功能，用于根据您的应用程序流量和目标利用率指标来管理表吞吐量。这可以确保您的表具有应用程序模式所需的容量。

Application Auto Scaling 服务将监控您当前的表利用率，并与目标利用率值 `TargetValue` 作比较。当需要增加或减少分配的容量时，它会通知您。

主题

了解 Application Auto Scaling 设置

为目标利用率、初始步骤和最终值定义正确的值是一项需要运营团队参与的活动。这让您可以根据应用程序的历史使用量来适当地定义值，以便触发 Application Auto Scaling 策略。利用率目标是总容量的百分比值，需要在一段时间内达到后，才会应用 Application Auto Scaling 规则。

当您设置高利用率目标（大约 90%）时，意味着流量在一段时间内高于 90% 后才会激活 Application Auto Scaling。除非您的应用程序非常稳定且不会收到高峰流量，否则不应使用高利用率目标。

当您设置非常低的利用率目标（低于 50%）时，意味着您的应用程序需要达到预置容量的 50% 后才会触发 Application Auto Scaling 策略。除非您的应用程序流量以非常激进的速度增长，否则这通常会造成未用的容量和浪费的资源。

如何识别具有低目标利用率 (<=50%) 的表

您可以使用 AWS CLI 或 AWS Management Console 在 Amazon Keyspaces 资源`TargetValues`中监控和识别您的应用程序 Auto Scaling 策略：

AWS CLI

1. 通过运行以下命令返回整个资源列表：

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    cassandra
   ```

   此命令将返回发给任何 Amazon Keyspaces 资源的 Application Auto Scaling 策略的完整列表。如果您只想检索来自特定表的资源，则可以添加 `–resource-id parameter`。例如：

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    cassandra --resource-id "keyspace/keyspace-name/table/table-name”
   ```

2. 通过运行以下命令仅返回特定表的自动扩缩策略

```
aws application-autoscaling describe-scaling-policies --service-namespace
 cassandra --resource-id "keyspace/keyspace-name/table/table-name”
```

Application Auto Scaling 策略的值在下面突出显示。您需要确保目标值大于 50%，以避免过度预置。您应该得到类似如下的结果：

```
{
    "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. 登录 AWS Management Console 并导航到 “入门” 中的 CloudWatch 服务页面 AWS Management Console。如有必 AWS 区域 要，请选择相应的。
2. 在左侧导航栏上，选择表。在表页面上，选择表的名称。
3. 在表详细信息页面的容量选项卡上，查看表的 Application Auto Scaling 设置。

如果您的目标利用率值小于或等于 50%，则应浏览您的表利用率指标，看看这些指标是配置不足还是过度配置。

如何处理具有季节性差异的工作负载

考虑以下场景：您的应用程序大部分时间都在最低平均值下运行，但是利用率目标较低，所以您的应用程序可以对一天中特定时间发生的事件做出快速反应，并且您有足够的容量来避免节流。当您的应用程序在正常办公时间（上午 9 点到下午 5 点）非常繁忙，但在下班后处于基本工作水平时，这种场景很常见。因为一些用户会在上午 9 点前开始连接，所以应用程序使用这个低阈值来实现快速提高，以便在高峰时段达到所需容量。

此场景可能是这样的：

- 下午 5 点到上午 9 点之间，`ConsumedWriteCapacityUnits` 单位保持在 90 到 100 之间
- 用户在上午 9 点之前开始连接到应用程序，容量单位显著增加（您看到的最大值为 1500WCU）
- 平均下来，您的应用程序在工作时间内的使用量在 800 到 1200 之间变化

如果前面的场景适合您的应用程序，可考虑使用计划 Application Auto Scaling，在这种情况下，您的表仍可以配置 Application Auto Scaling 规则，但目标利用率不那么激进，只是在您需要的特定间隔预置了额外容量。

您可以使用执行以下步骤 AWS CLI 来创建根据一天中的时间和星期几执行的定时自动缩放规则。

1. 使用 Application Auto Scaling将您的 Amazon Keyspaces 表注册为可扩展目标。可扩展目标是 Application Auto Scaling 可以扩大或缩小的资源。

```
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. 根据您的要求设置预定操作。

   在此场景中，您需要两条规则：一条规则用来扩展，一条规则用来缩减。用来扩展预定操作的第一条规则如下面的示例所示。

```
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"
```

   用来缩减预定操作的第二条规则如下面的示例所示。

```
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"
```

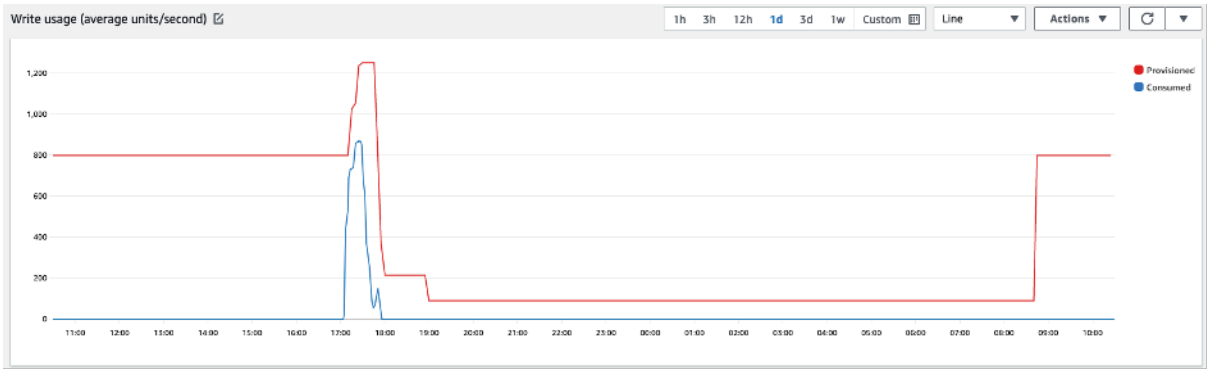
3. 运行以下命令来验证两条规则是否已激活：

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

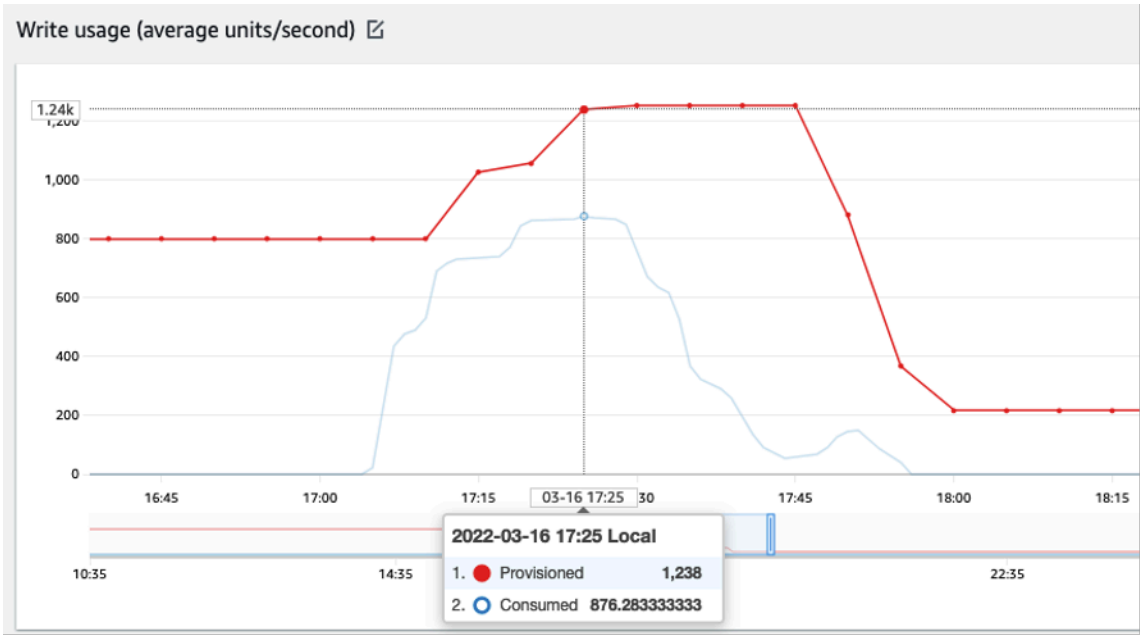
   应得到类似如下的结果：

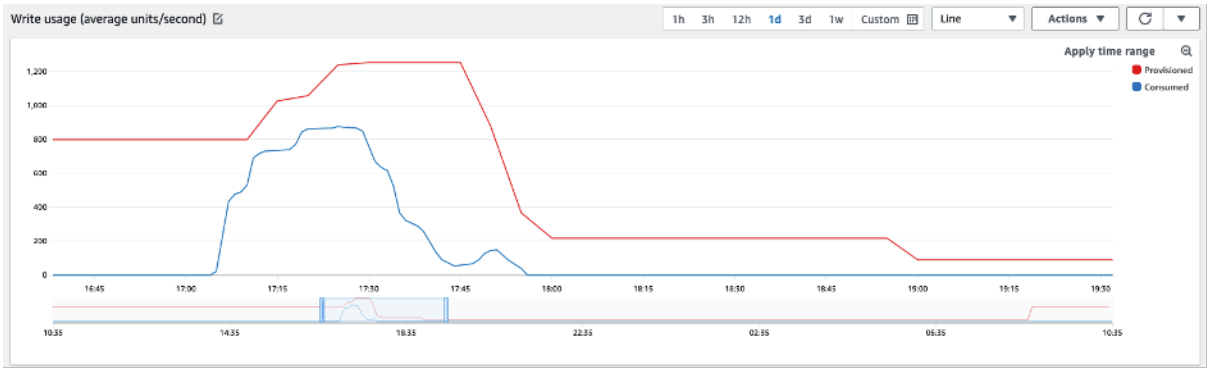
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/keyspaces/
table/table-name:scheduledActionName/my-5-8-scheduled-down-action",
     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

下图显示了一个始终保持 70% 目标利用率的示例工作负载。请注意观察，如何做到自动扩缩规则仍在应用，而吞吐量不会降低。



放大图片，我们可以看到，应用程序中有一个高峰，它触发了 70% 自动扩缩阈值，迫使自动扩缩启动，来为表提供所需的额外容量。计划的 auto Scaling 操作将影响最大值和最小值，您有责任对其进行设置。





如何处理具有未知模式的尖峰工作负载

在这种场景中，应用程序使用非常低的利用率目标，因为您尚不清楚应用程序的模式，需要确保工作负载不遇到吞出容量不足错误。

可考虑改用按需容量模式。按需模式表非常适合您不知道流量模式的尖峰工作负载。在按需容量模式下，您按请求为应用程序在表上执行的数据读取和写入付费。您无需指定应用程序预计将执行多少读取和写入吞吐量，因为 Amazon Keyspaces 会根据工作负载的增减来即时做出调整。

如何处理具有关联应用程序的工作负载

在这种场景中，应用程序依赖其他系统，例如在批处理场景中，根据应用程序逻辑中的事件，您会遇到非常大的流量尖峰。

可考虑开发自定义 Application Auto Scaling 逻辑，以便对那些您可以根据自己的具体需求增加表容量和 `TargetValues` 的事件作出反应。您可以从 λ Amazon EventBridge 和 Step Functi AWS ons 等服务组合中受益并使用这些服务来满足您的特定应用需求。

确定未使用的资源

本部分概述如何定期评估未使用资源。随着应用程序需求的演变，您应确保没有未使用的资源，并且不会导致不必要的 Amazon Keyspaces 成本。下述程序使用 Amazon CloudWatch 指标来识别未使用的资源并采取措施降低成本。

您可以使用监控 Amazon Keyspaces CloudWatch，它收集来自亚马逊密钥空间的原始数据并将其处理为可读的、近乎实时的指标。这些统计数据会保留一段时间，这样您便可以访问历史信息，更好地了解使用情况。默认情况下，Amazon Keyspaces 指标数据会自动发送到。 CloudWatch 有关更多信息，请参阅 Amazon CloudWatch 是什么？ 以及《亚马逊 CloudWatch 用户指南》中的指标保留率。

主题

- 如何确定未使用的资源
- 确定未使用的表资源
- 清理未使用的表资源
- 清理未使用的 point-in-time 恢复 (PITR) 备份

如何确定未使用的资源

要识别未使用的表，您可以在 30 天内查看以下 CloudWatch 指标，以了解特定表上是否有任何活跃的读取或写入：

ConsumedReadCapacityUnits

在指定时间段内使用的读取容量单位数，这样便可以跟踪您在已占用容量中使用的容量。您可以检索表已使用的总读取容量。

ConsumedWriteCapacityUnits

在指定时间段内使用的写入容量单位数，这样便可以跟踪您在已占用容量中使用的容量。您可以检索表使用的总写入容量。

确定未使用的表资源

Amazon CloudWatch 是一项监控和可观察性服务，它提供 Amazon Keyspaces 表指标，您可以用来识别未使用的资源。 CloudWatch 可以通过， AWS Management Console 也可以通过查看指标 AWS Command Line Interface。

AWS Command Line Interface

要通过查看您的表格指标 AWS Command Line Interface，您可以使用以下命令。

1. 首先，评估您的表的读取数：

> **Note**
>
> 如果表名称在账户中不是唯一的，则还必须指定键空间的名称。

```
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>
```

为了避免错误地将表标识为未使用，请评估较长时段内的指标。选择合适的开始时间和结束时间范围（如 30 天）以及合适的时间段（如 86400）。

在返回的数据中，任何大于 0 的 Sum（合计）都表示您所评估的表在该时间段内收到了读取流量。

以下结果显示表在评估时段内收到了读取流量：

```
        {
```

```
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

以下结果显示表在评估时段内未收到读取流量：

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. 接下来，评估表的写入数量：

```
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>
```

为了避免错误地将表标识为未使用，您需要评估较长时段内的指标。选择合适的开始时间和结束时间范围（例如 30 天）以及合适的时间段，例如 86400。

在返回的数据中，任何大于 0 的 Sum（合计）都表示您所评估的表在该时间段内收到了读取流量。

以下结果显示表在评估时段内收到了写入流量：

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
```

```
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

以下结果显示表在评估时段内未收到写入流量：

```
        {
            "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

您可以按照以下步骤，通过 AWS Management Console评估资源利用率。

1. 登录 AWS Management Console 并导航到 CloudWatch 服务页面，网址为 https://console.aws.amazon.com/cloudwatch/。如有必要 AWS 区域 ，请在控制台右上角选择相应的选项。
2. 在左侧导航栏中，找到指标部分，然后选择所有指标。
3. 以上操作将打开一个控制面板，其中包含两个面板。在顶部面板中，您可以看到当前绘成图表的指标。在底部，您可以选择用于绘制图表的指标。在底部面板中选择“Amazon Keyspaces”。
4. 在 Amazon Keyspaces 指标选择面板中，选择表指标类别以显示当前区域中表的指标。
5. 向下滚动菜单，找到您的表名称，然后选择表的指标 `ConsumedReadCapacityUnits` 和 `ConsumedWriteCapacityUnits`。
6. 选择绘成图表的指标 (2) 选项卡，然后将统计数据列调整为总计。
7. 为了避免错误地将表标识为未使用，请评估较长时段内的表指标。在图表面板顶部选择相应的时间范围（如 1 个月）来评估表。选择自定义，在下拉菜单中选择 1 个月，然后选择应用。
8. 评估您的表的绘成图表的指标，以确定是否使用了该表。大于 0 的指标表示在评估时间段内使用了表。读取和写入均为 0 的空白图表表示表未使用。

清理未使用的表资源

如果您已确定未使用的表资源，则可以通过以下方式降低其持续成本。

> **Note**
>
> 如果您已确定未使用的表，但仍希望将其保持可用状态，以防将来需要访问该表，请考虑将其切换到按需模式。否则，您可以考虑删除该表。

容量模式

Amazon Keyspaces 对读取、写入和存储 Amazon Keyspaces 表中的数据收费。

Amazon Keyspaces 具有按需和预置这两种容量模式，它们对表上的读取和写入处理采用特定的计费方式。读/写容量模式控制对读写吞吐量收费的方式以及管理容量的方式。

对于按需模式表，您无需指定预期应用程序执行的读取和写入吞吐量。Amazon Keyspaces 按照读取请求单位和写入请求单位对应用程序在表上执行的读取和写入操作收费。如果表上没有活动，则您无需为吞吐量付费，但仍会产生存储费用。

删除表

如果您发现了一个未使用的表并想将其删除，可考虑先备份或导出数据。

完成的备份 AWS Backup 可以利用冷存储分层，从而进一步降低成本。有关如何使用生命周期将备份移动到冷存储的信息，请参阅管理备份计划文档。

备份表后，您可以选择通过 AWS Management Console 或通过 AWS Command Line Interface将其删除。

清理未使用的 point-in-time 恢复 (PITR) 备份

Amazon Keyspaces 提供 P oint-in-time 恢复，可提供 35 天的连续备份，以帮助您防止意外写入或删除。PITR 备份会产生相关成本。

请参阅文档 时间点恢复，确定您的表是否启用了可能不再需要的备份。

评估您的表使用模式

本节概述如何评估您是否高效地使用 Amazon Keyspaces 表。有些使用模式对于 Amazon Keyspaces 来说不是最佳的，在性能和成本方面还有优化的空间。

主题

- 执行更少的强一致性读取操作
- 启用生存时间（TTL）

执行更少的强一致性读取操作

Amazon Keyspaces 允许您根据每个请求来配置读取一致性。默认情况下，读取请求为最终一致性的。最终一致性读取的收费标准是 4KB 及以下数据量为 0.5 个 RCU。

大多数分布式工作负载具有灵活性，能够容许最终一致性。但是，也有一些访问模式需要强一致性读取。强一致性读取的收费标准是 4KB 及以下数据量为 1 个 RCU，读取成本基本上翻了一倍。Amazon Keyspaces 让您能够灵活地在同一个表上使用这两种一致性模式。

您可以评估一下您的工作负载和应用程序代码，确认是否只在需要时才使用强一致性读取。

启用生存时间（TTL）

存活时间 (TTL) 通过自动使表中的数据过期，来帮助您简化应用程序逻辑和优化存储价格。系统会根据您设置的“存活时间”值自动从表中删除不再需要的数据。

评估预置容量大小是否合适

本节概述如何评估您的 Amazon Keyspaces 表预置容量大小是否合适。随着工作负载的演变，您应该相应地修改操作程序，尤其是在您的 Amazon Keyspaces 表以预置模式配置时，因为它们可能存在过度预置或预置不足的风险。

本节中所述的过程需要从支持您的生产应用程序的 Amazon Keyspaces 表捕获的统计信息。要了解您的应用程序行为，应该定义一个足够长的周期，以捕获应用程序的数据季节性特点。例如，如果您的应用程序表现出周模式，则不妨使用三周的周期来分析应用程序吞吐量需求。

如果您不知道从哪里开始，可根据至少一个月的使用数据来进行以下计算。

在评估容量时，您可以为 Amazon Keyspaces 表单独配置读取容量单位 (RCU) 和写入容量单位 (WCU)。

主题

- 如何从 Amazon Keyspaces 表中检索使用指标

- 如何识别预置不足的 Amazon Keyspaces 表
- 如何识别过度预置的 Amazon Keyspaces 表

如何从 Amazon Keyspaces 表中检索使用指标

要评估表容量，请监控以下 CloudWatch 指标并选择相应的维度来检索表格信息：

读取容量单位	写入容量单位
ConsumedReadCapacityUnits	ConsumedWriteCapacityUnits
ProvisionedReadCapacityUnits	ProvisionedWriteCapacityUnits
ReadThrottleEvents	WriteThrottleEvents

您可以通过 AWS CLI 或来执行此操作 AWS Management Console。

AWS CLI

在检索表格消耗指标之前，您需要先使用 CloudWatch API 捕获一些历史数据点。

首先创建两个文件：write-calc.json 和 read-calc.json。这些文件代表表的计算。您需要更新一些字段（如下表所示）以匹配您的环境。

> **Note**
>
> 如果表名称在账户中不是唯一的，则还必须指定键空间的名称。

字段名称	定义	示例
<table-name>	您要分析的表的名称	SampleTable
<period>	您要用来评估利用率目标的周期（以秒为单位）	对于 1 小时的周期，应指定：3600
<start-time>	评估间隔的开始时间，以 ISO8601 格式指定	2022-02-21T23:00:00

字段名称	定义	示例
<end-time>	评估间隔的结束时间，以 ISO8601 格式指定	2022-02-22T06:00:00

写入计算文件将检索指定日期范围内的时间段中预置和使用的 WCU 数量。它还将生成可用于分析的利用率百分比。write-calc.json 文件的完整内容应类似于以下示例。

```
{
  "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
}
```

读取计算文件使用类似的指标。此文件将检索指定日期范围内的时间段中预置和使用的 RCU 数量。read-calc.json 文件的内容应类似于以下示例。

```
{
  "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
}
```

创建文件后，即可开始检索利用率数据。

1. 要检索写入利用率数据，请发出以下命令：

```
aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
```

2. 要检索读取利用率数据，请发出以下命令：

```
aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
```

这两个查询的结果都是一系列 JSON 格式的数据点，可用于分析。您的结果取决于您指定的数据点数量、周期和您自己的特定工作负载数据。它可能类似于以下示例。

```
{
    "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**
>
> 如果您指定了短周期和长时间范围，则可能需要修改 `MaxDatapoints` 值，该值在脚本中默认设置为 24。这表示每小时一个数据点，每天 24 个数据点。

AWS Management Console

1. 登录 AWS Management Console 并导航到 “入门” 中的 CloudWatch 服务页面 AWS Management Console。如有必 AWS 区域 要，请选择相应的。
2. 在左侧导航栏中，找到“指标”部分，然后选择所有指标。
3. 这将打开一个控制面板，其中包含两个面板。顶部面板显示图表，底部面板显示要绘成图表的指标。选择“Amazon Keyspaces”面板。
4. 从子面板中选择表指标类别。这会向您显示当前表格 AWS 区域。
5. 识别您的表名，方法是向下滚动菜单并选择写入操作指标：`ConsumedWriteCapacityUnits` 和 `ProvisionedWriteCapacityUnits`。

   > **Note**
   >
   > 本例中使用了写入操作指标，但您也可以使用这些步骤绘制读取操作指标图。

6. 选择绘成图表的指标(2) 选项卡来修改公式。默认情况下，为图表 CloudWatch 选择统计函数“平均值”。
7. 选中两个图表化指标（左侧的复选框）后，选择菜单添加数学函数，然后选择常用，再选择 Percentage 函数。重复该过程两次。

   第一次选择 Percentage 函数。

   第二次选择 Percentage 函数。
8. 此时，底部菜单中应该有四个指标。我们来进行 `ConsumedWriteCapacityUnits` 计算。为了保持一致，您需要将名称与您在本 AWS CLI 节中使用的名称相匹配。单击 m1 ID 并将此值更改为 consumedWCU。
9. 将统计函数从 Average 改为 Sum。此操作将自动创建另一个名为 ANOMALY_DETECTION_BAND 的指标。在此过程中，您可以通过删除新生成的 ad1 指标上的复选框来忽略它。
10. 重复步骤 8，将 m2 ID 重命名为 provisionedWCU。保留统计函数设置为 Average。

11. 选择 Expression1 标签，并将值更新为 m1，将标签更新为 Consumed WCUs。

> **Note**
>
> 确保您只选择了 m1（左侧的复选框）和 provisionedWCU 以正确可视化数据。更新公式，方法是单击详细信息并将公式更改为 consumedWCU/PERIOD(consumedWCU)。这一步还可能生成另一个 ANOMALY_DETECTION_BAND 指标，但在此过程中，您可以忽略它。

12. 您现在应该有两个图表：一个表示表上预置的 WCU，另一个表示已使用的 WCU。
13. 通过选择 Expression2 图形 (e2) 来更新百分比公式。将标签和 ID 重命名为 utilizationPercentage。重命名公式，以匹配 100*(m1/provisionedWCU)。
14. 清除除 utilizationPercentage 之外的所有指标的复选框，以可视化您的利用率模式。默认间隔设置为 1 分钟，但您可以根据需要随意修改。

您获得的结果取决于您的工作负载中的实际数据。利用率超过 100% 的间隔容易导致吞出容量不足错误事件。Amazon Keyspaces 提供容量暴增，但一旦这些容量耗尽，任何超过 100% 的使用都会遇到吞出容量不足错误事件。

如何识别预置不足的 Amazon Keyspaces 表

对于大多数工作负载，如果一个表持续使用其预置容量超过 80%，该表将被视为预置不足。

容量暴增是 Amazon Keyspaces 的一项功能，它允许客户临时使用比最初预置更多的 RCU/WCU（超过为表定义的每秒预置吞吐量）。创建容量暴增的目的是应对因特殊事件或使用量高峰而突然增加的流量。这个容量暴增是有限的，要了解更多信息，请参阅the section called “容量爆增”。一旦未使用的 RCU 和 WCU 耗尽，您再试图使用比预置更多的容量时，可能就会遇到吞出容量不足错误事件。当您的应用程序流量接近 80% 的利用率时，您遇到吞出容量不足错误事件的风险就会大大增加。

80% 的利用率规则因数据的季节性和流量增长而异。考虑以下场景：

- 如果您的流量在过去 12 个月中一直稳定在大约 90% 的利用率，那么您的表容量正合适
- 如果您的应用程序流量在不到 3 个月内以每月 8% 的速度增长，那么您将达到 100% 利用率
- 如果您的应用程序流量在略长于 4 个月内以每月 5% 的速度增长，那么您仍然将达到 100% 利用率

以上查询的结果可以说明您的利用率。将它们作为指导，来进一步评估其他指标，以帮助您在需要时选择增加表容量（例如：每月或每周增长率）。与您的运营团队合作，为您的工作负载和表定义一个合适的百分比。

在有些特殊场景中，当您每天或每周进行数据分析时，会发现数据是倾斜的。例如，季节性应用程序在工作时间会出现使用量激增情况（但在工作时间之外则降至接近零），您可以使用计划 Application Auto Scaling 来指定一天中什么时间（以及一周中的星期几）增加预置容量，以及什么时间减少预置容量。即使您的季节性不那么明显，也可以利用 Amazon Keyspaces 表自动扩缩配置，而无需为了满足繁忙时段的需求而设定较高的容量。

如何识别过度预置的 Amazon Keyspaces 表

从上述脚本中获得的查询结果提供了执行某些初始分析所需的数据点。如果您的数据集在多个时间间隔内显示为利用率低于 20%，则您的表可能配置过度。要进一步确定是否需要减少 WCU 和 RCU 的数量，应重新查看间隔内的其他读数。

当您的表包含多个低使用量间隔时，您可以使用 Application Auto Scaling 策略，计划 Application Auto Scaling 或者为表配置基于利用率的默认 Application Auto Scaling 策略。

如果您的工作负载具有低利用率与高限制比（间隔内的最大值 (ThrottleEvents) /最小 (ThrottleEvents)），则当您的工作负载非常激增，流量在特定日期（或一天中的时间）显著增加，但其他方面却持续较低时，可能会发生这种情况。在这些场景中，使用计划 Application Auto Scaling 可能很有好处。

将 NoSQL Workbench 与 Amazon Keyspaces（Apache Cassandra 兼容）结合使用

NoSQL Workbench 是一款客户端应用程序，可帮助您更轻松地设计和可视化用于 Amazon Keyspaces 的非关系数据模型。NoSQL Workbench 客户端可用于 Windows、macOS 和 Linux。

设计数据模型并自动创建资源

NoSQL Workbench 为您提供了一个点击式界面，用于设计和创建 Amazon Keyspaces 数据模型。通过定义键空间、表和列，您可以轻松地从头开始创建新的数据模型。您还可以导入现有数据模型并进行修改（例如添加、编辑或删除列），以使数据模型适应新的应用程序。然后，NoSQL Workbench 允许您向 Amazon Keyspaces 或 Apache Cassandra 提交数据模型，并自动创建键空间和表。要了解如何构建数据模型，请参阅the section called “数据建模器”。

可视化数据模型

使用 NoSQL Workbench，您可以对数据模型进行可视化，以帮助确保数据模型能够支持应用程序的查询和访问模式。您还可以以各种格式保存和导出数据模型，用于协作、文档和演示。有关更多信息，请参阅the section called “数据可视化工具”。

主题

下载 NoSQL Workbench

请按照以下说明下载并安装 NoSQL Workbench。

下载并安装 NoSQL Workbench

1. 使用以下其中一个链接免费下载 NoSQL Workbench。

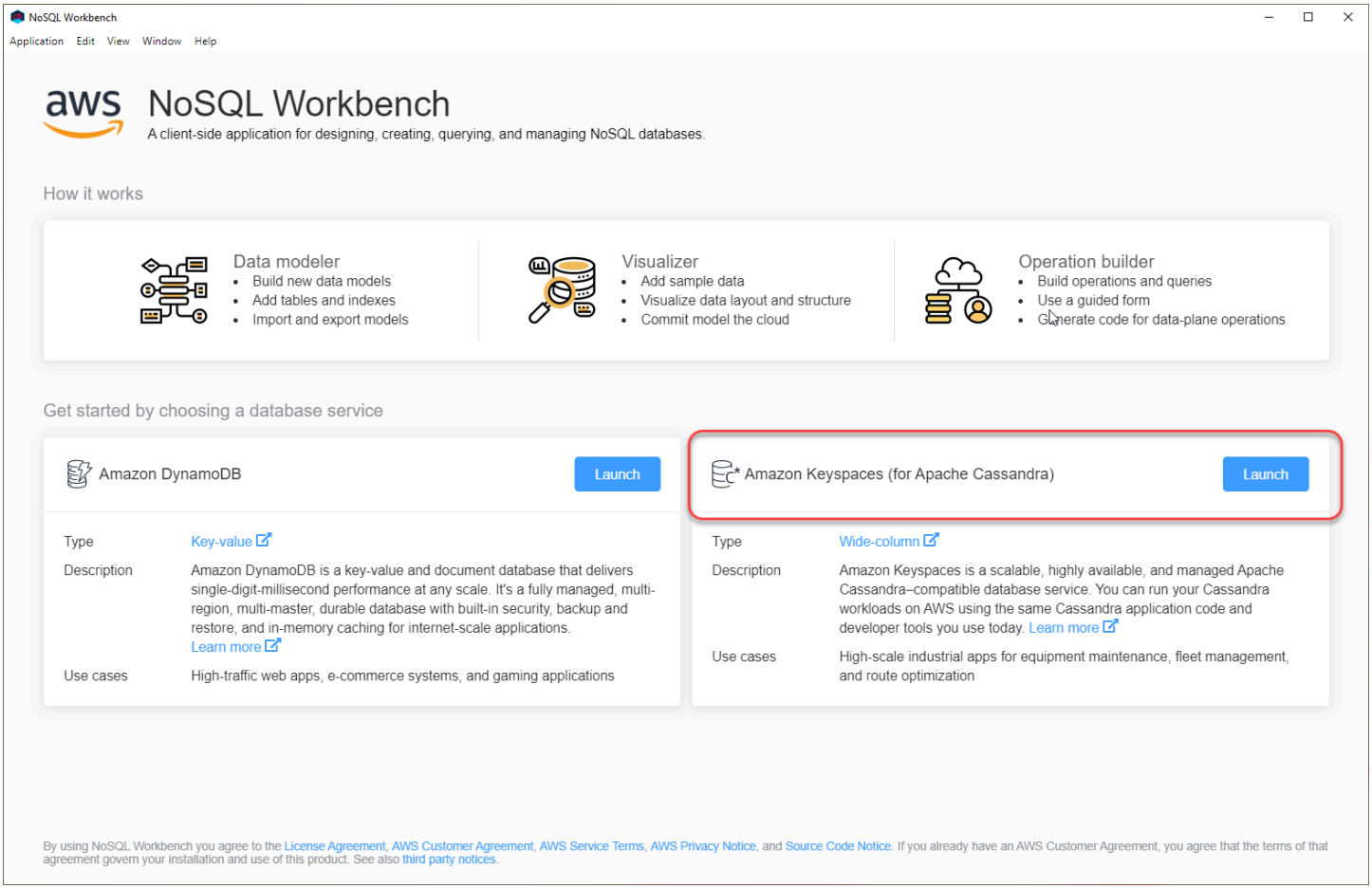
操作系统	下载链接
macOS	下载 macOS 版
Linux*	适用于 Linux 的下载
Windows	下载 Windows 版

* NoSQL Workbench 支持 Ubuntu 12.04、Fedora 21 和 Debian 8 或这些 Linux 发行版的任何较新版本。

2. 下载完成后，启动应用程序并按照屏幕上的说明完成安装。

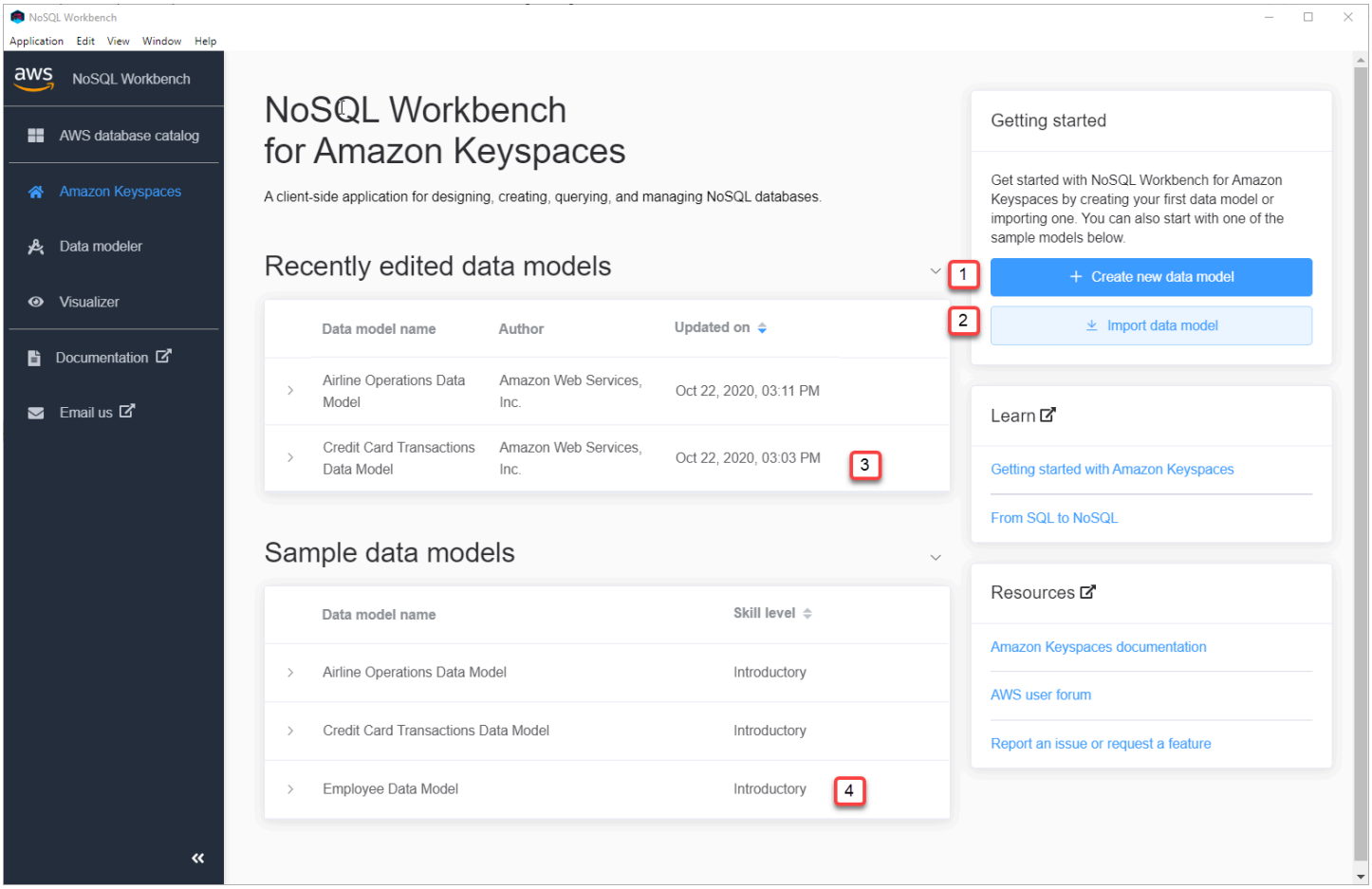
开始使用 NoSQL Workbench

要开始使用 NoSQL Workbench，请在 NoSQL Workbench 的数据库目录页面上选择 Amazon Keyspaces，然后选择启动。



这将打开适用于 Amazon Keyspaces 的 NoSQL Workbench 主页，您可以从中开始使用以下选项：

1. 创建新数据模型。
2. 以 JSON 格式导入现有的数据模型。
3. 打开最近编辑的数据模型。
4. 打开其中一个可用的示例模型。



每个选项都会打开 NoSQL Workbench 数据建模工具。要继续创建新的数据模型，请参阅 the section called “创建数据模型”。要编辑现有数据模型，请参阅 the section called “编辑数据模型”。

如何构建数据模型

您可以根据应用程序的数据访问模式，使用 NoSQL Workbench 数据建模工具设计新的数据模型。您可以使用数据建模工具设计新的数据模型，或导入和修改使用 NoSQL Workbench 创建的现有数据模型。数据建模工具包含一些示例数据模型，可帮助您快速开始数据建模。

主题

- 使用 NoSQL Workbench 构建新数据模型
- 使用 NoSQL Workbench 编辑现有数据模型

使用 NoSQL Workbench 构建新数据模型

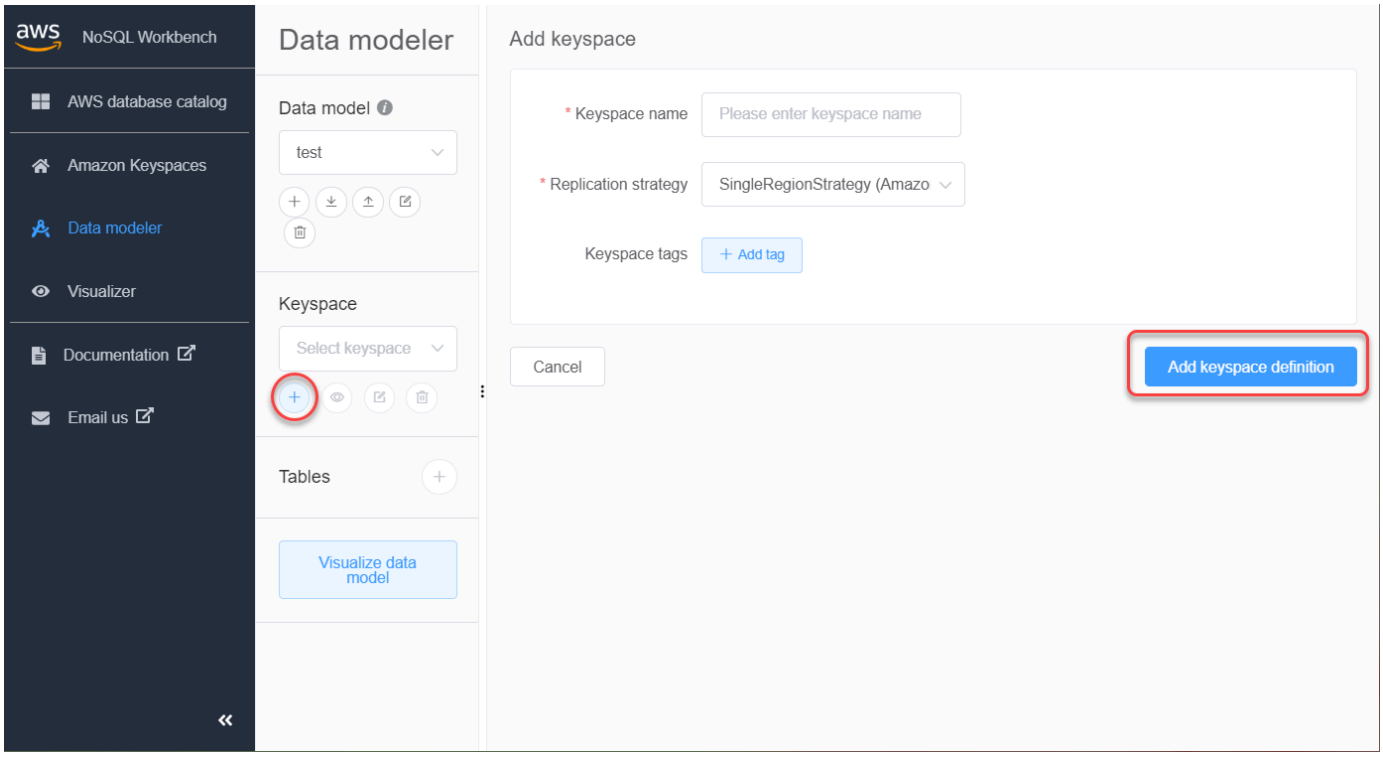
要为 Amazon Keyspaces 创建新的数据模型，您可以使用 NoSQL Workbench 数据建模器工具来创建键空间、表和列。按照以下步骤创建新的数据模型。

1. 要创建新的键空间，请选择键空间下的加号。

   在此步骤中，选择以下属性和设置。

   - 键空间名称：输入新键空间的名称。
   - 复制策略：为键空间选择复制策略。Amazon Keyspaces 使用 SingleRegionStrategy 在多个 AWS 可用区中自动复制三次数据。如果您打算将数据模型提交到 Apache Cassandra 集群，则可以选择 SimpleStrategy 或 NetworkTopologyStrategy。
   - 键空间标签：资源标签是可选的，并且允许您按各种方法对资源进行分类，例如，按用途、所有者、环境或其他标准。要了解有关 Amazon Keyspaces 资源标签的更多信息，请参阅 the section called “使用标签”。

2. 选择添加键空间定义以创建键空间。

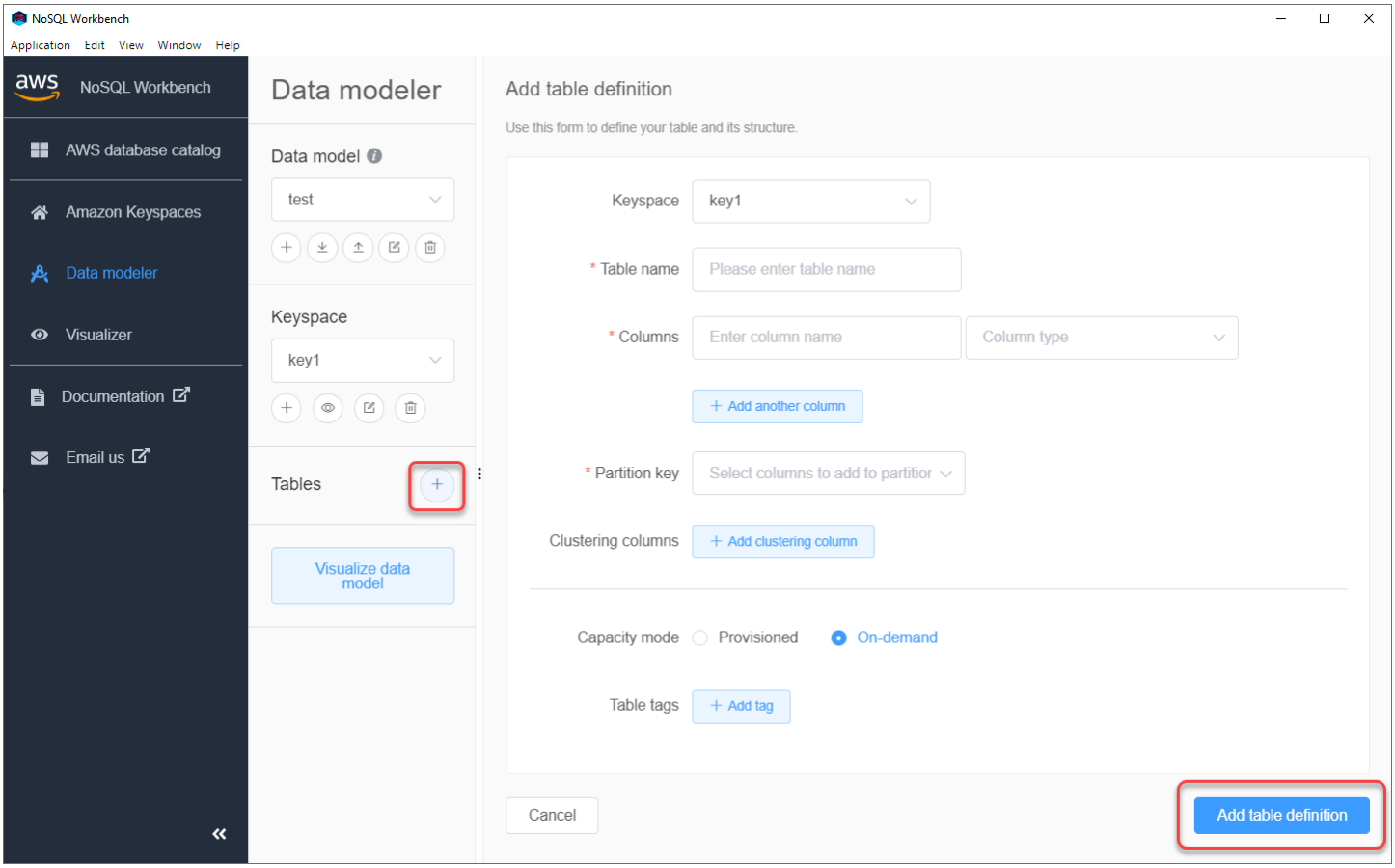


3. 要创建新表，请选择表旁边的加号。在此步骤中，您将定义以下属性和设置。

   - 表名称：新表的名称。
   - 列：添加列名并选择数据类型。对架构中的每一列重复这些步骤。

- 分区键：为分区键选择列。
- 集群列：选择集群列（可选）。
- 容量模式：选择表的读/写容量模式。您可以选择已预置或按需容量。要了解有关容量模式的更多信息，请参阅 the section called “读/写容量模式”。
- 表标签：资源标签是可选的，并且允许您按各种方法对资源进行分类，例如，按用途、所有者、环境或其他标准。要了解有关 Amazon Keyspaces 资源标签的更多信息，请参阅 the section called “使用标签”。

4. 选择添加表定义以创建新表。
5. 重复这些步骤以创建更多表。
6. 继续the section called “可视化数据模型”，可视化您创建的数据模型。



使用 NoSQL Workbench 编辑现有数据模型

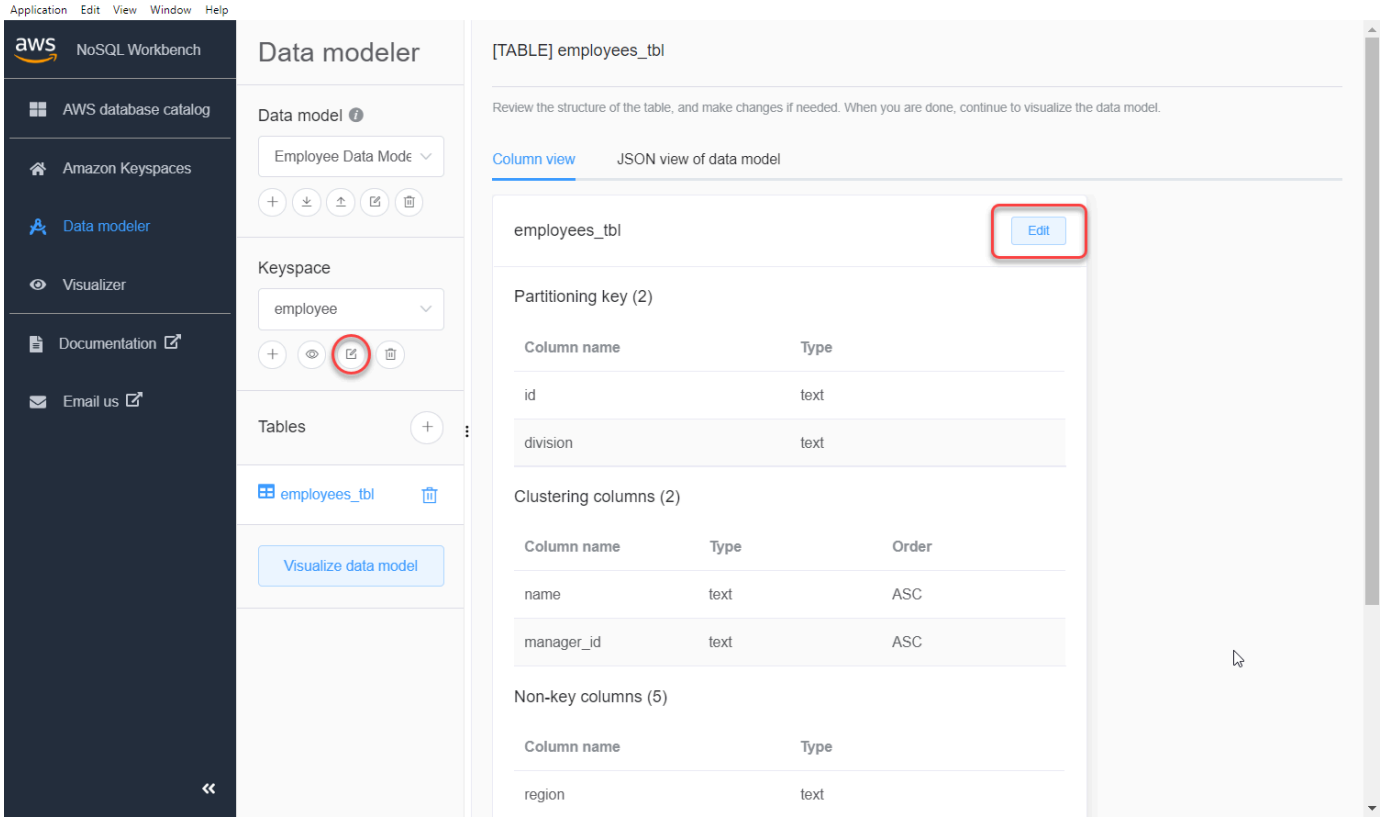
使用 NoSQL Workbench 数据建模工具，您可以编辑 Amazon Keyspaces 中的现有数据模型。这些模型可以是从文件导入的数据模型、提供的样本数据模型或您之前创建的数据模型。

1. 要编辑键空间，请在键空间下选择编辑符号。

   在此步骤中，您将编辑以下属性和设置。

   - 键空间名称：输入新键空间的名称。
   - 复制策略：为键空间选择复制策略。Amazon Keyspaces 使用 SingleRegionStrategy 在多个 AWS 可用区中自动复制三次数据。如果您打算将数据模型提交到 Apache Cassandra 集群，则可以选择 SimpleStrategy 或 NetworkTopologyStrategy。
   - 键空间标签：资源标签是可选的，并且允许您按各种方法对资源进行分类，例如，按用途、所有者、环境或其他标准。要了解有关 Amazon Keyspaces 资源标签的更多信息，请参阅 the section called “使用标签”。

2. 选择保存编辑内容以更新键空间。



3. 要编辑表，请选择表名称旁边的编辑。在此步骤中，您可以更新以下属性和设置。

   - 表名称：新表的名称。
   - 列：添加列名并选择数据类型。对架构中的每一列重复这些步骤。
   - 分区键：为分区键选择列。
   - 集群列：选择集群列（可选）。

- 容量模式：选择表的读/写容量模式。您可以选择已预置或按需容量。要了解有关容量模式的更多信息，请参阅 the section called “读/写容量模式”。
- 表标签：资源标签是可选的，并且允许您按各种方法对资源进行分类，例如，按用途、所有者、环境或其他标准。要了解有关 Amazon Keyspaces 资源标签的更多信息，请参阅 the section called “使用标签”。

4. 选择保存编辑内容以更新表。
5. 继续the section called “可视化数据模型”，可视化您更新的数据模型。

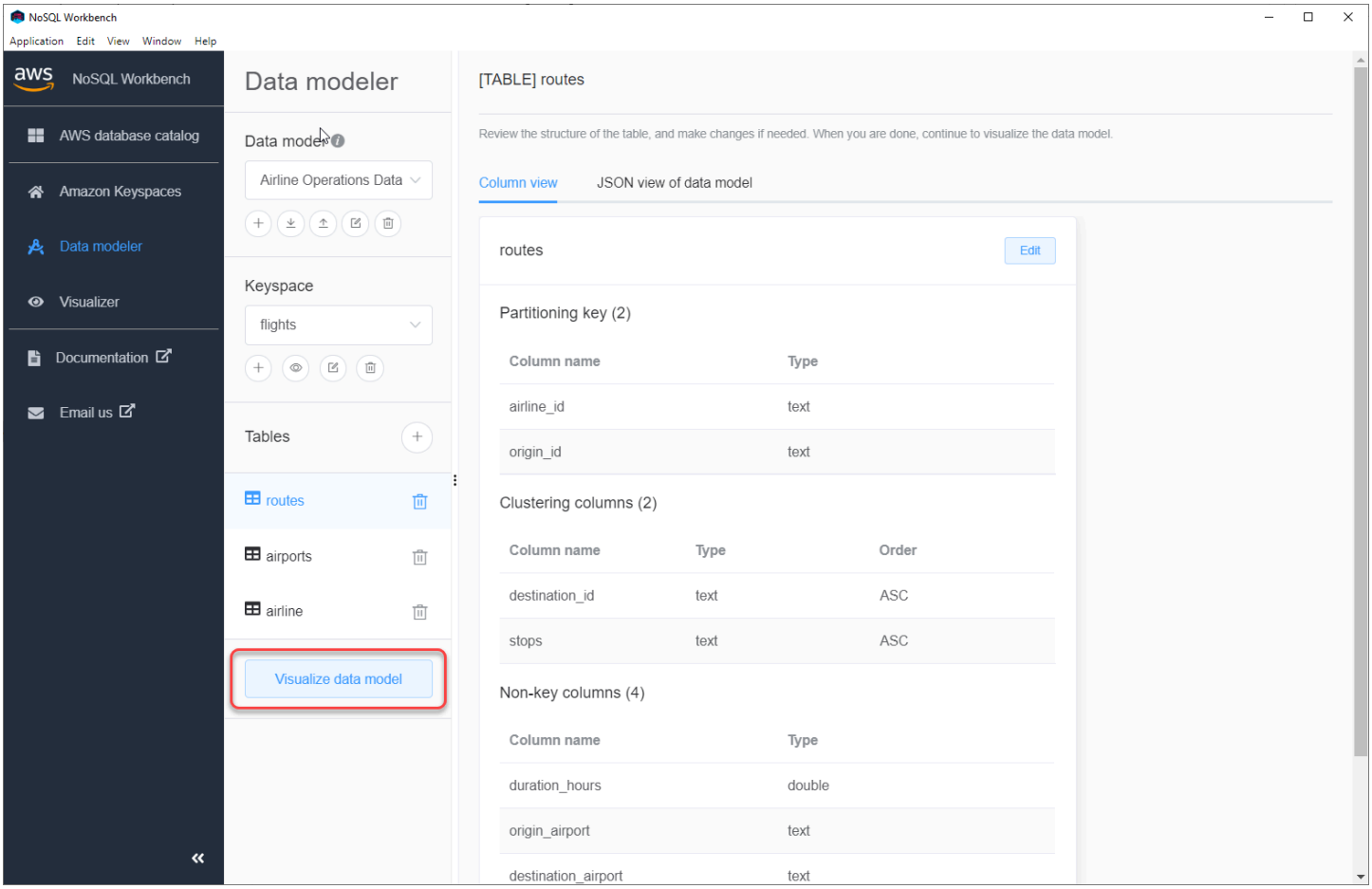
如何可视化数据模型

使用 NoSQL Workbench，您可以对数据模型进行可视化，以帮助确保数据模型能够支持应用程序的查询和访问模式。您还可以以各种格式保存和导出数据模型，用于协作、文档和演示。

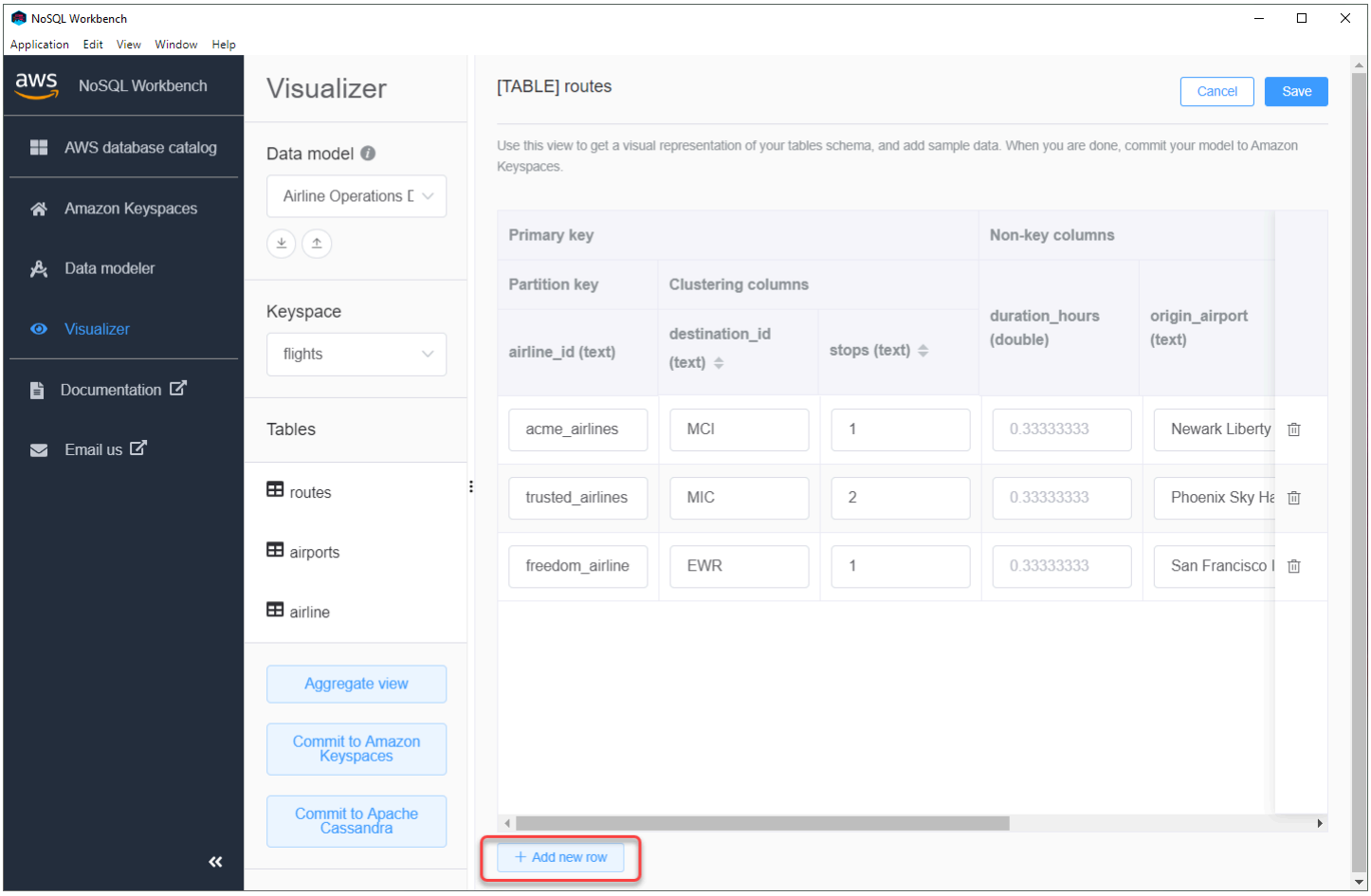
创建新数据模型或编辑现有数据模型后，可以对模型进行可视化。

使用 NoSQL Workbench 可视化数据模型

在数据建模工具中完成数据建模后，选择可视化数据模型。

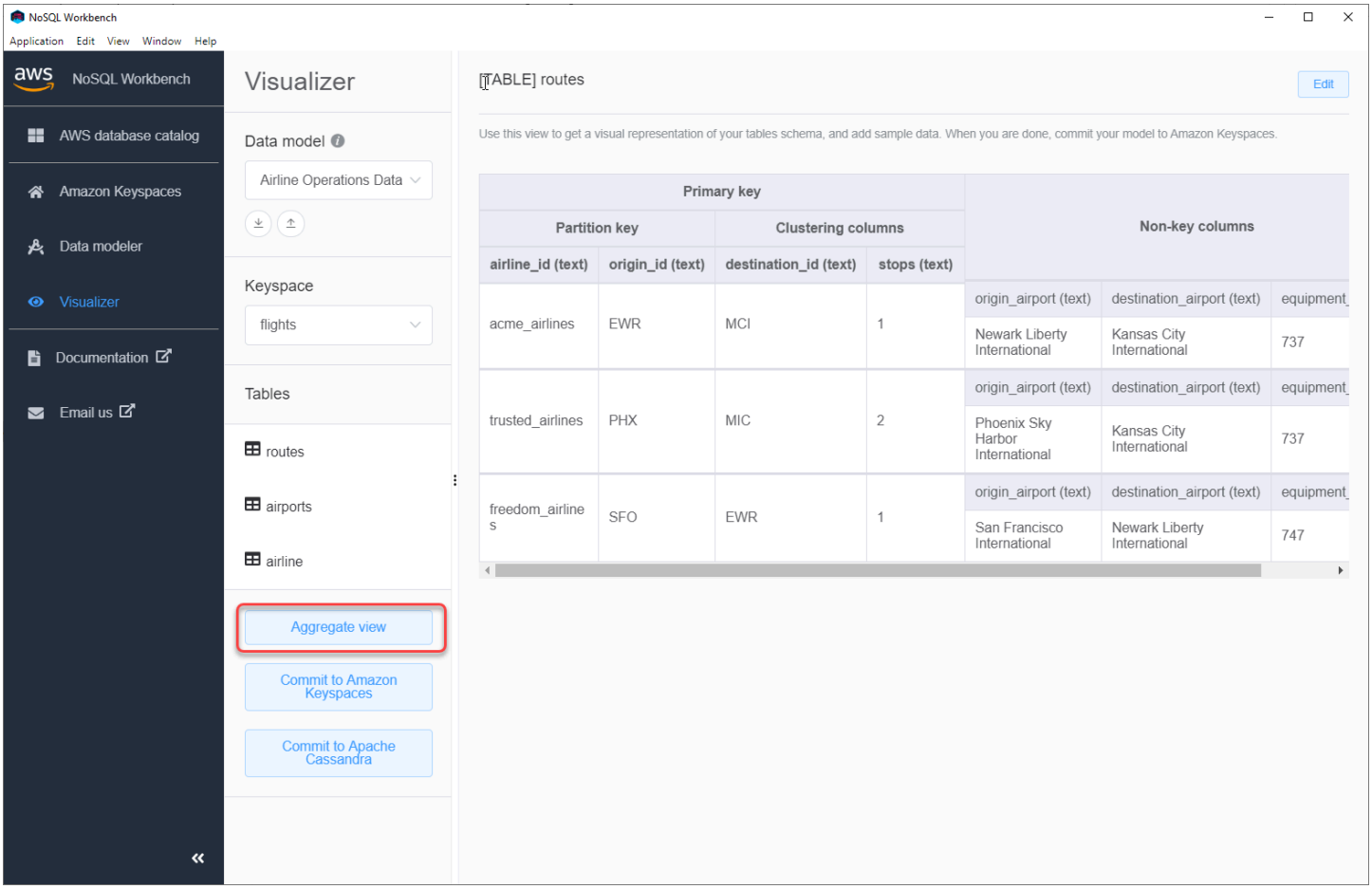


这会将您带到 NoSQL Workbench 中的数据可视化工具。数据可视化工具提供表架构的可视化表示，并允许您添加示例数据。要向表中添加样本数据，请从模型中选择一个表，然后选择编辑。要添加一行新数据，请选择屏幕底部的添加新行。完成此操作后，选择 Save。



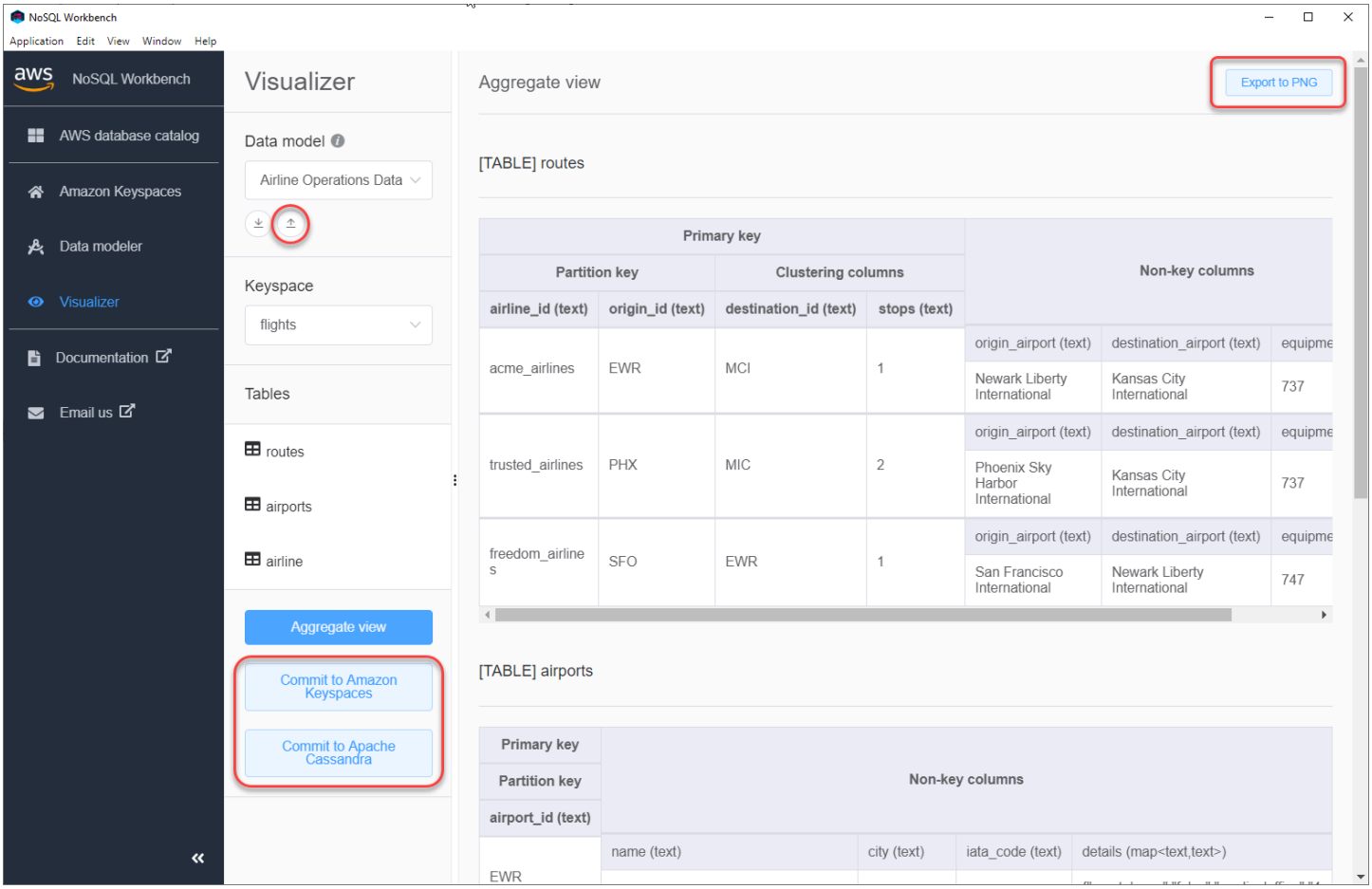
聚合视图

确认表的架构后，您可以聚合数据模型的可视化效果。



聚合数据模型视图后，可以将视图导出为 PNG 文件。要将数据模型导出到 JSON 文件，请选择数据模型名称下方的上传标志。

> **Note**
>
> 在设计过程中，您可以随时以 JSON 格式导出数据模型。



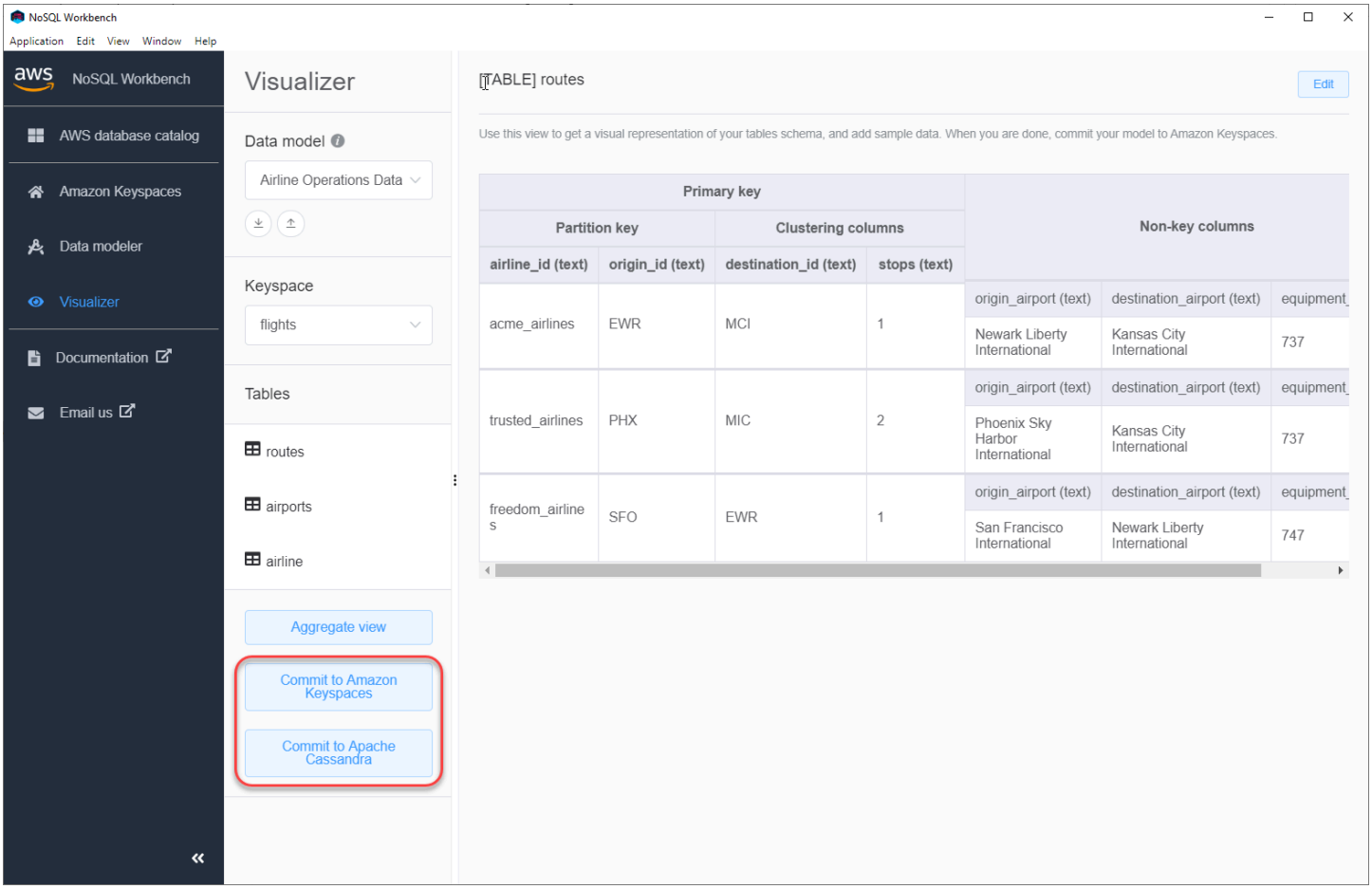
您可以通过以下几种方式提交更改：

- 提交至 Amazon Keyspaces
- 提交到 Apache Cassandra 集群

要详细了解如何提交更改，请参阅 the section called “提交数据模型”。

如何向 Amazon Keyspaces 和 Apache Cassandra 提交数据模型

本节说明了如何将完成的数据模型提交到 Amazon Keyspache 和 Apache Cassandra 集群。此过程会根据您在数据模型中定义的设置自动为键空间和表创建服务器端资源。



主题

开始前的准备工作

Amazon Keyspaces 要求使用传输层安全性协议 (TLS) 来帮助保护与客户端的连接。要使用 TLS 连接到 Amazon Keyspaces，您需要先完成以下任务。

- 使用以下命令下载 Starfield 数字证书，并将 `sf-class2-root.crt` 保存在本地或您的主目录中。

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

> **Note**
>
> 您还可以使用 Amazon 数字证书连接到 Amazon Keyspaces。如果您的客户端成功连接到 Amazon Keyspaces，您可以继续这样做。Starfield 证书为使用旧证书颁发机构的客户端提供了额外的向后兼容性。

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

保存证书文件后，您可以连接到 Amazon Keyspaces。一种选择是使用服务特定凭证进行连接。服务特定凭证是与特定 IAM 用户关联的用户名和密码，并且只能用于指定的服务。第二种选择是使用 AWS 签名版本 4 流程 (Sigv4) 的 IAM 证书进行连接。要了解有关这两种选择的更多信息，请参阅 the section called “创建凭证”。

要使用服务特定凭证进行连接，请参阅the section called “使用服务特定凭证进行连接”。

要使用 IAM 凭证进行连接，请参阅the section called “使用 IAM 凭证连接”。

使用服务特定凭证连接 Amazon Keyspaces。

本节介绍如何使用服务特定凭证提交您使用 NoSQL Workbench 创建或编辑的数据模型。

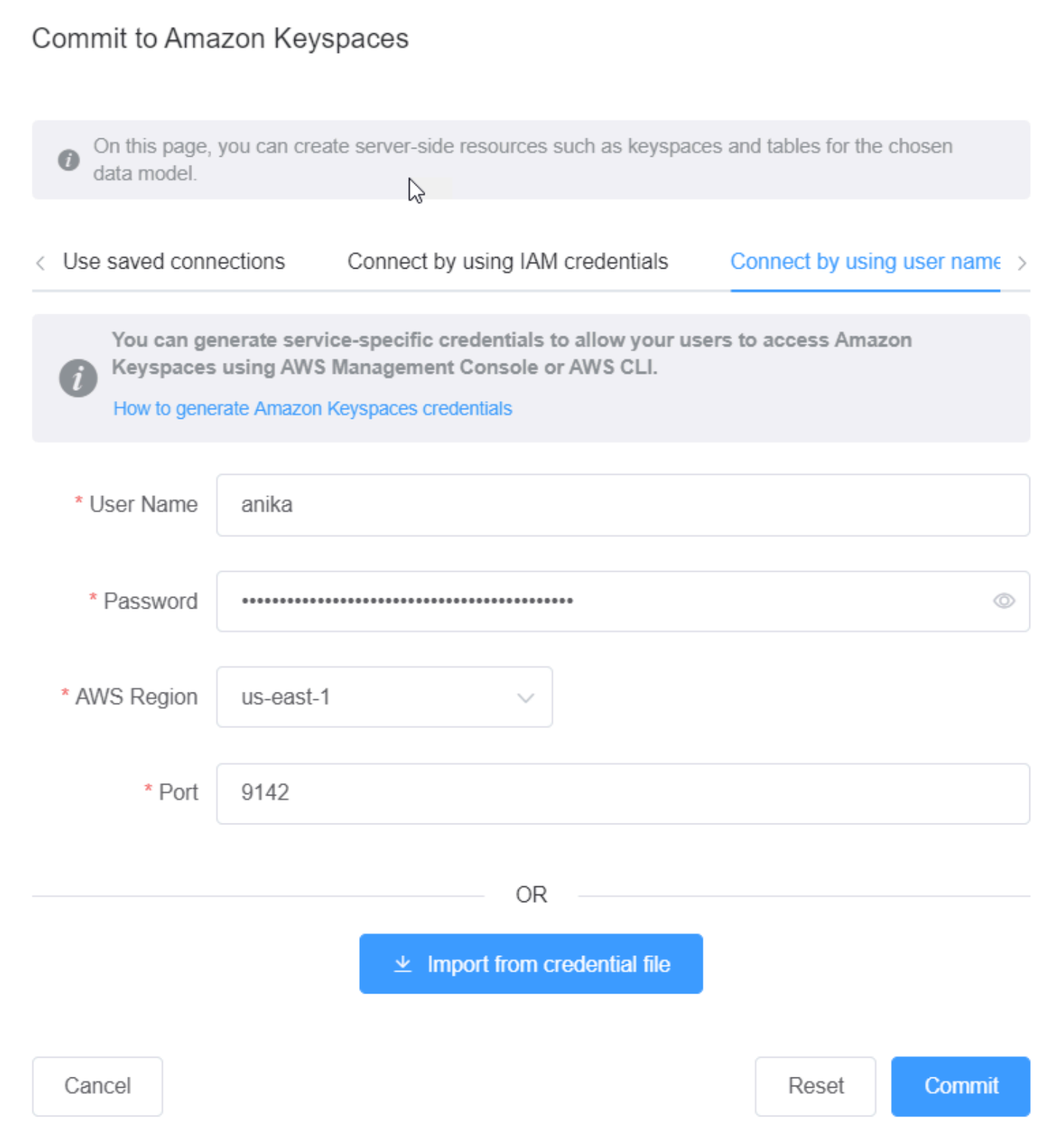
1. 要使用服务特定凭证创建新连接，请选择使用用户名和密码连接选项卡。

   - 开始之前，您必须使用 the section called “服务特定凭证” 中记录的过程创建服务特定凭证。

   获得服务特定凭证后，您可以继续设置连接。请继续执行下列操作之一：

   - 用户名：输入用户名。
   - 密码：输入密码。
   - AWS 区域：有关可用区域的信息，请参阅 the section called “服务端点”。
   - 端口：Amazon Keyspaces 使用端口 9142。

   另外，您也可以从文件中导入保存的凭证。
2. 选择提交，使用数据模型更新 Amazon Keyspaces。


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

使用 AWS Identity and Access Management (IAM) 凭证连接 Amazon Keyspaces。

本节介绍如何使用 IAM 凭证提交使用 NoSQL Workbench 创建或编辑的数据模型。

1. 要使用 IAM 凭证创建新连接，请选择使用 IAM 凭证连接选项卡。

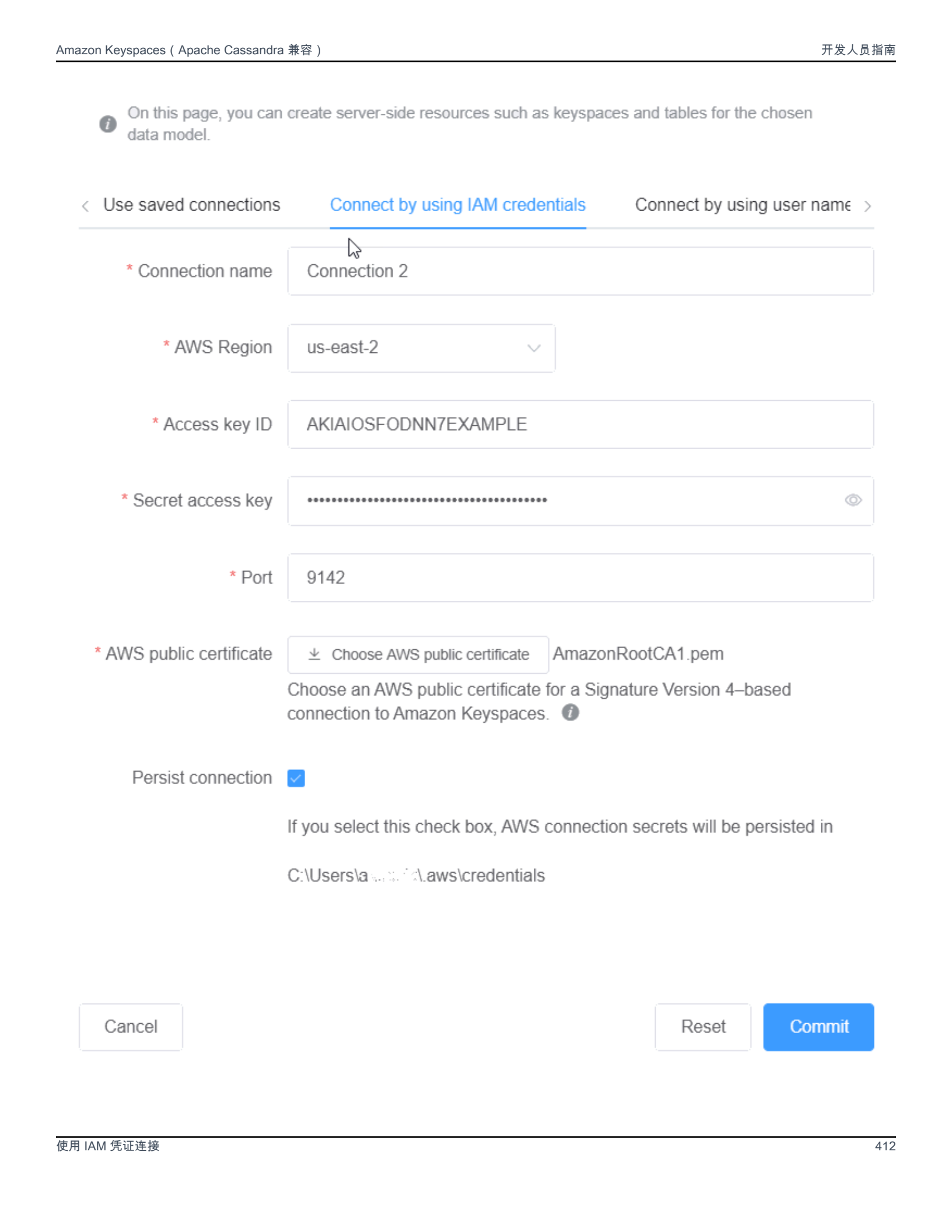
   - 开始之前，您必须使用以下方法之一创建 IAM 凭证。

     - 要访问控制台，请使用您的 IAM 用户名和密码从 IAM 登录页面登录 AWS Management Console。有关 AWS 安全凭证的信息，包括以编程方式访问和长期凭证的替代方案，请参阅《IAM 用户指南》中的 AWS 安全凭证。有关登录到 AWS 账户 的更多信息，请参阅《AWS 登录 User Guide》中的 How to sign in to AWS。
     - 要进行 CLI 访问，您需要访问密钥 ID 和秘密访问密钥。如果可能，请使用临时凭证代替长期访问密钥。临时凭证包括访问密钥 ID、秘密访问密钥，以及一个指示凭证何时到期的安全令牌。有关更多信息，请参阅《IAM 用户指南》中的将临时凭证用于 AWS 资源。
     - 对于 API 访问，您需要访问密钥 ID 和秘密访问密钥。使用 IAM 用户访问密钥而不是 AWS 账户根用户 访问密钥。有关创建访问密钥的更多信息，请参阅 IAM 用户指南 中的管理 IAM 用户的访问密钥。

     有关更多信息，请参阅管理 IAM 用户的访问密钥。

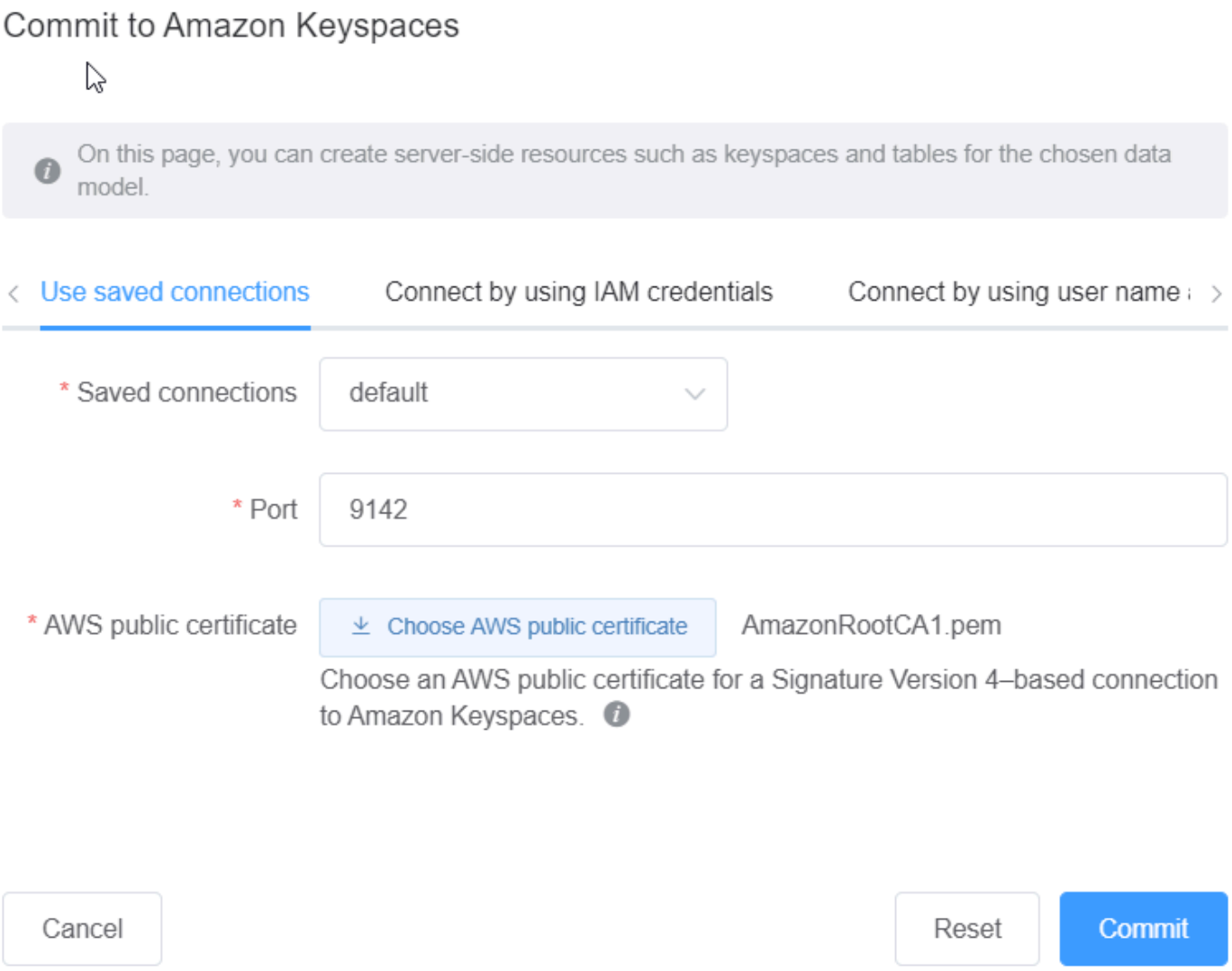
   获得 IAM 凭证后，您可以继续设置连接。

   - 连接名称：连接的名称。
   - AWS 区域：有关可用区域的信息，请参阅 the section called “服务端点”。
   - 访问密钥 ID：输入访问密钥 ID。
   - 秘密访问密钥：输入您的秘密访问密钥。
   - 端口：Amazon Keyspaces 使用端口 9142。
   - AWS 公共证书：指向在第一步中下载的 AWS 证书。
   - 保持连接：如果要将 AWS 连接密钥保存到本地，请选中该复选框。
2. 选择提交，使用数据模型更新 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
* 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

使用已保存的连接

如果您之前设置了与 Amazon Keyspaces 的连接，则可以将其用作提交数据模型更改的默认连接。选择使用已保存的连接选项卡，然后继续提交更新。

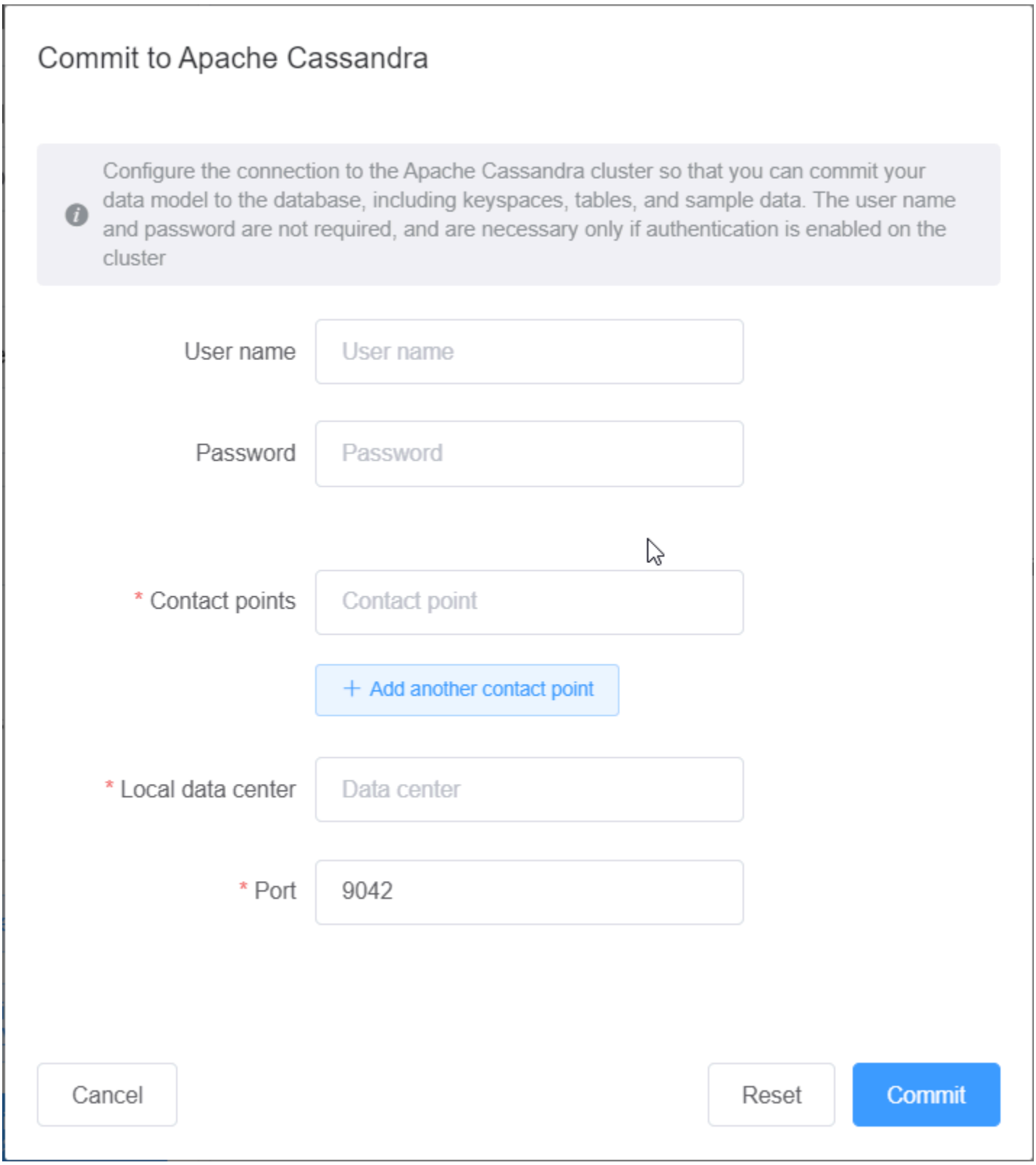


提交到 Apache Cassandra

本节将引导您完成与 Apache Cassandra 集群的连接，以提交使用 NoSQL Workbench 创建或编辑的数据模型。

> **Note**
>
> 只有使用 `SimpleStrategy` 或 `NetworkTopologyStrategy` 创建的数据模型可以提交到 Apache Cassandra 集群。要更改复制策略，请在数据建模工具中编辑键空间。

1.
   - 用户名：如果在集群上启用了身份验证，则输入用户名。
   - 密码：如果在集群上启用了身份验证，则输入密码。
   - 联络点：输入联络点。
   - 本地数据中心：输入本地数据中心的名称。
   - 端口：连接使用端口 9042。
2. 选择提交，使用数据模型更新 Apache Cassandra 集群。


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 中的示例数据模型

建模工具和可视化工具主页显示了一些随 NoSQL Workbench 提供的示例模型。本节介绍了这些模型及其潜在用途。

主题

员工数据模型

此数据模型代表员工数据库应用程序的 Amazon Keyspaces 架构。

访问给定公司员工信息的应用程序可以使用此数据模型。

该数据模型支持的访问模式是：

- 检索具有给定 ID 的员工记录。
- 检索具有给定 ID 和部门的员工记录。
- 检索具有给定 ID 和姓名的员工记录。

信用卡交易数据模型

此数据模型代表零售商店信用卡交易的 Amazon Keyspaces 架构。

信用卡交易的存储不仅可以帮助商店记帐，还可以帮助商店经理分析购买趋势，这可以帮助他们进行预测和规划。

该数据模型支持的访问模式是：

- 按信用卡号、月份和年份以及日期检索交易。
- 按信用卡号、类别和日期检索交易。
- 按类别、位置和信用卡号检索交易。
- 按信用卡号和争议状态检索交易。

航空公司运营数据模型

该数据模型显示有关飞机航班的数据，包括机场、航空公司和飞行路线。

演示的 Amazon Keyspaces 建模的关键组件包括键值对、宽列数据存储、复合键和复杂数据类型，例如用于演示常见 NoSQL 数据访问模式的地图。

该数据模型支持的访问模式是：

- 在给定机场检索从给定航空公司出发的路线。
- 检索具有给定目的地机场的路线。
- 检索有直飞航班的机场。
- 检索机场详情和航空公司详情。

NoSQL Workbench 的发布历史记录

下表介绍了 NoSQL Workbench 客户端应用程序每一版中的重要更改。

更改	说明	日期
适用于 Amazon Keyspaces 的 NoSQL Workbench – GA。	适用于 Amazon Keyspaces 的 NoSQL Workbench 已正式推出。	2020 年 10 月 28 日
NoSQL Workbench 预览版已发布。	NoSQL Workbench 是一款客户端应用程序，可帮助您更轻松地设计和可视化用于 Amazon Keyspaces 的非关系数据模型。NoSQL Workbench 客户端可用于 Windows、macOS 和 Linux。有关更多信息，请参阅适用于 Amazon Keyspaces 的 NoSQL Workbench。	2020 年 10 月 5 日

适用于 Amazon Keyspaces（Apache Cassandra 兼容）的多区域复制

您可以使用 Amazon Keyspaces 多区域复制，通过自动、完全托管、主动-主动复制，在您选择的范围内复制数据。 AWS 区域 通过主动/主动复制，每个区域都能够独立执行读取和写入。您可以通过区域退化来提高可用性和弹性，同时还可以从全球应用程序的低延迟本地读取和写入中受益。

通过多区域复制，Amazon Keyspaces 可在区域之间异步复制数据，并且数据通常在一秒钟内即可跨区域传播。此外，借助多区域复制，您就不再需要负责解决冲突和纠正数据差异问题的繁重工作，从而可以专注于您的应用程序。

默认情况下， AWS 区域 为了持久性和高可用性，Amazon Keyspaces 会在同一个可用区内的三个可用区之间复制数据。借助多区域复制，您可以创建多区域密钥空间，在您选择的多达六个不同的地理位置 AWS 区域 复制您的表。

主题

使用多区域复制的好处

多区域复制具有以下好处。

- 全局读取和写入延迟为个位数毫秒 — 在 Amazon Keyspaces 中，复制是主动-主动的。您可以在任何规模下，以不超过十毫秒的延迟从距离客户最近的区域提供本地读取和写入服务。您可以将 Amazon Keyspaces 多区域表用于在世界任何地方都需要快速响应时间的全球应用程序。
- 改善业务连续性并防止单区域降级 — 借助多区域复制，您可以将应用程序重定向到多区域密钥空间中的其他区域， AWS 区域 从而在单个区域中从降级中恢复。由于 Amazon Keyspaces 提供主动/主动复制，因此不会对您的读取和写入产生任何影响。

  Amazon Keyspaces 会跟踪已在多区域键空间上执行但尚未传播到所有副本区域的任何写入。该区域恢复在线后，Amazon Keyspaces 会自动同步所有缺失的更改，这样您就可以在不影响应用程序的情况下进行恢复。

- 跨区域的高速复制 — 多区域复制使用基于存储的跨区域快速物理复制数据，复制延迟通常小于 1 秒。

  Amazon Keyspaces 中的复制对您的数据库查询几乎没有影响，因为它不与您的应用程序共享计算资源。这意味着您可以解决高写入吞吐量用例或吞吐量突然激增或突发的用例，而不会对应用程序造成任何影响。
- 一致性和冲突解决方案 — 对任何区域的数据所做的任何更改都将复制到多区域密钥空间中的其他区域。如果应用程序同时在不同区域更新了相同的数据，则会出现冲突。

  为了帮助提供最终一致性，Amazon Keyspaces 在并发更新之间使用单元级时间戳和以最后写入者为准协调机制。冲突解决是完全托管的，并且在后台进行，不会对应用程序产生任何影响。

有关支持的配置和功能的更多信息，请参阅 the section called “使用说明”。

容量模式和定价

对于多区域密钥空间，您可以使用按需容量模式或预配置容量模式。有关更多信息，请参阅the section called “读/写容量模式”。

对于按需模式，每行最多写入 1 KB 的数据需要支付 1.25 个写入请求单元 (WRU) 的费用。您需要为多区域密钥空间的每个区域的写入付费。例如，在具有两个区域的多区域键空间中写入一行 3 KB 数据需要 7.5 个 WRU：3 * 1.25 * 2 = 7.5 个 WRU。此外，包含静态数据和非静态数据的写入需要额外的写入操作。

对于预配置模式，每行写入最多 1 KB 的数据需要支付 1.25 个写入容量单位 (WCU) 的费用。您需要为多区域密钥空间的每个区域的写入付费。例如，在具有两个区域的多区域密钥空间中写入一行每秒 3 KB 的数据需要 7.5 WCU：3 * 1.25 * 2 = 7.5 WCU。此外，包含静态数据和非静态数据的写入需要额外的写入操作。

有关定价的更多信息，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定价。

多区域复制在 Amazon Keyspaces 中的工作原理

本节概述了 Amazon Keyspaces 多区域复制的工作原理。有关定价的更多信息，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定价。

主题

多区域复制在 Amazon Keyspaces 中的工作原理

Amazon Keyspaces 多区域复制实施了一种数据弹性架构，可将您的数据分布在独立且地理位置分散的 AWS 区域。它使用主动/主动复制，提供本地低延迟，每个区域都能够独立执行读取和写入。

创建 Amazon Keyspaces 多区域键空间时，您最多可以再选择五个区域来将数据复制到这些区域。您在多区域键空间中创建的每个表都包含多个副本表（每个区域一个），Amazon Keyspaces 将这些副本表视为一个单元。

每个副本都具有相同的表名和相同的主键架构。当应用程序将数据写入一个区域中的本地表时，数据将使用 `LOCAL_QUORUM` 一致性级别持久写入。Amazon Keyspaces 会自动将数据异步复制到其他复制区域。跨区域的复制滞后通常小于一秒，并且不会影响应用程序的性能或吞吐量。

数据写入后，您可以从另一个具有 `LOCAL_ONE/LOCAL_QUORUM` 一致性级别的复制区域的多区域表中读取数据。有关支持的配置和功能的更多信息，请参阅 the section called “使用说明”。

多区域复制冲突解决方案

Amazon Keyspaces 多区域复制是完全托管的，这意味着您不必执行复制任务，例如定期运行修复操作来清理数据同步问题。Amazon Keyspaces 通过检测和修复冲突来监控不同 AWS 区域中表之间的数据一致性，并自动同步副本。

Amazon Keyspaces 使用以最后写入者为准数据协调方法。通过这种冲突解决机制，多区域键空间中的所有区域都将同意最新的更新，并收敛到它们都具有相同数据的状态。协调过程对应用程序性能没有影响。为了支持冲突解决，多区域表的客户端时间戳会自动启用并且无法禁用。有关更多信息，请参阅客户端时间戳。

多区域复制灾难恢复

使用 Amazon Keyspaces 多区域复制，读取和写入都可以在每个区域异步复制。在极少数情况下，如果单个区域性能下降或出现故障，多区域复制可以帮助您在对应用程序几乎没有影响的情况下从灾难中恢复。灾难恢复通常使用恢复时间目标 (RTO) 和恢复点目标 (RPO) 的值来衡量。

恢复时间目标：灾难后系统恢复工作状态所需的时间。RTO 用于衡量您的工作负载可以容忍的停机时间（按时间衡量）。对于使用多区域复制失效转移到未受影响区域的灾难恢复计划，RTO 几乎为零。RTO 受到应用程序检测到故障并将流量重定向到另一个区域的速度的限制。

恢复点目标：可能丢失的数据量（按时间衡量）。对于使用多区域复制失效转移到未受影响区域的灾难恢复计划，RPO 通常不到 10 秒。RPO 受到失效转移目标副本的复制延迟的限制。

如果出现区域性故障或性能下降，您无需提升辅助区域或执行数据库失效转移过程，因为 Amazon Keyspaces 中的复制采用主动/主动模式。相反，您可以使用 Amazon Route 53 将您的应用程序路由到最近的运行状况良好的区域。要了解有关 Route 53 的更多信息，请参阅什么是 Amazon Route 53？。

如果单个 AWS 区域变得孤立或性能下降，您的应用程序可以使用 Route 53 将流量重定向到不同的区域，以对不同的副本表执行读取和写入操作。您还可以应用自定义业务逻辑来确定何时将请求重定向到其他区域。例如，这可让您的应用程序知道有多个可用的端点。

当该区域恢复联机状态时，Amazon Keyspaces 会继续将任何待处理的写入操作从该区域传播到其他区域中的副本表。它还会继续将写入从其他副本表传播到现在重新联机的区域。

创建多区域键空间和表所需的 IAM 权限

要成功创建多区域键空间和表，IAM 主体需要能够创建服务相关角色。该服务相关角色是一种独特的 IAM 角色类型，由 Amazon Keyspaces 预定义。它包括 Amazon Keyspaces 代表您执行操作所需的所有权限。有关 service-linked role 服务相关角色的更多信息，请参阅the section called “多区域复制”。

要创建多区域复制所需的服务相关角色，IAM 主体的策略需要以下元素：

- `iam:CreateServiceLinkedRole`：主体可以执行的操作。
- `arn:aws:iam::*:role/aws-service-role/replication.cassandra.amazonaws.com/AWSServiceRoleForKeyspacesReplication`：可对其执行操作的资源。
- `iam:AWSServiceName": "replication.cassandra.amazonaws.com`：此角色可以附加到的唯一 AWS 服务是 Amazon Keyspaces。

以下是向主体授予创建多区域键空间和表所需的最低权限的策略示例。

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

有关多区域键空间和表的其他 IAM 权限，请参阅《服务授权参考》中的 Amazon Keyspaces（Apache Cassandra 兼容）的操作、资源和条件键。

多区域复制以及与时间点故障恢复 (PITR) 的集成

多区域表支持时间点故障恢复。要使用 PITR 成功还原多区域表，必须满足以下条件。

- 源表和目标表必须配置为多区域表。
- 源表键空间和目标表键空间的复制区域必须相同。

您可以从源表可用的任何区域运行还原语句。Amazon Keyspaces 会自动还原每个区域中的目标表。有关 PITR 的更多信息，请参阅the section called “工作方式”。

多区域复制以及与 AWS 服务的集成

您可以使用 Amazon CloudWatch 指标监控不同 AWS 区域中的表之间的复制性能。以下指标提供对多区域键空间的持续监控。

- `ReplicationLatency`：该指标可衡量将 `updates`、`inserts` 或 `deletes` 从一个副本表复制到多区域键空间中的另一个副本表所花费的时间。

有关如何使用 CloudWatch 指标的更多信息，请参阅 the section called “使用监控 CloudWatch”。

Amazon Keyspaces 多区域复制使用说明

使用 Amazon Keyspaces 多区域复制时，请考虑以下几点。

- 您最多可以选择六个可用的公众 AWS 区域。 AWS GovCloud (US) Regions、中国区域以及 AWS 区域 默认情况下处于禁用的区域均不支持。
- 请谨慎选择键空间的复制区域，因为稍后无法添加或删除它们。
- 在创建多区域表之前确定表架构，因为稍后无法添加新列。
- 对于静态加密，请使用 AWS 自有密钥。多区域表不支持客户自主管理型密钥。有关更多信息，请参阅

the section called “工作原理”.

- 当您将预配置容量管理与 Amazon Keyspaces 自动扩展配合使用时，请务必使用 Amazon Keyspaces API 操作来创建和配置您的多区域表。Amazon Keyspaces 代表您调用的底层应用程序 Auto Scaling API 操作不具有多区域功能。

  有关更多信息，请参阅the section called “如何使用多区域复制”。有关如何估算已配置的多区域表的写入容量吞吐量的更多信息，请参阅。the section called “多区域表”
- 确定表是否需要生存时间 (TTL)。稍后您将无法启用生存时间。有关更多信息，请参阅使用生存时间让数据过期。
- 尽管数据会在多区域表的选定区域之间自动复制，但当客户端连接到一个区域中的端点并查询 `system.peers` 表时，该查询仅返回本地信息。查询结果对于客户端来说就像一个数据中心集群。
- Amazon Keyspaces 多区域复制是异步的，它支持写入`LOCAL_QUORUM`的一致性。`LOCAL_QUORUM`一致性要求在本地区域的两个副本上持久保留对行的更新，然后才能将成功返回给客户端。然后以异步方式向复制的区域（一个或多个区域）执行写入的传播。

  Amazon Keyspaces 多区域复制不支持同步复制或一致性。`QUORUM`
- 创建多区域键空间或表时，您在创建过程中定义的任何标签都会自动应用于所有区域中的所有键空间和表。使用`ALTER KEYSPACE`或更改现有标签时`ALTER TABLE`，更新仅适用于您进行更改的区域中的密钥空间或表。
- Amazon 为每个复制区域 CloudWatch 提供了一个`ReplicationLatency`指标。它通过跟踪到达的行、将它们的到达时间与初始写入时间进行比较并计算平均值来计算该指标。时间存储 CloudWatch 在源区域内。有关更多信息，请参阅the section called “使用监控 CloudWatch”。

  查看平均和最大时机以确定平均和最坏情况下的复制延迟可能很有用。对于这种延迟，没有 SLA。

如何使用多区域复制

您可以使用亚马逊密钥空间（适用于 Apache Cassandra）控制台、Cassandra 查询语言 (CQL)、软件开发工具包和 () 来创建和管理多区域密钥空间和表。 AWS AWS Command Line Interface AWS CLI

本节提供示例，说明如何使用控制台、CQL 以及使用按需和预配置容量模式使用控制台创建多区域密钥空间和表。 AWS CLI在多区域密钥空间中创建的所有表都会自动从密钥空间继承多区域设置。

本节还包括如何使用控制台、CQL 和来管理已配置的多区域表的 Amazon Keyspaces 自动扩展设置的示例。 AWS CLI 有关常规 auto Scaling 配置选项及其工作原理的更多信息，请参阅the section called “使用 auto scaling 管理吞吐容量”。

请注意，如果您对多区域表使用预配置容量模式，则必须始终使用 Amazon Keyspaces API 调用来配置自动扩展。这是因为底层的 Application Auto Scaling API 操作不支持区域。

有关如何估算已配置的多区域表的写入容量吞吐量的更多信息，请参阅。the section called “多区域表”

有关亚马逊密钥空间 API 的更多信息，请参阅亚马逊密钥空间 API 参考。

有关支持的配置和多区域复制功能的更多信息，请参阅the section called “使用说明”。

主题

- 使用控制台创建和管理多区域表
- 使用 CQL 创建和管理多区域表
- 使用 AWS CLI 创建和管理多区域表

使用控制台创建和管理多区域表

本节提供了如何使用 Amazon Keyspaces（适用于 Apache Cassandra）控制台在按需和预配置容量模式下创建多区域密钥空间和表的示例。您在多区域密钥空间中创建的所有表都会自动从密钥空间继承多区域设置。

有关 CQL 示例，请参阅the section called “使用 CQL”。有关 AWS CLI 示例，请参见the section called “使用 AWS CLI”。

主题

- 创建多区域键空间（控制台）
- 使用默认设置创建多区域表（控制台）
- 在启用了 auto Scaling 的预配置模式下创建多区域表（控制台）
- 为现有多区域表启用 auto Scaling（控制台）
- 关闭多区域表的 auto 缩放（控制台）
- 在控制台上查看 Amazon Keyspaces 的自动扩展活动

创建多区域键空间（控制台）

按照以下步骤使用 Amazon Keyspaces 控制台创建新的多区域密钥空间。

创建多区域键空间（控制台）

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 在导航窗格中，选择 Keyspaces (键空间)，然后选择 Create keyspace (创建键空间)。
3. 对于键空间名称，输入键空间的名称。
4. 在多区域复制部分，您最多可以添加列表中可用的另外五个区域。
5. 要完成操作，请选择创建键空间。

> **Note**
>
> 当您创建多区域键空间时，Amazon Keyspaces 会在您的账户中创建一个名为 `AWSServiceRoleForAmazonKeyspacesReplication` 的服务相关角色。此角色允许 Amazon Keyspaces 代表您将写入复制到多区域表的所有副本。要了解更多信息，请参阅the section called “多区域复制”。

使用默认设置创建多区域表（控制台）

按照以下步骤使用 Amazon Keyspaces 控制台创建多区域表。

创建多区域表（控制台）

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 选择一个多区域键空间。
3. 在表选项卡上，选择创建表。
4. 对于表名称，输入表的名称。正在其中复制此表的 AWS 区域 会显示在信息框中。
5. 继续设置表架构。
6. 在表格设置下，继续使用默认设置选项。请注意多区域表的以下默认设置。
   - 容量模式-默认容量模式为按需。有关配置预配模式的更多信息，请参阅the section called “在启用了 auto Scaling 的预配置模式下创建多区域表（控制台）”。
   - 加密密钥管理：仅支持 AWS 拥有的密钥选项。
   - 客户端时间戳：多区域表需要此功能。

- 如果您需要为该表及其所有副本启用生存时间 (TTL)，请选择自定义设置。

> **Note**
>
> 您将无法更改现有多区域表的 TTL 设置。

7. 要完成操作，请选择创建表。

在启用了 auto Scaling 的预配置模式下创建多区域表（控制台）

> **Note**
>
> Amazon Keyspaces 自动扩缩需要存在一个代表您执行自动扩缩操作的服务相关角色 (`AWSServiceRoleForApplicationAutoScaling_CassandraTable`)。将自动为您创建此角色。有关更多信息，请参阅the section called “使用服务相关角色”。

创建启用自动缩放功能的新多区域表

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 选择一个多区域键空间。
3. 在表选项卡上，选择创建表。
4. 在创建表页面的表详细信息部分中，选择一个键空间并为新表提供一个名称。
5. 在 “列” 部分中，为您的表创建架构。
6. 在主键部分中，定义表的主键并选择可选的聚类列。
7. 在表设置部分，选择自定义设置。
8. 继续读取/写入容量设置。
9. 对于 Capacity mode (容量模式)，选择 Provisioned (预置)。
10. 在 Read capacity (读取容量) 部分中，确认已选择 Scale automatically(自动扩展)。

    您可以选择为所有 AWS 区域 复制表的读取容量单位配置相同的读取容量单位。或者，您可以清除该复选框并以不同的方式配置每个区域的读取容量。

如果您选择以不同的方式配置每个区域，则可以为每个表副本选择最小和最大读取容量单位以及目标利用率。

- 最小容量单位：输入表应始终支持的最小吞吐量级别的值。该值必须介于 1 和账户的每秒最大吞吐量配额（默认为 40000）之间。
- 最大容量单位-输入要为表预配置的最大吞吐量。该值必须介于 1 和账户的每秒最大吞吐量配额（默认为 40000）之间。
- 目标利用率：输入介于 20% 和 90% 之间的目标利用率。当流量超过定义的目标利用率时，容量将自动扩展。当流量低于定义的目标时，容量将自动重新缩减。

- 如果要手动配置表的读取容量，请清除“自动扩展”复选框。此设置适用于表的所有副本。

> **Note**
>
> 为确保所有副本都有足够的读取容量，我们建议 Amazon Keyspaces 自动扩展预配置的多区域表。

> **Note**
>
> 要了解有关账户的默认配额以及如何增加此配额的更多信息，请参阅 限额。

11. 在“写入容量”部分，确认已选择“自动缩放”。然后为表配置容量单位。写入容量单位在所有区域之间保持同步， AWS 区域 以确保有足够的容量跨区域复制写入事件。
    - 如果要手动配置表的写入容量，请清除“自动扩展”。此设置适用于表的所有副本。

    > **Note**
    >
    > 为确保所有副本都有足够的写入容量，我们建议 Amazon Keyspaces 自动扩展预配置的多区域表。

12. 选择创建表。使用指定的自动扩展参数创建表。

为现有多区域表启用 auto Scaling（控制台）

使用 Amazon Keyspaces 控制台，按照以下步骤在预配置模式下为多区域表启用自动缩放。

> **Note**
>
> Amazon Keyspaces 自动扩缩需要存在一个代表您执行自动扩缩操作的服务相关角色 (`AWSServiceRoleForApplicationAutoScaling_CassandraTable`)。将自动为您创建此角色。有关更多信息，请参阅the section called “使用服务相关角色”。

为现有多区域表启用 Amazon Keyspaces 自动缩放

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 选择要使用的表，然后转到 “容量” 选项卡。
3. 在容量设置部分，选择编辑。
4. 在容量模式下，确保表使用预置容量模式。
5. 选择 “自动扩展”，请参阅中的步骤 9 在启用了 auto Scaling 的预配置模式下创建多区域表（控制台） 以编辑读取和写入容量。
6. 在定义自动扩展设置时，选择 Save (保存)。

关闭多区域表的 auto 缩放（控制台）

按照以下步骤使用 Amazon Keyspaces 控制台在预配置模式下关闭多区域表的自动缩放。

关闭现有多区域表的 Amazon Keyspaces 自动缩放功能

1. 登录并打开 Amazon Keyspaces 控制台，网址为 https://console.aws.amazon.com/keyspaces/home。 AWS Management Console
2. 选择要使用的表，然后选择 “容量” 选项卡。
3. 在容量设置部分，选择编辑。
4. 要禁用 Amazon Keyspaces 的自动缩放，请清除 “自动缩放” 复选框。禁用自动缩放功能会在 Application Auto Scaling 中取消表作为可扩展目标的注册。要删除 Application Auto Scaling 用于访问您的 Amazon Keyspaces 表的服务相关角色，请按照中的步骤操作。the section called “删除适用于 Amazon Keyspaces 的服务相关角色”

> **Note**
>
> 要删除 Application Auto Scaling 使用的服务相关角色，必须禁用账户中所有表的自动缩放功能。 AWS 区域

5. 在定义自动扩展设置时，选择 Save (保存)。

在控制台上查看 Amazon Keyspaces 的自动扩展活动

您可以使用亚马逊来监控 Amazon Keyspaces 自动扩展如何使用资源 CloudWatch，亚马逊会生成有关您的使用情况和性能的指标。按照《Application Auto Scaling 用户指南》中的步骤创建 CloudWatch 仪表板。

使用 CQL 创建和管理多区域表

您可以使用 Cassandra 查询语言 (CQL) 在 Amazon Keyspaces 中创建和管理多区域密钥空间和表。

本节提供了如何使用 CQL 创建和管理多区域表的示例。您在多区域密钥空间中创建的所有表都会自动从密钥空间继承多区域设置。有关 CQL 的更多信息，请参阅 Amazon Keyspaces C QL 语言参考。

有关支持的配置和功能的更多信息，请参阅 the section called “使用说明”。

主题

- 创建多区域键空间 (CQL)
- 使用默认设置创建多区域表 (CQL)
- 创建具有预置容量模式和 auto Scaling (CQL) 的多区域表
- 更新多区域表 (CQL) 的预配置容量和 auto scaling 设置
- 查看多区域表 (CQL) 的预配置容量和 auto Scaling 设置
- 关闭多区域表 (CQL) 的自动缩放
- 手动设置多区域表的预配置容量 (CQL)

创建多区域键空间 (CQL)

要创建多区域密钥空间，请使用指定`NetworkTopologyStrategy` AWS 区域 要在其中复制密钥空间。您必须包括您当前的区域和至少一个其他区域。下面是一个示例 CQL 语句。

```
CREATE KEYSPACE mykeyspace
```

```
WITH REPLICATION = {'class':'NetworkTopologyStrategy', 'us-east-1':'3', 'ap-
southeast-1':'3','eu-west-1':'3' };
```

键空间中的所有表都使用与键空间相同的复制策略。您无法更改表级别的复制策略。

NetworkTopologyStrategy— 每个区域的重复系数为三，因为默认情况下，Amazon Keyspaces 会在同一个AWS 区域区域内的三个可用区之间复制数据。

> **Note**
>
> 当您创建多区域键空间时，Amazon Keyspaces 会在您的账户中创建一个名为 AWSServiceRoleForAmazonKeyspacesReplication 的服务相关角色。此角色允许 Amazon Keyspaces 代表您将写入复制到多区域表的所有副本。要了解更多信息，请参阅the section called “多区域复制”。

您可以使用 CQL 语句在system_multiregion_info密钥空间中查询tables表，以编程方式列出您指定的多区域表的区域和状态。下面是一个代码示例。

```
SELECT * from system_multiregion_info.tables WHERE keyspace_name = 'mykeyspace' AND
 table_name = 'mytable';
```

该语句的输出如下所示：

```
 keyspace_name  | table_name     | region         | status
----------------+----------------+----------------+--------
 mykeyspace     | mytable        | us-east-1      | ACTIVE
 mykeyspace     | mytable        | ap-southeast-1 | ACTIVE
 mykeyspace     | mytable        | eu-west-1      | ACTIVE
```

使用默认设置创建多区域表 (CQL)

要使用默认设置创建多区域表，可以使用以下示例。

```
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'
 }
};
```

创建具有预置容量模式和 auto Scaling (CQL) 的多区域表

要使用 auto Scaling 在预配置模式下创建多区域表，必须先通过CUSTOM_PROPERTIES为表定义来指定容量模式。指定预配置容量模式后，您可以使用配置表的 auto Scaling 设置。AUTOSCALING_SETTINGS

有关 auto Scaling 设置、目标跟踪策略、目标值和可选设置的详细信息，请参阅the section called “使用 CQL 创建具有自动缩放功能的新表”。

创建多区域表时，您还可以为表的每个副本指定不同的读取容量和读取 auto Scaling 设置。您指定的设置会覆盖表中指定内容的常规设置。 AWS 区域但是，写入容量在所有副本之间保持同步，以确保有足够的容量在所有区域之间复制写入。

要定义特定区域中表副本的读取容量，您可以将以下参数配置为表的一部分replica_updates：

- 区域
- 预配置的读取容量单位（可选）
- 读取容量的自动缩放设置（可选）

以下示例显示了预配置模式下多区域表的CREATE TABLE语句。一般的写入和读取容量 auto scaling 设置相同。但是，在向上或向下扩展表的读取容量之前，read auto scaling 设置指定了 60 秒的额外冷却时间。此外，美国东部区域（弗吉尼亚北部）的读取容量 auto scaling 设置高于其他副本的读取容量 auto scaling 设置。此外，目标值设置为 70% 而不是 50%。

```
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
                    }
                }
            }
        }
    }
};
```

更新多区域表 (CQL) 的预配置容量和 auto scaling 设置

您可以使用`ALTER TABLE`更新现有表的容量模式和 auto Scaling 设置。如果您要更新当前处于按需容量模式的表，`capacity_mode`则为必填项。如果您的表已处于预置容量模式，则可以省略此字段。

有关 auto Scaling 设置、目标跟踪策略、目标值和可选设置的详细信息，请参阅the section called “使用 CQL 创建具有自动缩放功能的新表”。

在同一语句中，您还可以通过更新表的replica_updates属性来更新特定区域中表副本的读取容量和自动缩放设置。下面是一个示例语句。

```
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
                    }
                }
            }
        }
```

```
    }
};
```

查看多区域表 (CQL) 的预配置容量和 auto Scaling 设置

要查看多区域表的 auto Scaling 配置，请使用以下命令。

```
SELECT * FROM system_multiregion_info.autoscaling WHERE keyspace_name = 'mykeyspace'
 AND table_name = 'mytable';
```

此命令的输出如下所示：

```
 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}}}
```

关闭多区域表 (CQL) 的自动缩放

您可以使用关闭ALTER TABLE现有表的 auto 缩放。请注意，您无法为单个表副本禁用 auto 缩放。

在以下示例中，针对表的读取容量关闭了 auto Scaling。

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

> **Note**
>
> 要删除 Application Auto Scaling 使用的服务相关角色，您必须在所有 AWS 区域中禁用账户中所有表的自动扩缩。

手动设置多区域表的预配置容量 (CQL)

如果您必须关闭多区域表的 auto Scaling，则可以使用ALTER TABLE手动为副本表配置表的读取容量。

```
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**
>
> 我们建议对使用预配置容量的多区域表使用 auto scaling。有关更多信息，请参阅 the section called “多区域表”。

使用 AWS CLI 创建和管理多区域表

您可以使用 AWS Command Line Interface (AWS CLI) 在 Amazon Keyspaces 中创建和管理多区域密钥空间和表。

本节提供了如何使用创建和管理多区域表的 AWS CLI示例。您在多区域密钥空间中创建的所有表都会自动从密钥空间继承多区域设置。

有关本主题中描述的 Amazon Key AWS CLI spaces 命令的更多信息，请参阅《亚马逊密钥空间AWS CLI 命令参考》。

主题

- 创建新的多区域键空间 (CLI)
- 使用默认设置创建新的多区域表 (CLI)
- 使用 auto Scaling (CLI) 在预配置模式下创建新的多区域表
- 更新多区域表 (CLI) 的预配置容量和 auto Scaling 设置
- 查看多区域表 (CLI) 的预配置容量和 auto Scaling 设置
- 关闭多区域表的自动缩放 (CLI)
- 手动设置多区域表的预配置容量 (CLI)

创建新的多区域键空间 (CLI)

要创建多区域键空间，您可以使用以下 CLI 语句。请指定您当前的区域和 `regionList` 中的至少一个其他区域。

```
aws keyspaces create-keyspace --keyspace-name mykeyspace
            \ --replication-specification
 replicationStrategy=MULTI_REGION,regionList=us-east-1,eu-west-1
```

> **Note**
>
> 当您创建多区域键空间时，Amazon Keyspaces 会在您的账户中创建一个名为 AWSServiceRoleForAmazonKeyspacesReplication 的服务相关角色。此角色允许 Amazon Keyspaces 代表您将写入复制到多区域表的所有副本。要了解更多信息，请参阅the section called “多区域复制”。

使用默认设置创建新的多区域表 (CLI)

要使用默认设置创建多区域表，您只需要指定架构即可。你可以使用以下示例。

```
aws keyspaces create-table --keyspace-name mykeyspace --table-name mytable
            \ --schema-definition 'allColumns=[{name=pk,type=int}],partitionKeys={name=
 pk}'
```

该命令的输出是：

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

要确认表的设置，可以使用以下语句。

```
aws keyspaces get-table --keyspace-name mykeyspace --table-name mytable
```

输出显示了多区域表的所有默认设置。

```
{
    "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
            }
        }
```

```
    ]
}
```

使用 auto Scaling (CLI) 在预配置模式下创建新的多区域表

要在预配置模式下创建具有 auto Scaling 配置的多区域表，可以使用。 AWS CLI请注意，您必须使用 Amazon Keyspaces CLI `create-table` 命令来配置多区域自动扩展设置。这是因为 Amazon Keyspaces 用来代表您执行自动扩展的服务 Application Auto Scaling 不支持多个区域。

有关 auto Scaling 设置、目标跟踪策略、目标值和可选设置的更多信息，请参阅the section called “使用创建具有自动缩放功能的新表 AWS CLI”。

当您在预配置模式下使用自动缩放设置创建新的多区域表时，您可以为该表指定对复制 AWS 区域 该表的所有内容有效的常规设置。然后，您可以覆盖每个副本的读取容量设置并读取 auto Scaling 设置。但是，写入容量在所有副本之间保持同步，以确保有足够的容量跨所有区域复制写入。

要定义特定区域中表副本的读取容量，您可以将以下参数配置为表的一部分`replicaSpecifications`：

- 区域
- 预配置的读取容量单位（可选）
- 读取容量的自动缩放设置（可选）

当您使用复杂的自动缩放设置和不同的表副本配置创建预配置的多区域表时，从 JSON 文件加载表的自动缩放设置和副本配置会很有帮助。

要使用以下代码示例，您可以从 auto-scaling.zip 下载示例 JSON 文件，然后提取`auto-scaling.json`和`replication.json`。记下文件的路径。

在此示例中，JSON 文件位于当前目录中。有关不同的文件路径选项，请参阅如何从文件加载参数。

```
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
```

更新多区域表 (CLI) 的预配置容量和 auto Scaling 设置

要更新现有表的预配置模式和 auto Scaling 配置，您可以使用 AWS CLI `update-table`命令。

请注意，您必须使用 Amazon Keyspaces CLI 命令来创建或修改多区域自动扩展设置。这是因为 Amazon Keyspaces 用来代表您自动扩展表容量的服务 Application Auto Scaling 不支持多个。 AWS 区域

更新多区域表的预配置模式或自动扩展设置时，可以更新表每个副本的读取容量设置和读取自动扩展配置。

但是，写入容量在所有副本之间保持同步，以确保有足够的容量跨所有区域复制写入。要更新特定区域中表副本的读取容量，您可以更改表的以下可选参数之一`replicaSpecifications`：

- 预配置的读取容量单位（可选）
- 读取容量的自动缩放设置（可选）

当您使用复杂的自动缩放设置和不同的表副本配置来更新多区域表时，从 JSON 文件加载表的自动缩放设置和副本配置会很有帮助。

要使用以下代码示例，您可以从 auto-scaling.zip 下载示例 JSON 文件，然后提取`auto-scaling.json`和`replication.json`。记下文件的路径。

在此示例中，JSON 文件位于当前目录中。有关不同的文件路径选项，请参阅如何从文件加载参数。

```
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
```

查看多区域表 (CLI) 的预配置容量和 auto Scaling 设置

要查看多区域表的 auto Scaling 配置，您可以使用`get-table-auto-scaling-settings`操作。以下 CLI 命令就是一个示例。

```
aws keyspaces get-table-auto-scaling-settings --keyspace-name mykeyspace --table-name
 mytable
```

您应当看到如下输出。

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

关闭多区域表的自动缩放 (CLI)

您可以使用 AWS CLI `update-table`命令关闭现有表的 auto 缩放。请注意，您无法为单个表副本禁用 auto 缩放。

在以下示例中，针对表的读取容量关闭了 auto Scaling。

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

> **Note**
>
> 要删除 Application Auto Scaling 使用的服务相关角色，必须禁用账户中所有表的自动缩放功能。 AWS 区域

手动设置多区域表的预配置容量 (CLI)

如果您必须关闭多区域表的 auto Scaling，则可以使用`update-table`手动为副本表配置表的读取容量。

```
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**
>
> 我们建议对使用预配置容量的多区域表使用 auto scaling。有关更多信息，请参阅 the section called “多区域表”。

Amazon Keyspaces（Apache Cassandra 兼容）的时间点故障恢复

时间点故障恢复 (PITR) 可连续备份表数据，帮助保护 Amazon Keyspaces 表免受意外写入或删除操作的影响。

例如，假设测试脚本意外写入生产 Amazon Keyspaces 表中。借助时间点故障恢复，您可以将该表的数据还原到过去 35 天内启用 PITR 以来的任何时间点。如果删除了启用时间点故障恢复的表，您可以查询已删除表的 35 天数据（无需额外费用），并将其还原到删除点之前的状态。

您可以使用控制台、AWS SDK 和 AWS Command Line Interface (AWS CLI) 或 Cassandra 查询语言 (CQL) 将 Amazon Keyspaces 表还原到某个时间点。有关更多信息，请参阅 将 Amazon Keyspaces 表还原到某个时间点。

时间点操作对基表的性能或可用性没有影响，还原表也不会消耗额外的吞吐量。

有关 PITR 限额的更多信息，请参阅 限额。

有关定价的信息，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定价。

主题

- Amazon Keyspaces 中 point-in-time 恢复的工作原理
- 将 Amazon Keyspaces 表还原到某个时间点

Amazon Keyspaces 中 point-in-time 恢复的工作原理

本节概述了 Amazon Keyspaces point-in-time 恢复 (PITR) 的工作原理。有关定价的更多信息，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定价。

主题

- 启用 point-in-time 恢复 (PITR)
- 还原表所需的权限
- PITR 连续备份的时段
- PITR 还原设置
- 使用 PITR 还原加密表
- 使用 PITR 还原多区域表

- 使用 PITR 还原表所需的时间
- Amazon Keyspaces PITR 以及与 AWS 服务的集成

启用 point-in-time 恢复 (PITR)

您可以使用控制台启用 PITR，也可以编程方式启用它。

使用控制台启用 PITR

可在自定义设置选项下管理新表的 PITR 设置。默认情况下，通过控制台创建的新表已启用 PITR。

要为现有表启用 PITR，请完成以下步骤：

1. 登录 AWS Management Console并打开 Amazon Keyspaces 控制台：https://console.aws.amazon.com/keyspaces/home。
2. 在导航窗格中，选择表，然后选择要编辑的表。
3. 在备份选项卡上，选择编辑。
4. 在 “编辑 point-in-time 恢复设置” 部分中，选择 “启用 P oint-in-time 恢复”。

您可以随时通过以下步骤禁用表的 PITR。

1. 登录 AWS Management Console并打开 Amazon Keyspaces 控制台：https://console.aws.amazon.com/keyspaces/home。
2. 在导航窗格中，选择表，然后选择要编辑的表。
3. 在备份选项卡上，选择编辑。
4. 在 “编辑 point-in-time 恢复设置” 部分中，清除 “启用 P oint-in-time 恢复” 复选框。

> **Important**
>
> 禁用 PITR 会立即删除您的备份历史记录，即使您在 35 天内为表重新启用 PITR 也是如此。

要了解如何使用控制台还原表，请参阅 the section called “将表还原到某个时间点（控制台）”。

使用 AWS CLI 启用 PITR

您可以使用 `UpdateTable` API 管理表的 PITR 设置。

使用 AWS CLI 创建新表时，必须在创建新表时明确启用 PITR。

要在创建新表时启用 PITR，可以使用以下示例 AWS CLI 命令。为了提高可读性，该命令已分成不同的行。

```
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**
>
> 如果未指定 point-in-time 恢复值，则默认情况下会禁用 point-in-time恢复。

要确认表的 point-in-time 恢复设置，可以使用以下AWS CLI命令。

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

要使用 AWS CLI 为现有表启用 PITR，请运行以下命令。

```
aws keyspaces update-table --keyspace-name 'myKeyspace' --table-name 'myTable' --point-
in-time-recovery 'status=ENABLED'
```

要为现有表禁用 PITR，请运行以下 AWS CLI 命令。

```
aws keyspaces update-table --keyspace-name 'myKeyspace' --table-name 'myTable' --point-
in-time-recovery 'status=DISABLED'
```

> **Important**
>
> 禁用 PITR 会立即删除您的备份历史记录，即使您在 35 天内为表重新启用 PITR 也是如此。

使用 CQL 启用 PITR

您可以使用 `point_in_time_recovery` 自定义属性管理表的 PITR 设置。

使用 CQL 创建新表时，必须在创建新表时明确启用 PITR。

要在创建新表时启用 PITR，可以使用以下示例 CQL 命令。

```
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**
>
> 如果未指定 point-in-time 恢复自定义属性，则默认情况下会禁用 point-in-time恢复。

要使用 CQL 为现有表启用 PITR，请运行以下 CQL 命令。

```
ALTER TABLE mykeyspace.mytable
WITH custom_properties = {'point_in_time_recovery': {'status': 'enabled'}}
```

要为现有表禁用 PITR，请运行以下 CQL 命令。

```
ALTER TABLE mykeyspace.mytable
WITH custom_properties = {'point_in_time_recovery': {'status': 'disabled'}}
```

> **Important**
>
> 禁用 PITR 会立即删除您的备份历史记录，即使您在 35 天内为表重新启用 PITR 也是如此。

有关 CQL 语言参考中的更多信息，请参阅 the section called “CREATE TABLE” 和 the section called “ALTER TABLE”。要了解如何使用 CQL 还原表，请参阅 the section called “使用 CQL 将表还原到某个时间点”。

还原表所需的权限

要成功还原表，IAM 用户或角色需要以下最低权限：

- `cassandra:Restore`：需要执行还原操作才能还原目标表。
- `cassandra:Select`：需要执行选择操作才能从源表中读取数据。
- `cassandra:TagResource`：标签操作是可选的，只有在还原操作添加标签时才需要执行此操作。

以下是向用户授予还原键空间 `mykeyspace` 中的表所需的最低权限的策略示例。

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

根据选定的其他功能，可能需要其他权限才能还原表。例如，如果使用客户自主管理型密钥对源表进行静态加密，则 Amazon Keyspaces 必须有权访问源表的客户自主管理型密钥才能成功还原表。有关更多信息，请参见 the section called “PITR 和加密表”。

如果您使用带有条件键的 IAM 策略来限制特定源的传入流量，则必须确保 Amazon Keyspaces 有权代表您的主体执行还原操作。如果您的策略将传入流量限制为以下任一项，则您必须将 `aws:ViaAWSService` 条件键添加到 IAM 策略：

- 具有 `aws:SourceVpce` 的 VPC 端点
- 使用 `aws:SourceIp` 的 IP 范围
- 具有 `aws:SourceVpc` 的 VPC

`aws:ViaAWSService` 条件键允许在任何 AWS 服务使用主体的凭证发出请求时进行访问。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件键。

以下是将源流量限制到特定 IP 地址并允许 Amazon Keyspaces 代表主体还原表的策略示例。

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

有关使用 `aws:ViaAWSService` 全局条件键的策略示例，请参阅 the section called “VPC 终端节点策略和 Amazon Keyspaces point-in-time 恢复 (PITR)”。

PITR 连续备份的时段

Amazon Keyspaces PITR 使用两个时间戳来维护表中可还原备份的可用时间范围。

- 最早可还原时间：标记最早可还原备份的时间。最早的可还原备份最长可追溯到 35 天或启用 PITR 时（以较晚者为准）。最长备份期限为 35 天，无法修改。

- 当前时间：最新可还原备份的时间戳为当前时间。如果在还原期间未提供时间戳，则使用当前时间。

启用 PITR 后，您可以还原到 `EarliestRestorableDateTime` 和 `CurrentTime` 之间的任意时间点。您只能将表数据还原到启用 PITR 的时间。

如果您禁用 PITR，然后再次将其重新启用，则系统会将第一个可用备份的开始时间重置为重新启用 PITR 的时间。这意味着，禁用 PITR 会删除您的备份历史记录。

> **Note**
>
> 对表的数据定义语言 (DL) 操作（如架构更改）是异步执行的。您只能在还原的表数据中看到已完成的操作，但如果还原时正在进行其他操作，则您可能会在源表上看到这些操作。有关 DDL 语句列表，请参阅 the section called “DDL 语句”。

表不一定要处于活动状态才能还原。如果对已删除的表启用了 PITR，并且删除操作发生在备份时段内（或最近 35 天内），您还可以还原已删除的表。

> **Note**
>
> 如果使用与先前删除的表相同的限定名称（例如 mykeyspace.mytable）创建了新表，则已删除的表将无法再还原。如果您尝试从控制台执行此操作，则系统会显示警告。

PITR 还原设置

当您使用 PITR 还原表时，Amazon Keyspaces 会根据新表中选择的时间戳 (`day:hour:minute:second`) 将源表的架构和数据还原到相应状态。PITR 不会覆盖现有表。

除了表的架构和数据外，PITR 还会从源表中还原 `custom_properties`。表的数据是根据所选时间戳（在最早还原时间和当前时间之间）进行还原的，与此不同的是，自定义属性始终根据截至当前时间的表设置进行还原。

还原后的表的设置与源表的设置以及启动还原时的时间戳相匹配。如果要在还原期间覆盖这些设置，则可以使用 `WITH custom_properties` 来执行此操作。自定义属性包括以下设置：

- 读取/写入容量模式
- 预置的吞吐容量设置
- PITR 设置

如果表处于预置容量模式且启用了自动缩放，则还原操作还会恢复表的自动缩放设置。您可以使用 CQL 中的`autoscaling_settings`参数或 `autoScalingSpecification` CLI 来覆盖它们。有关 auto Scaling 设置的更多信息，请参阅the section called “使用 auto scaling 管理吞吐容量”。

执行完整表还原时，还原表的所有表设置都来自还原时源表的当前设置。

例如，假设一个表的预配置的吞吐量最近下降到 50 个读取容量单位和 50 个写入容量单位。然后，您将表的状态还原到三周前的状态。表在该时间的预置吞吐量为 100 个读取容量单位和 100 个写入容量单位。在这种情况下，Amazon Keyspaces restores 将表数据还原到该时间点，但使用当前预置吞吐量设置（50 个读取容量单位和 50 个写入容量单位）。

以下设置不会还原，您必须手动为新表配置这些设置。

- AWS Identity and Access Management (IAM) 策略
- Amazon CloudWatch 指标和警报
- 标签（可以使用 `WITH TAGS` 将其添加到 CQL `RESTORE` 语句中）

使用 PITR 还原加密表

当您使用 PITR 还原表时，Amazon Keyspaces 会还原源表的加密设置。如果表是使用 AWS 拥有的密钥（默认）加密的，则系统会自动以相同的设置还原表。如果您要还原的表是使用客户自主管理型密钥加密的，那么 Amazon Keyspaces 需要能够访问相同的客户自主管理型密钥才能还原表数据。

您可以在还原时更改表的加密设置。要从 AWS 拥有的密钥更改为客户自主管理型密钥，您需要在还原时提供有效且可访问的客户自主管理型密钥。

如果您想从客户自主管理型密钥更改为 AWS 拥有的密钥，请确认 Amazon Keyspaces 有权访问源表的客户自主管理型密钥以使用 AWS 拥有的密钥还原表。有关表的静态加密设置的更多信息，请参阅the section called “工作原理”。

> **Note**
>
> 如果表是因为 Amazon Keyspaces 无法访问您的客户自主管理型密钥而被删除，则在尝试还原表之前，您需要确保 Amazon Keyspaces 能够访问客户自主管理型密钥。如果 Amazon Keyspaces 无权访问该密钥，则无法还原使用客户自主管理型密钥加密的表。有关更多信息，请参阅《AWS Key Management Service Developer Guide》中的 Troubleshooting key access。

使用 PITR 还原多区域表

您可以使用 PITR 还原多区域表。要使还原操作成功，必须将源表和目标表都复制到同一个 AWS 区域。

Amazon Keyspaces 在作为密钥空间一部分的每个复制区域中恢复源表的设置。您还可以在还原操作期间覆盖设置。有关可在还原期间更改的设置的更多信息，请参阅 the section called “还原设置”。

有关多区域复制的更多信息，请参阅 the section called “工作原理”。

使用 PITR 还原表所需的时间

还原表所需的时间取决于多种因素，并且并不是始终与表的大小直接相关。

以下是还原用时的一些注意事项。

- 将备份还原到新表。执行用于创建新表和启动还原流程的所有操作可能最多需要 20 分钟（即使表是空的）。
- 具有分布良好的数据模型的大型表的还原时间可能是几个小时或更长时间。
- 如果源表包含严重偏斜的数据，则还原时间可能会增加。例如，如果表的主键使用一年中的月份作为分区键，而您的所有数据均来自 12 月，那么就出现了偏斜数据。

规划灾难恢复的最佳做法是定期记录平均还原完成时间，并确定这些时间对整个恢复时间目标的影响。

Amazon Keyspaces PITR 以及与 AWS 服务的集成

使用 AWS CloudTrail 记录以下 PITR 操作，以实现持续监控和审计。

- 创建启用或禁用 PITR 的新表。
- 为现有表启用或禁用 PITR。
- 还原活动或已删除的表。

有关更多信息，请参见 使用 AWS CloudTrail 记录 Amazon Keyspaces API 调用。

您可以使用 AWS CloudFormation 执行以下 PITR 操作。

- 创建启用或禁用 PITR 的新表。

- 为现有表启用或禁用 PITR。

有关更多信息，请参阅《AWS CloudFormation 用户指南》中的 Cassandra 资源类型参考。

将 Amazon Keyspaces 表还原到某个时间点

Amazon Keyspaces（Apache Cassandra 兼容）时间点故障恢复 (PITR) 让您能够将 Amazon Keyspaces 表数据还原到最近 35 天中的任何时间点。本教程的第一部分介绍了如何使用 Amazon Keyspaces 控制台、AWS Command Line Interface (AWS CLI) 和 Cassandra 查询语言 (CQL) 将表还原到某个时间点。第二部分介绍了如何使用 AWS CLI 和 CQL 还原已删除的表。

主题

开始前的准备工作

您必须为用户配置相应的权限才能还原 Amazon Keyspaces 表（如果尚未执行此操作）。在 AWS Identity and Access Management (IAM) 中，AWS 托管式策略 `AmazonKeyspacesFullAccess` 包括还原 Amazon Keyspaces 表所需的权限。有关实施包含所需最低权限的策略的详细步骤，请参阅 the section called “还原权限。”。

将表还原到某个时间点（控制台）

以下示例演示了如何使用 Amazon Keyspaces 控制台将名为 `mytable` 的现有表还原到某个时间点。

> **Note**
>
> 此过程假定您已启用时间点故障恢复。要为 `mytable` 表启用 PITR，请按照 the section called “使用 控制台” 中的步骤操作。

1. 登录 AWS Management Console并打开 Amazon Keyspaces 控制台：https://console.aws.amazon.com/keyspaces/home。
2. 在控制台左侧的导航窗格中，选择 Tables (表)。
3. 在表的列表中，选择 `mytable` 表。
4. 在 `mytable` 表的备份选项卡的时间点故障恢复部分，选择还原。
5. 对于新表名称，输入 **`mytable_restored`**。
6. 要定义还原操作的时间点，可以在两个选项之间进行选择：
   - 选择预配置的最早时间。
   - 选择指定日期和时间并输入要将新表还原到的日期和时间。

   > **Note**
   >
   > 您可以还原到最早时间和当前时间之间的任何时间点。Amazon Keyspaces 会根据所选日期和时间 (day:hour:minute:second) 将表数据还原到相应状态。

7. 选择还原，启动还原过程。

   正在还原的表显示状态为 Restoring (正在还原)。还原过程完成后，`mytable_restored` 表的状态更改为 Active (活动)。

   > **Important**
   >
   > 正在还原时，请勿修改或删除授予 IAM 实体（例如，用户、组或角色）执行还原的权限的 AWS Identity and Access Management (IAM) 策略。否则，可能会出现意外行为。例如，假设您在还原表时删除了对该表的写入权限。在这种情况下，底层的 `RestoreTableToPointInTime` 操作将无法向该表中写入任何还原的数据。
   > 您只能在还原操作完成之后修改或删除权限。

使用 AWS CLI 将表还原到某个时间点

以下过程演示如何使用 AWS CLI 将名为 `myTable` 的现有表还原到某个时间点。

1. 在第一步中，您将创建一个名为 `myTable` 且启用了 PITR 的简单表。为了便于阅读，该命令已分成不同的行。

```
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. 确认新表的属性并查看 PITR 的 `earliestRestorableTimestamp`。

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

该命令的输出返回了以下内容。

```
{
    "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": ""
    }
}
```

您可以每隔一秒钟将活动表还原到 earliestRestorableTimestamp 对应的时间和当前时间之间的任意时间点。默认值为当前时间。

3. 要将表还原到某个时间点，请指定 ISO 8601 格式的 restore_timestamp。您可以选择最近 35 天中的任何时间点，时间间隔为 1 秒。例如，以下命令使表还原到 EarliestRestorableDateTime。

```
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"
```

此命令的输出会返回已还原的表的 ARN。

```
{
    "restoredTableARN": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/
myKeyspace/table/myTable_restored"
}
```

要将该表还原到当前时间，可以省略 restore-timestamp。

```
aws keyspaces restore-table --source-keyspace-name 'myKeyspace' --source-
table-name 'myTable' --target-keyspace-name 'myKeyspace' --target-table-name
 'myTable_restored1'"
```

> **Important**
>
> 正在还原时，请勿修改或删除授予 IAM 实体（例如，用户、组或角色）执行还原的权限的 AWS Identity and Access Management (IAM) 策略。否则，可能会出现意外行为。例如，假设您在还原表时删除了对该表的写入权限。在这种情况下，底层的 `RestoreTableToPointInTime` 操作将无法向该表中写入任何还原的数据。您只能在还原操作完成之后修改或删除权限。

使用 CQL 将表还原到某个时间点

以下过程展示了如何使用 CQL 将名为 `mytable` 的现有表还原到某个时间点。

> **Note**
>
> 此过程假定您已启用时间点故障恢复。要为表启用 PITR，请按照 the section called “CQL” 中的步骤操作。

1. 您可以将活动表还原到 `earliest_restorable_timestamp` 对应的时间和当前时间之间的某个时间点。默认值为当前时间。

   要确认是否为 `mytable` 表启用了时间点故障恢复，请按如下方式查询 `system_schema_mcs.tables`。

   ```
   SELECT custom_properties
   FROM system_schema_mcs.tables
   WHERE keyspace_name = 'mykeyspace' AND table_name = 'mytable';
   ```

   时间点故障恢复处于启用状态，如以下示例输出所示。

   ```
   custom_properties
   -----------------
   {
     ...,
       "point_in_time_recovery": {
       "earliest_restorable_timestamp":"2020-06-30T19:19:21.175Z"
       "status":"enabled"
     }
   ```

```
}
```

2. 将表还原到由 ISO 8601 格式的 `restore_timestamp` 指定的时间点。在这种情况下，`mytable` 表会还原到当前时间。您可以省略 `WITH restore_timestamp = ...` 子句。如果没有该子句，系统将使用当前时间戳。

   ```
   RESTORE TABLE mykeyspace.mytable_restored
   FROM TABLE mykeyspace.mytable;
   ```

   您还可以还原到特定时间点。您可以指定最近 35 天内的任何时间点。例如，以下命令使表还原到 `EarliestRestorableDateTime`。

   ```
   RESTORE TABLE mykeyspace.mytable_restored
   FROM TABLE mykeyspace.mytable
   WITH restore_timestamp = '2020-06-30T19:19:21.175Z';
   ```

   有关完整的语法描述，请参阅语言参考中的 the section called “RESTORE TABLE”。

要验证表的还原是否成功，请查询 `system_schema_mcs.tables`，确认表的状态。

```
SELECT status
FROM system_schema_mcs.tables
WHERE keyspace_name = 'mykeyspace' AND table_name = 'mytable_restored'
```

该查询会显示以下输出。

```
status
------
RESTORING
```

正在还原的表显示状态为 Restoring (正在还原)。还原过程完成后，`mytable_restored` 表的状态更改为 Active (活动)。

> **Important**
>
> 正在还原时，请勿修改或删除授予 IAM 实体（例如，用户、组或角色）执行还原的权限的 AWS Identity and Access Management (IAM) 策略。否则，可能会出现意外行为。例如，假设您在还原表时删除了对该表的写入权限。在这种情况下，底层的 `RestoreTableToPointInTime` 操作将无法向该表中写入任何还原的数据。

您只能在还原操作完成之后修改或删除权限。

使用 AWS CLI 还原已删除的表

以下过程展示了如何使用 AWS CLI 将名为 myTable 的已删除表还原到删除时的状态。

> **Note**
>
> 此过程假定您已为删除的表启用 PITR。

1. 删除您在前面教程中创建的表。

```
aws keyspaces delete-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

2. 使用以下命令将已删除的表还原到删除时的状态。

```
aws keyspaces restore-table --source-keyspace-name 'myKeyspace' --source-
table-name 'myTable' --target-keyspace-name 'myKeyspace' --target-table-name
 'myTable_restored2'
```

此命令的输出会返回已还原的表的 ARN。

```
{
    "restoredTableARN": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/
myKeyspace/table/myTable_restored2"
}
```

使用 CQL 还原已删除的表

以下过程展示了如何使用 CQL 将名为 mytable 的已删除表还原到删除时的状态。

> **Note**
>
> 此过程假定您已为删除的表启用 PITR。

1. 要确认是否为已删除表启用时间点故障恢复，请查询系统表。仅显示已启用时间点故障恢复的表。

```
SELECT custom_properties
FROM system_schema_mcs.tables_history
WHERE keyspace_name = 'mykeyspace' AND table_name = 'my_table';
```

该查询会显示以下输出。

```
custom_properties
------------------
{
    ...,
   "point_in_time_recovery":{
      "restorable_until_time":"2020-08-04T00:48:58.381Z",
      "status":"enabled"
   }
}
```

2. 使用以下示例语句将表还原到删除时的状态。

```
RESTORE TABLE mykeyspace.mytable_restored
FROM TABLE mykeyspace.mytable;
```

使用 Amazon Keyspaces 生存时间 (TTL) 让数据过期

Amazon Keyspaces（Apache Cassandra 兼容）生存时间 (TTL) 可自动让表中的数据过期，帮助您简化应用程序逻辑并优化存储价格。系统会根据您设置的“存活时间”值自动从表中删除不再需要的数据。这样就能更容易地遵守基于业务、行业或监管要求的数据留存政策，这些政策规定了数据需要保留的时间或必须删除数据的时间。

例如，您可以在 AdTech 应用程序中使用 TTL 来安排特定广告的数据何时过期，也就是对客户不再可见。您还可以使用 TTL 自动停用旧数据，节省存储成本。您可以为整个表设置默认 TTL 值，也可以为单个行和列覆盖该值。TTL 操作不会影响应用程序的性能。此外，用 TTL 标记为过期的行和列的数量不会影响表的可用性。

Amazon Keyspaces 会自动筛选出过期的数据，这样查询结果中就不会返回过期的数据，也不会在数据操作语言 (DML) 语句中使用。Amazon Keyspaces 通常会在过期日期后的 10 天内从存储空间中删除过期的数据。在极少数情况下，如果底层存储分区持续存在活动，Amazon Keyspaces 可能无法在 10 天内删除数据，以保护可用性。对于这些情况，一旦分区上的流量减少，Amazon Keyspaces 会继续尝试删除过期数据。从存储中永久删除数据后，就不再产生存储费用。有关更多信息，请参阅the section called “工作原理”。

您可以使用控制台或 Cassandra 查询语言 (CQL) 设置、修改或禁用新表和现有表的默认 TTL 设置。在配置了默认 TTL 的表上，您可以使用 Cassandra 查询语言 (CQL) 来覆盖默认 TTL 设置，并将自定义 TTL 值应用于行和列。有关更多信息，请参阅the section called “如何使用生存时间”。

TTL 定价基于使用“生存时间”删除或更新的行的大小。TTL 操作的计量单位是 TTL deletes。每删除或更新一行数据，将消耗每 KB 数据一次 TTL 删除操作。例如，要更新一个存储了 2.5 KB 数据的行并同时删除该行内的一个或多个列，将需要三次 TTL 删除操作。或者，要删除一个包含 3.5 KB 数据的完整行，将需要四次 TTL 删除操作。每删除一行中的每 KB 数据，将消耗一次 TTL 删除操作。有关定价的更多信息，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定价。

主题

工作原理：Amazon Keyspaces 生存时间 (TTL)

Amazon Keyspaces 生存时间 (TTL) 是完全托管式的。您不必管理诸如压缩策略之类的低级系统设置。数据会在您指定的时间过期，Amazon Keyspaces 会自动删除过期数据（通常在 10 天内），而不会影响您的应用程序性能或可用性。

过期数据会标记为已删除，且无法用于数据操作语言 (DML) 语句。在继续对包含过期数据的行执行读取和写入操作时，过期数据将继续计入读取容量单位 (RCU) 和写入容量单位 (WCU)，直至从存储中删除。

主题

为表设置默认 TTL 值

在 Amazon Keyspaces 中，您可以在创建表时为表中的所有行设置默认 TTL 值。您也可以编辑现有表格，以设置或更改插入到表中的新行的默认 TTL 值。更改表的默认 TTL 值不会修改表中任何现有数据的 TTL 值。表的默认 TTL 值为零，表示数据不会自动过期。如果表的默认 TTL 值大于零，则会为每行添加一个过期时间戳。

每次数据更新时，Amazon Keyspaces 都会计算一个新的 TTL 时间戳。TTL 值以秒为单位设置，最大可配置值为 630,720,000 秒，相当于 20 年。有关如何使用 AWS Management Console 或 CQL 设置、修改和禁用表的默认 TTL 值的更多信息，请参阅the section called “如何使用生存时间”。

为行和列设置自定义 TTL 值

> **Note**
>
> 在为行和列设置自定义 TTL 值之前，必须先在表上启用 TTL。有关更多信息，请参阅the section called “如何使用自定义属性对现有表启用生存时间 (TTL)”。

要覆盖表的默认 TTL 值或为各行设置过期日期，可以使用以下 CQL 数据操作语言 (DML) 语句：

- `INSERT` – 用于插入设置了 TTL 值的新数据行。

- UPDATE – 用于使用新的 TTL 值修改现有数据行。

为行设置 TTL 值优先于表的默认 TTL 设置。

有关 CQL 语法和示例，请参阅the section called “使用 CQL 通过 INSERT 编辑自定义生存时间 (TTL) 设置”。

要覆盖或设置单个列的 TTL 值，可以使用以下 CQL DML 语句更新现有行中列子集的 TTL 设置：

- UPDATE – 用于更新一列数据。

为列设置 TTL 值优先于表的默认 TTL 设置和该行的任何自定义 TTL 设置。有关 CQL 语法和示例，请参阅the section called “使用 CQL 通过 UPDATE 编辑自定义生存时间 (TTL) 设置”。

在表上启用 TTL

当您在 CREATE TABLE 或 ALTER TABLE 语句中指定大于 0 的 default_time_to_live 值时，会自动为表启用 TTL。如果您没有为表指定 default_time_to_live，但要使用 INSERT 或 UPDATE 操作为行或列指定自定义 TTL 值，则必须先为该表启用 TTL。您可以使用 ttl 自定义属性为表启用 TTL。

当您在表上启用 TTL 时，Amazon Keyspaces 会开始为每行存储其他与 TTL 相关的元数据。此外，TTL 使用过期时间戳来跟踪行或列的过期时间。时间戳存储为行的元数据，并占该行的存储成本。

TTL 功能启用后，无法为表禁用它。将表的 default_time_to_live 设置为 0 会禁用新数据的默认过期时间，但不会停用 TTL 功能，也不会将表恢复到原来的 Amazon Keyspaces 存储元数据或写入行为。

Amazon Keyspaces 生存时间以及与 AWS 服务的集成

Amazon CloudWatch 中提供了以下 TTL 指标，用于实现持续监控。

- TTLDeletes – 使用生存时间 (TTL) 删除或更新一行数据所消耗的单位。

有关如何使用 CloudWatch 指标的更多信息，请参阅the section called “使用监控 CloudWatch”。

使用 AWS CloudFormation 时，您可以在创建 Amazon Keyspaces 表时启用 TTL。有关更多信息，请参阅 AWS CloudFormation 用户指南。

如何使用生存时间 (TL)

您可以使用 Amazon Keyspaces（Apache Cassandra 兼容）控制台或 CQL 启用、更新和禁用“生存时间”设置。

主题

创建启用了默认生存时间 (TTL) 设置的新表（控制台）

按照以下步骤，使用 Amazon Keyspaces 控制台创建启用了生存时间设置的新表。

1. 登录 https://console.aws.amazon.com/keyspaces/home，然后打开 AWS Management Console Amazon Keyspaces 控制台。
2. 在导航窗格中，选择 Tables (表)，然后选择 Create table (创建表)。
3. 在创建表页面的表详细信息部分中，选择一个键空间并为新表提供一个名称。
4. 在架构部分，为您的表创建架构。
5. 在表设置部分，选择自定义设置。
6. 继续设置生存时间 (TTL)。

   在此步骤中，您为表选择默认 TTL 设置。

   对于默认 TTL 周期，输入到期时间并选择您输入的时间单位，例如秒、天或年。Amazon Keyspaces 将在几秒钟内存储值。
7. 选择 Create Table（创建表）。系统将使用使用指定的默认 TTL 值创建您的表。

> **Note**
>
> 通过使用 CQL 编辑器中的数据操作语言 (DML)，可以覆盖特定行或列的表默认 TTL 设置。

更新现有表的默认生存时间 (TTL) 设置（控制台）

按照以下步骤，使用 Amazon Keyspaces 控制台更新现有表的生存时间设置。

1. 登录 https://console.aws.amazon.com/keyspaces/home，然后打开 AWS Management Console Amazon Keyspaces 控制台。
2. 选择要更新的表，然后选择其他设置选项卡。
3. 继续选择生存时间 (TTL)，然后选择编辑。
4. 对于默认 TTL 周期，输入到期时间并选择您输入的时间单位，例如秒、天或年。Amazon Keyspaces 将在几秒钟内存储值。这不会更改现有行的 TTL 值。
5. 定义 TTL 设置后，选择保存更改。

禁用现有表的默认生存时间 (TTL) 设置（控制台）

按照以下步骤使用 Amazon Key AWS Management Console禁用现有表的生存时间设置。

1. 登录 https://console.aws.amazon.com/keyspaces/home，然后打开 AWS Management Console Amazon Keyspaces 控制台。
2. 选择要更新的表，然后选择其他设置选项卡。
3. 继续选择生存时间 (TTL)，然后选择编辑。
4. 选择默认 TTL 周期，并将该值设置为零。这会禁用表中未来数据的默认 TTL。它不会更改现有行的 TTL 值。
5. 定义 TTL 设置后，选择保存更改。

使用 CQL 创建启用了默认生存时间 (TTL) 设置的新表

在创建新表时启用 TTL，默认 TTL 值设置为 3,024,000 秒（代表 35 天）。

```
CREATE TABLE my_table (
                userid uuid,
                time timeuuid,
```

```
                subject text,
                body text,
                user inet,
                PRIMARY KEY (userid, time)
                ) WITH default_time_to_live = 3024000;
```

要确认新表的 TTL 设置，请使用 cqlsh describe 语句，如以下示例所示。输出将该表的默认 TTL 设置显示为 default_time_to_live。

```
describe my_table;
```

使用 CQL 通过 ALTER TABLE 编辑默认生存时间 (TTL) 设置

将现有表的 TTL 设置更新为 2,592,000 秒，即 30 天。

```
ALTER TABLE my_table WITH default_time_to_live = 2592000;
```

要确认更新后的表的 TTL 设置，请使用的 cqlsh describe 语句，如以下示例所示。输出将该表的默认 TTL 设置显示为 default_time_to_live。

```
describe my_table;
```

如何使用自定义属性在新表上启用生存时间 (TTL)

要在不对整个表启用 TTL 默认设置的情况下启用可应用于行和列的生存时间自定义设置，可以使用以下 CQL 语句。

```
CREATE TABLE my_keyspace.my_table (id int primary key) WITH CUSTOM_PROPERTIES={'ttl':
{'status': 'enabled'}};
```

为表启用 ttl 后便无法禁用该设置。

如何使用自定义属性对现有表启用生存时间 (TTL)

要在不对整个表启用 TTL 默认设置的情况下启用可应用于行和列的生存时间自定义设置，可以使用以下 CQL 语句。

```
ALTER TABLE my_table WITH CUSTOM_PROPERTIES={'ttl':{'status': 'enabled'}};
```

为表启用 ttl 后便无法禁用该设置。

使用 CQL 通过 INSERT 编辑自定义生存时间 (TTL) 设置

以下 CQL 语句在表中插入一行数据，并将默认 TTL 设置更改为 259,200 秒（相当于 3 天）。

```
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;
```

要确认插入行的 TTL 设置，请使用以下语句。

```
SELECT TTL (subject) from my_table;
```

使用 CQL 通过 UPDATE 编辑自定义生存时间 (TTL) 设置

要将前面插入的 subject 列的 TTL 设置从 259,200 秒（3 天）改为 86,400 秒（1 天），请使用以下语句。

```
UPDATE my_table USING TTL 86400 set subject = 'Updated Message' WHERE userid =
 B79CB3BA-745E-5D9A-8903-4A02327A7E09 and time = 96a29100-5e25-11ec-90d7-b5d91eceda0a;
```

您可以运行一个简单的选择查询，查看到期时间之前更新的记录。

```
SELECT * from my_table;
```

该查询会显示以下输出。

```
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
```

要确认过期已成功，请在配置的过期时间后再次运行相同的查询。

```
SELECT * from my_table;
```

在 subject 列过期后，该查询会显示以下输出。

```
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
```

在 Amazon Keyspaces 中使用客户端时间戳

在 Amazon Keyspaces 中，客户端时间戳是与 Cassandra 兼容的时间戳，表中的每个单元格都会保留这些时间戳。您可以让您的客户端应用程序确定写入顺序，从而使用客户端时间戳来解决冲突。例如，当全球分布式应用程序的客户端对相同的数据进行更新时，客户端时间戳将保留在客户端上进行更新的顺序。Amazon Keyspaces 使用这些时间戳来处理写入操作。有关更多信息，请参阅the section called “工作原理”。

为表打开客户端时间戳后，您可以在 Data Manipulation Language (DML) CQL 查询中使用 `USING TIMESTAMP` 子句指定时间戳。如果您未在 CQL 查询中指定时间戳，Amazon Keyspaces 将使用您的客户端驱动程序传递的时间戳。如果客户端驱动程序不提供时间戳，Amazon Keyspaces 会自动分配单元格级别的时间戳。要查询时间戳，可以在 DML 语句中使用 `WRITETIME` 函数。有关更多信息，请参阅the section called “如何使用客户端时间戳”。

Amazon Keyspaces 不会针对打开客户端时间戳收取额外费用。但是，使用客户端时间戳，您可能为行中的每个值存储和写入其他数据。这可能会导致额外的存储使用量，在某些情况下还会导致吞吐量使用量增加。要了解有关估计对行大小的影响的更多信息，请参阅 the section called “工作原理”。有关 Amazon Keyspaces 定价的更多信息，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定价。

主题

- 客户端时间戳在 Amazon Keyspaces 中的工作原理
- 在 Amazon Keyspaces 中使用客户端时间戳

客户端时间戳在 Amazon Keyspaces 中的工作原理

Amazon Keyspaces 客户端时间戳是完全托管的。您不必管理诸如清理和压缩策略之类的低级系统设置。

删除数据时，会使用 Tombstone 标记行来进行删除。Amazon Keyspaces 会自动移除 Tombstoned 数据（通常在 10 天内），而不会影响您的应用程序性能或可用性。Tombstoned 数据不可用于 Data Manipulation Language (DML) 语句。在继续对包含 Tombstoned 数据的行执行读写操作时，Tombstoned 数据将继续计入存储、读取容量单位 (RCU) 和写入容量单位 (WCU)，直至从存储中删除。

主题

- 客户端时间戳在 Amazon Keyspaces 中的工作原理

- Amazon Keyspaces 客户端时间戳以及与 AWS 服务的集成

客户端时间戳在 Amazon Keyspaces 中的工作原理

在 Amazon Keyspaces 中打开客户端时间戳后，每行的每一列都存储一个时间戳。这些时间戳大约占用 20-40 字节（取决于您的数据），并会增加该行的存储和吞吐量成本。这些元数据字节也计入您的 1 MB 行大小配额。要确定存储空间的总体增加情况（确保行大小保持在 1 MB 以内），请考虑表中的列数和每行中集合元素的数量。例如，如果一个表有 20 列，每列存储 40 字节的数据，则该行的大小将从 800 字节增加到 1200 字节。有关如何估计行大小的更多信息，请参阅the section called “计算行大小”。除了额外的 400 字节存储空间外，在本示例中，每次写入消耗的写入容量单位 (WCU) 数量从 1 个 WCU 增加到 2 个 WCU。有关如何计算读取和写入容量的更多信息，请参阅 the section called “读/写容量模式”。

为表打开客户端时间戳后，您无法将其关闭。此外，时间戳不能是 NULL，如果 CQL 语句或客户端驱动程序未提供客户端时间戳，则系统会自动添加由 Amazon Keyspaces 生成的时间戳。

Amazon Keyspaces 客户端时间戳以及与 AWS 服务的集成

Amazon CloudWatch 中提供了以下客户端时间戳指标，用于实现持续监控。

- `SystemReconciliationDeletes`：删除 Tombstoned 数据所需的删除操作数。

有关如何使用 CloudWatch 指标的更多信息，请参阅 the section called “使用监控 CloudWatch”。

在 Amazon Keyspaces 中使用客户端时间戳

您可以使用 Amazon Keyspaces（Apache Cassandra 兼容）控制台、Cassandra Query Language (CQL)、AWS SDK 和 AWS Command Line Interface (AWS CLI) 来打开客户端时间戳。本节提供了有关如何在新表和现有表上打开客户端时间戳以及如何在查询中使用客户端时间戳的示例。有关该 API 的更多信息，请参阅 Amazon Keyspaces API Reference。

> **Important**
>
> 客户端时间戳无法关闭。打开客户端时间戳是一次性更改。Amazon Keyspaces 没有在不删除表的情况下关闭客户端时间戳的选项。

主题

创建打开客户端时间戳的新表（控制台）

按照以下步骤使用 Amazon Keyspaces 控制台创建打开客户端时间戳的新表。

创建打开客户端时间戳的新表（控制台）

1. 登录 AWS Management Console并打开 Amazon Keyspaces 控制台：https://console.aws.amazon.com/msk/home。
2. 在导航窗格中，选择 Tables (表)，然后选择 Create table (创建表)。
3. 在创建表页面的表详细信息部分中，选择一个键空间并为新表提供一个名称。
4. 在架构部分，为您的表创建架构。
5. 在表设置部分，选择自定义设置。
6. 继续查看客户端时间戳。

   选择打开客户端时间戳，为表打开客户端时间戳。
7. 选择 Create Table（创建表）。您的表是在打开客户端时间戳的情况下创建的。

在现有表上打开客户端时间戳（控制台）

按照以下步骤使用 Amazon Keyspaces AWS Management Console为现有表打开客户端时间戳。

为现有表打开客户端时间戳（控制台）

1. 登录 AWS Management Console并打开 Amazon Keyspaces 控制台：https://console.aws.amazon.com/msk/home。
2. 选择要更新的表，然后选择其他设置选项卡。

3. 在其他设置选项卡上，前往修改客户端时间戳，然后选择打开客户端时间戳
4. 选择保存更改以更改表的设置。

创建打开客户端时间戳的新表 (CQL)

要在创建新表时打开客户端时间戳，可以使用以下 CQL 语句。

```
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'}};
```

要确认新表的客户端时间戳设置，请使用 SELECT 语句查看 custom_properties，如以下示例所示。

```
SELECT custom_properties from system_schema_mcs.tables where keyspace_name =
 'my_keyspace' and table_name = 'my_table';
```

此语句的输出显示了客户端时间戳的状态。

```
'client_side_timestamps': {'status': 'enabled'}
```

使用 ALTER TABLE 为现有表打开客户端时间戳 (CQL)

要为现有表打开客户端时间戳，可以使用以下 CQL 语句。

```
ALTER TABLE my_table WITH custom_properties = {'client_side_timestamps': {'status':
 'enabled'}};;
```

要确认新表的客户端时间戳设置，请使用 SELECT 语句查看 custom_properties，如以下示例所示。

```
SELECT custom_properties from system_schema_mcs.tables where keyspace_name =
 'my_keyspace' and table_name = 'my_table';
```

此语句的输出显示了客户端时间戳的状态。

```
'client_side_timestamps': {'status': 'enabled'}
```

创建打开客户端时间戳的新表 (CLI)

要在创建新表时打开客户端时间戳，可以使用以下 CLI 语句。

```
./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}]'
```

要确认新表的客户端时间戳已打开，请运行以下代码。

```
./aws keyspaces get-table \
--keyspace-name my_keyspace \
--table-name my_table
```

输出应类似于以下示例：

```
{
    "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": ""
    }
}
```

在现有表上打开客户端时间戳 (CLI)

要使用 CLI 为现有表打开客户端时间戳，可以使用以下代码。

```
./aws keyspaces update-table \
--keyspace-name my_keyspace \
--table-name my_table \
--client-side-timestamps 'status=ENABLED'
```

要确认表的客户端时间戳已打开，请运行以下代码。

```
./aws keyspaces get-table \
--keyspace-name my_keyspace \
--table-name my_table
```

输出应类似于以下示例：

```
{
    "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": ""
    }
}
```

在 Data Manipulation Language (DML) 语句中使用客户端时间戳

打开客户端时间戳后，您可以在 INSERT、UPDATE 和 DELETE 语句中使用 USING TIMESTAMP 子句传递时间戳。时间戳值是一个 bigint 值，表示自标准基本时间（称为 epoch）以来的微秒数：1970 年 1 月 1 日 00:00:00 GMT。客户端提供的时间戳必须介于当前挂钟时间的过去 2 天和未来 5 分钟之间。Amazon Keyspaces 会在数据的生命周期内保留时间戳元数据。您可以使用 WRITETIME 函数来查找过去几年发生的时间戳。有关 CQL 语法的更多信息，请参阅 the section called “DML 语句”。

以下 CQL 语句是如何使用时间戳作为 update_parameter 的示例。

```
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;
```

如果您未在 CQL 查询中指定时间戳，Amazon Keyspaces 将使用您的客户端驱动程序传递的时间戳。如果客户端驱动程序未提供时间戳，Amazon Keyspaces 会为您的写入操作分配服务器端时间戳。

要查看为特定列存储的时间戳值，可以在 SELECT 语句中使用 WRITETIME 函数，如以下示例所示。

```
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;
```

使用 AWS CloudFormation 创建 Amazon Keyspaces 资源

Amazon Keyspaces（Apache Cassandra 兼容）与 AWS CloudFormation 集成，后者是一项服务，可帮助您对 AWS 资源进行建模和设置，这样您只需花较少的时间来创建和管理资源与基础设施。您创建一个描述您所需的所有 AWS 资源（如键空间和表）的模板，AWS CloudFormation 将负责为您设置和配置这些资源。

在您使用 AWS CloudFormation 时，可重复使用您的模板来不断地重复设置您的 Amazon Keyspaces 资源。描述您的资源一次，然后在多个 AWS 账户 和区域中反复预调配相同的资源。

Amazon Keyspaces 和 AWS CloudFormation 模板

要为 Amazon Keyspaces 预置和配置资源，您必须了解 AWS CloudFormation 模板。模板是 JSON 或 YAML 格式的文本文件。这些模板描述要在 AWS CloudFormation 堆栈中调配的资源。如果您不熟悉 JSON 或 YAML，可以在 AWS CloudFormation Designer 的帮助下开始使用 AWS CloudFormation 模板。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的什么是 AWS CloudFormation Designer？。

Amazon Keyspaces 支持在 AWS CloudFormation 中创建键空间和表。对于您使用 AWS CloudFormation 模板创建的表，您可以指定架构、读/写模式和预置的吞吐量设置。有关更多信息（包括键空间和表的 JSON 和 YAML 模板示例），请参阅《AWS CloudFormation 用户指南》中的 Cassandra resource type reference。

了解有关 AWS CloudFormation 的更多信息

要了解有关 AWS CloudFormation 的更多信息，请参阅以下资源：

- AWS CloudFormation
- AWS CloudFormation 用户指南
- AWS CloudFormation command line interface User Guide

监控 Amazon Keyspaces（Apache Cassandra 兼容）

监控是保持 Amazon Keyspaces 和您的其他 AWS 解决方案的可靠性、可用性和性能的重要方面。AWS 提供以下监控工具来监视 Amazon Keyspaces，在出现问题时进行报告，并在适当的时候采取自动措施：

- Amazon Keyspaces 在AWS Management Console中提供了一个预配置的控制面板，用于显示账户中所有表中汇总的延迟和错误。
- Amazon CloudWatch 实时监控您的 AWS 资源以及在 AWS 上运行的应用程序。您可以使用自定义的控制面板收集并跟踪指标。例如，您可以通过在不同时间和不同负载条件下衡量性能来为您的环境中的正常 Amazon Keyspaces 性能创建基准。在监控 Amazon Keyspaces 时，应存储历史监控数据，以便将此数据与当前性能数据进行比较，确定正常性能模式和性能异常，并设计解决问题的方法。要建立基准，您至少应监控系统错误。有关更多信息，请参阅 Amazon CloudWatch 用户指南。
- Amazon CloudWatch 警报按您指定的时间段监控单个指标，并根据相对于给定阈值的指标值在若干时间段内执行一项或多项操作。例如，如果您在预置模式下将 Amazon Keyspaces 与 Application Auto Scaling 结合使用，则该操作是由 Amazon Simple Notification Service (Amazon SNS) 发送的通知，用于评估 Application Auto Scaling 策略。

  CloudWatch 告警将不会调用操作，因为这些操作处于特定状态。该状态必须已改变并在指定的若干个时间段内保持不变。有关更多信息，请参阅 使用亚马逊监控亚马逊密钥空间 CloudWatch。
- Amazon CloudWatch Logs 使您能够监控、存储和访问来自 Amazon Keyspaces 表、CloudTrail 和其他来源的日志文件。CloudWatch Logs 可以监控日志文件中的信息，并在达到特定阈值时通知您。您还可以在高持久性存储中检索您的日志数据。有关更多信息，请参阅 Amazon CloudWatch Logs 用户指南。
- AWS CloudTrail 捕获由您的 AWS 账户 或代表该账户发出的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 存储桶。您可以标识哪些用户和账户调用了 AWS、从中发出调用的源 IP 地址以及调用的发生时间。有关更多信息，请参阅 AWS CloudTrail 用户指南。

Amazon EventBridge 是一种无服务器事件总线服务，可以轻松地将应用程序与来自各种来源的数据相连接。EventBridge 可以从您自己的应用程序、软件即服务 (SaaS) 应用程序和AWS服务传输实时数据流，然后将该数据路由到诸如 Lambda 之类的目标。这使您能够监控服务中发生的事件，并构建事件驱动的架构。有关更多信息，请参阅 Amazon EventBridge 用户指南。

主题

- 使用 AWS CloudTrail 记录 Amazon Keyspaces API 调用

使用亚马逊监控亚马逊密钥空间 CloudWatch

您可以使用 Amazon 监控 Amazon Keyspaces CloudWatch，亚马逊会收集原始数据并将其处理为可读的近乎实时的指标。这些统计数据会保存 15 个月，从而使您能够访问历史信息，并能够更好地了解您的 Web 应用程序或服务的执行情况。

此外，可以设置用于监测特定阈值的警报，并在达到相应阈值时发送通知或执行操作。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

> **Note**
>
> 要快速开始使用显示 Amazon Keyspaces 常用指标的预配置 CloudWatch 控制面板，您可以使用提供的 AWS CloudFormation 模板。https://github.com/aws-samples/amazon-keyspaces-cloudwatch-cloudformation-templates

主题

- 如何使用 Amazon Keyspaces 指标？
- Amazon Keyspaces 指标与维度
- 创建 CloudWatch 警报以监控 Amazon Keyspaces

如何使用 Amazon Keyspaces 指标？

Amazon Keyspaces 报告的指标为您提供了可通过不同方式分析的信息。下面的列表显示这些指标的一些常见用途。这些是入门建议，并不全面。有关指标和保留的更多信息，请参阅 Metrics。

我如何？	相关指标
如何确定是否发生了任何系统错误？	您可以监控 `SystemErrors` ，以确定是否有任何请求导致了服务器错误代码。通常，此指标应等于零。如果不是，您可能需要调查。
如何比较平均预置读取容量与使用的读取容量？	监控平均预置读取容量和使用的读取容量

我如何？	相关指标
	1. 将 ConsumedReadCapacityUnits 和 ProvisionedReadCapacityUnits 的周期设置为要监控的时间间隔。 2. 将 ConsumedReadCapacityUnits 的统计从 Average 更改为 Sum。 3. 新建一个空的 Math expression (数学表达式)。 4. 在新数学表达式的详细信息部分，输入指标的 ID，然后将指标除以该指标的**ConsumedReadCapacityUnits** CloudWatch 周期函数 (metric_id/ (PERIOD (metric_id))。 5. 取消选择 ConsumedReadCapacityUnits 。 现在，您可以将使用的平均读取容量与预置容量进行比较。有关基本算术函数以及如何创建时间序列的更多信息，请参阅 Using metric math。
如何比较平均预置写入容量与使用的写入容量？	监控平均预置写入容量和使用的写入容量 1. 将 ConsumedWriteCapacityUnits 和 ProvisionedWriteCapacityUnits 的周期设置为要监控的时间间隔。 2. 将 ConsumedWriteCapacityUnits 的统计从 Average 更改为 Sum。 3. 新建一个空的 Math expression (数学表达式)。 4. 在新数学表达式的详细信息部分，输入指标的 ID，然后将指标除以该指标的**ConsumedWriteCapacityUnits** CloudWatch 周期函数 (metric_id/ (PERIOD (metric_id))。 5. 取消选择 ConsumedWriteCapacityUnits 。 现在，您可以将使用的平均写入容量与预置容量进行比较。有关基本算术函数以及如何创建时间序列的更多信息，请参阅 Using metric math。

Amazon Keyspaces 指标与维度

当您与 Amazon Keyspaces 互动时，它会向亚马逊发送以下指标和维度。 CloudWatch所有指标每分钟汇总和报告一次。您可以使用以下流程查看 Amazon Keyspaces 的指标。

使用 CloudWatch 控制台查看指标

指标的分组首先依据服务命名空间，然后依据每个命名空间内的各种维度组合。

1. 打开 CloudWatch 控制台，网址为 https://console.aws.amazon.com/cloudwatch/。
2. 如果需要，可以更改区域。在导航栏上，选择您的 AWS 资源所在的区域。有关更多信息，请参阅AWS 服务端点。
3. 在导航窗格中，选择指标。
4. 在 All metrics (全部指标) 选项卡下，选择 `AWS/Cassandra.`。

使用 AWS CLI 查看指标

- 在命令提示符处，使用以下命令。

```
aws cloudwatch list-metrics --namespace "AWS/Cassandra"
```

Amazon Keyspaces 指标与维度

此处列出了 Amazon Keyspaces 发送给亚马逊 CloudWatch 的指标和维度。

Amazon Keyspaces 指标

亚马逊每 CloudWatch 隔一分钟汇总亚马逊密钥空间指标。

并非所有的统计数据，如 `Average` 或 `Sum`，都适用于每个指标。但是，所有这些值都可通过 Amazon Keyspaces 控制台获得，或者使用 CloudWatch 控制台或 AWS 软件开发工具包获取所有指标。 AWS CLI在下表中，每个度量都有一个适用于该度量的有效统计信息列表。

指标	描述
`AccountMaxTableLevelReads`	账户的表可以使用的最大读取容量单位数。对于按需表，此值将限制表可以使用的最大读取请求单位。

指标	描述
	单位：Count 有效统计数据： • Maximum：账户的表可以使用的最大读取容量单位数。
AccountMaxTableLevelWrites	账户的表可以使用的最大写入容量单位数。对于按需表，此值将限制表可以使用的最大写入请求单位。 单位：Count 有效统计数据： • Maximum：账户的表可以使用的最大写入容量单位数。
AccountProvisionedReadCapacityUtilization	账户使用的预置读取容量单位百分比。 单位：Percent 有效统计数据： • Maximum – 账户使用的最大预置读取容量单位百分比。 • Minimum – 账户使用的最小预置读取容量单位百分比。 • Average – 账户使用的平均预置读取容量单位百分比。该指标每五分钟发布一次。因此，如果快速调整预置读取容量单位，则此统计数据可能不会反映实际平均值。
AccountProvisionedWriteCapacityUtilization	账户使用的预置写入容量单位百分比。 单位：Percent 有效统计数据： • Maximum – 账户使用的最大预置写入容量单位百分比。 • Minimum – 账户使用的最小预置写入容量单位百分比。 • Average – 账户使用的平均预置写入容量单位百分比。该指标每五分钟发布一次。因此，如果快速调整预置写入容量单位，则此统计数据可能不会反映实际平均值。

指标	描述
BillableTableSizeInBytes	可计费的表大小（以字节为单位）。它是表中所有行的编码大小的总和。该指标可帮助您跟踪一段时间内的表存储成本。 单位：Bytes 维度：Keyspace, TableName 有效统计数据： • Maximum：表的最大存储大小。 • Minimum：表的最小存储大小。 • Average：表的平均存储大小。该指标的计算间隔为 4-6 小时。
ConditionalCheckFailedRequests	失败的轻量级事务 (LWT) 写入请求数。INSERT、UPDATE 和 DELETE 操作允许提供一个逻辑条件，该条件计算结果必须为 true，才能继续操作。如果此条件的计算结果为 false，则 ConditionalCheckFailedRequests 增加 1。评估为 false 的条件检查会根据行的大小消耗写入容量单位。有关更多信息，请参阅 the section called “轻量级事务”。 单位：Count 维度：Keyspace, TableName 有效统计数据： • Minimum • Maximum • Average • SampleCount • Sum

指标	描述
ConsumedReadCapacityUnits	在指定时间段内使用的读取容量单位数。有关更多信息，请参阅 Read/Write capacity mode。 **Note** 要了解每秒平均吞吐量利用率，请使用 Sum 统计数据计算一分钟内消耗的吞吐量。然后，将和值除以每分钟秒数 (60) 来计算每秒的平均 ConsumedReadCapacityUnits （确认此平均值不会突出这一分钟内出现的任何大而短暂的读取活动峰值）。有关将使用的平均读取容量与预置读取容量进行比较的详细信息，请参阅 the section called “使用指标”。 单位：Count 维度：Keyspace, TableName 有效统计数据： • Minimum：对表的任何单个请求所使用的最小读取容量单位数。 • Maximum：对表的任何单个请求所使用的最大读取容量单位数。 • Average – 每个请求占用的平均读取容量。 **Note** Average 值受不活动时间的影响，不活动时的采样值将为零。 • Sum – 占用的总读取容量单位。这是对 ConsumedReadCapacityUnits 指标最有用的统计数据。 • SampleCount ：对 Amazon Keyspaces 的请求数（即使未使用读取容量）。

指标	描述
	Note `SampleCount` 值受不活动时间的影响，不活动时的采样值将为零。

指标	描述
ConsumedWriteCapacityUnits	在指定时间段内使用的写入容量单位数。您可以检索表使用的总写入容量。有关更多信息，请参阅 Read/Write capacity mode。 **Note** 要了解每秒平均吞吐量利用率，请使用 Sum 统计数据计算一分钟内消耗的吞吐量。然后，将和值除以每分钟秒数 (60) 来计算每秒的平均 ConsumedWriteCapacityUnits （确认此平均值不会突出这一分钟内出现的任何大而短暂的写入活动峰值）。有关将使用的平均写入容量与预置写入容量进行比较的详细信息，请参阅 the section called “使用指标”。 单位：Count 维度：Keyspace, TableName 有效统计数据： • Minimum：对表的任何单个请求所使用的最小写入容量单位数。 • Maximum：对表的任何单个请求所使用的最大写入容量单位数。 • Average – 每个请求占用的平均写入容量。 **Note** Average 值受不活动时间的影响，不活动时的采样值将为零。 • Sum – 占用的总写入容量单位。这是对 ConsumedWriteCapacityUnits 指标最有用的统计数据。

指标	描述
	• SampleCount ：对 Amazon Keyspaces 的请求数（即使未使用写入容量）。 Note SampleCount 值受不活动时间的影响，不活动时的采样值将为零。
MaxProvisionedTableReadCapacityUtilization	账户的最高预置读取表所使用的预置读取容量单位的百分比。 单位：Percent 有效统计数据： • Maximum – ：账户的最高预置读取表使用的最大预置读取容量单位百分比。 • Minimum – 账户的最高预置读取表使用的最小预置读取容量单位百分比。 • Average – 账户的最高预置读取表使用的平均预置读取容量单位百分比。该指标每五分钟发布一次。因此，如果快速调整预置读取容量单位，则此统计数据可能不会反映实际平均值。

指标	描述
MaxProvisionedTableWriteCapacityUtilization	账户的最高预置写入表所使用的预置写入容量的百分比。 单位：Percent 有效统计数据： • Maximum：账户的最高预置写入表所使用的预置写入容量单位的最大百分比。 • Minimum：账户的最高预置写入表所使用的预置写入容量单位的最小百分比。 • Average：账户的最高预置写入表所使用的预置写入容量单位的平均百分比。该指标每五分钟发布一次。因此，如果快速调整预置写入容量单位，则此统计数据可能不会反映实际平均值。
PerConnectionRequestRateExceeded	对 Amazon Keyspaces 的请求数超出每个连接请求速率的限额。与 Amazon Keyspaces 的每个客户端连接每秒最多可支持 3000 个 CQL 请求。客户端可以创建多个连接以增加吞吐量。 当您使用多区域复制时，每项复制的写入也会占用此限额。作为最佳实践，我们建议增加表的连接数以避免 PerConnectionRequestRateExceeded 错误。对您在 Amazon Keyspaces 中可拥有的连接数没有限制。 单位：Count 维度：Keyspace, TableName, Operation 有效统计数据： • SampleCount • Sum

指标	描述
ProvisionedReadCapacityUnits	表的预置读取容量单位数。 TableName 维度返回表的 ProvisionedReadCapacityUnits 。 单位：Count 维度：Keyspace, TableName 有效统计数据： • Minimum – 预置读取容量的最低设置。如果使用 ALTER TABLE 增加读取容量，则此指标显示此时间段的预置 ReadCapacityUnits 最低值。 • Maximum – 预置读取容量的最高设置。如果使用 ALTER TABLE 减少读取容量，则此指标显示此时间段的预置 ReadCapacityUnits 最高值。 • Average – 平均预置读取容量。ProvisionedReadCapacityUnits 指标每五分钟发布一次。因此，如果快速调整预置读取容量单位，则此统计数据可能不会反映实际平均值。

指标	描述
ProvisionedWriteCapacityUnits	表的预置写入容量单位数。 TableName 维度返回表的 ProvisionedWriteCapacityUnits 。 单位：Count 维度：Keyspace, TableName 有效统计数据： • Minimum – 预置写入容量的最低设置。如果使用 ALTER TABLE 增加写入容量，则此指标显示此时间段的预置 WriteCapacityUnits 最低值。 • Maximum – 预置写入容量的最高设置。如果使用 ALTER TABLE 减少写入容量，则此指标显示此时间段的预置 WriteCapacityUnits 最高值。 • Average – 平均预置写入容量。ProvisionedWriteCapacityUnits 指标每五分钟发布一次。因此，如果快速调整预置写入容量单位，则此统计数据可能不会反映实际平均值。
ReadThrottleEvents	对 Amazon Keyspaces 的请求数超出表的预置读取容量、或账户级别限额、每个连接的请求限额或分区级别限额。 单位：Count 维度：Keyspace, TableName, Operation 有效统计数据： • SampleCount • Sum

指标	描述
ReplicationLatency	该指标仅适用于多区域键空间，并且可用于衡量将 updates、inserts 或 deletes 从一个副本表复制到多区域键空间中的另一副本表所花费的时间。 单位：Millisecond 维度：TableName, ReceivingRegion 有效统计数据： • Average • Maximum • Minimum
ReturnedItemCountBySelect	指定时间段内多行 SELECT 查询返回的行数。多行 SELECT 查询是不包含完全限定主键的查询，例如全表扫描和范围查询。 返回的 行数并不一定与已计算的行数相同。例如，假设您请求对一个具有 100 行的表执行具有 ALLOW FILTERING 的 SELECT *，但指定 WHERE 子句来缩小结果范围，以便仅返回 15 行。在此情况下，来自 SELECT 的响应包含的 ScanCount 为 100，Count 为返回的 15 行。 单位：Count 维度：Keyspace, TableName, Operation 有效统计数据： • Minimum • Maximum • Average • SampleCount • Sum

指标	描述
StoragePartitionThroughputCapacityExceeded	对 Amazon Keyspaces 存储分区的请求数超过了该分区的吞吐容量。Amazon Keyspaces 存储分区每秒最多可支持 1000 个 WCU/WRU，每秒最多可支持 3000 个 RCU/RRU。我们建议审查数据模型，将读/写流量分配到更多分区，以减少这些异常情况。 **Note** Amazon Keyspaces 逻辑分区可以跨越多个存储分区，并且大小几乎不受限制。 单位：Count 维度：Keyspace, TableName, Operation 有效统计数据： • SampleCount • Sum
SuccessfulRequestCount	在指定时间段内成功处理的请求数。 单位：Count 维度：Keyspace, TableName, Operation 有效统计数据： • SampleCount

指标	描述
SuccessfulRequestLatency	指定时间段内对 Amazon Keyspaces 的成功请求数。SuccessfulRequestLatency 可提供两种不同类型的信息： • 成功请求的所用时间（Minimum、Maximum、Sum 或 Average）。 • 成功的请求数 (SampleCount)。 SuccessfulRequestLatency 仅反映 Amazon Keyspaces 中的活动，并且不考虑网络延迟或客户端活动。 单位：Milliseconds 维度：Keyspace, TableName, Operation 有效统计数据： • Minimum • Maximum • Average • SampleCount
SystemErrors	在指定时间段内生成 ServerError 的对 Amazon Keyspaces 的请求数。ServerError 通常指示内部服务错误。 单位：Count 维度：Keyspace, TableName, Operation 有效统计数据： • Sum • SampleCount

指标	描述
SystemReconciliationDeletes	启用客户端时间戳时删除 Tombstoned 数据所使用的单位。每个 SystemReconciliationDelete 都提供足够的容量，可用于删除或更新每行最多 1 KB 的数据。例如，要更新一个存储了 2.5 KB 数据的行并同时删除该行内的一个或多个列，将需要 3 次 SystemReconciliationDeletes 。或者，要删除一个包含 3.5 KB 数据的完整行，将需要 4 次 SystemReconciliationDeletes 。 单位：Count 维度：Keyspace, TableName 有效统计数据： • Sum：一段时间内使用的 SystemReconciliationDeletes 总数。
TTLDeletes	使用生存时间 (TTL) 删除或更新一行数据所使用的单位。每个 TTLDelete 都提供足够的容量，可用于删除或更新每行最多 1 KB 的数据。例如，要更新一个存储了 2.5 KB 数据的行并同时删除该行内的一个或多个列，将需要 3 次 TTL 删除操作。或者，要删除一个包含 3.5 KB 数据的完整行，将需要 4 次 TTL 删除操作。 单位：Count 维度：Keyspace, TableName 有效统计数据： • Sum：一段时间内使用的 TTLDeletes 总数。

指标	描述
UserErrors	在指定时间段内生成 InvalidRequest 错误的对 Amazon Keyspaces 的请求。InvalidRequest 通常指示客户端错误，例如，参数组合无效、正在尝试更新不存在的表或请求签名错误。 UserErrors 表示当前请求 AWS 区域 和当前请求的无效请求的汇总 AWS 账户。 单位：Count 维度：Keyspace, TableName, Operation 有效统计数据： • Sum • SampleCount
WriteThrottleEvents	对 Amazon Keyspaces 的请求数超出表的预置写入容量、或账户级别限额、每个连接的请求限额或分区级别限额。 单位：Count 维度：Keyspace, TableName, Operation 有效统计数据： • SampleCount • Sum

Amazon Keyspaces 指标的维度

Amazon Keyspaces 的指标由账户、表名或操作的值进行限定。您可以使用 CloudWatch 控制台按下表中任意维度检索 Amazon Keyspaces 数据。

维度	描述
Keyspace	该维度将数据限定为特定键空间。该值可以是当前区域和当前 AWS 账户中的任意键空间。
Operation	该维度将数据限定为 Amazon Keyspaces CQL 操作之一，如 INSERT 或 SELECT 操作。
TableName	此维度将数据限制为特定表。该值可以是当前区域和当前 AWS 账户中的任意表名。如果表名称在账户中不是唯一的，则还必须指定 Keyspace。

创建 CloudWatch 警报以监控 Amazon Keyspaces

您可以为亚马逊密钥空间创建亚马逊 CloudWatch 警报，当警报状态发生变化时，该警报会发送亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 消息。告警会监控您指定的时间段内的某个指标。它在多个时间段内根据相对于给定阈值的指标值，执行一项或多项操作。操作是一个发送到 Amazon SNS 主题或 Application Auto Scaling 策略的通知。

当您在预配置模式下使用 Amazon Keyspaces 和 Application Auto Scaling 时，该服务会代表您创建两对 CloudWatch 警报。每对警报均表示预置和使用的吞吐量设置的上限和下限。当表的实际利用率在持续一段时间内偏离目标利用率时，就会触发这些 CloudWatch 警报。要了解有关 Application Auto Scaling 创建的 CloudWatch 警报的更多信息，请参阅the section called “Amazon Keyspaces 自动扩缩的工作原理”。

警报仅针对持续的状态变化调用操作。 CloudWatch 警报不会仅仅因为它们处于特定状态就调用操作。该状态必须已改变并在指定的若干个时间段内保持不变。

有关创建 CloudWatch 警报的更多信息，请参阅亚马逊 CloudWatch 用户指南中的使用亚马逊 CloudWatch 警报。

使用 AWS CloudTrail 记录 Amazon Keyspaces API 调用

Amazon Keyspaces 与 AWS CloudTrail 集成，后者是一项服务，用于在 Amazon Keyspaces 中提供用户、角色或 AWS 服务所采取操作的记录。CloudTrail 捕获 Amazon Keyspaces 的数据定义语言 (DDL, Data Definition Language) API 调用和数据操作语言 (DDL, Data Manipulation Language) API 调用作为事件。捕获的调用包括来自 Amazon Keyspaces 控制台的调用和对 Amazon Keyspaces API 操作的编程调用。

如果您创建跟踪，则可以使 CloudTrail 事件持续传送到 Amazon Simple Storage Service (Amazon S3) 桶，包括 Amazon Keyspaces 事件。

如果您不配置跟踪，则仍可在 CloudTrail 控制台的事件历史记录中查看最新支持的事件。利用 CloudTrail 收集的信息，您可以确定向 Amazon Keyspaces 发出的请求、发出请求的 IP 地址、发出请求的人员、发出请求的时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息，请参阅 AWS CloudTrail 用户指南。

主题

- 在 CloudTrail 中配置 Amazon Keyspaces 日志文件条目
- CloudTrail 中的 Amazon Keyspaces 数据定义语言 (DDL) 信息
- CloudTrail 中的 Amazon Keyspaces 数据操作语言 (DML) 信息
- 了解 Amazon Keyspaces 日志文件条目

在 CloudTrail 中配置 Amazon Keyspaces 日志文件条目

CloudTrail 中记录的每个 Amazon Keyspaces API 操作都包含以 CQL 查询语言表示的请求参数。有关更多信息，请参阅CQL 语言参考。

您可以在 AWS 账户 中查看、搜索和下载最新事件。有关更多信息，请参阅使用 CloudTrail 事件历史记录查看事件。

要持续记录 AWS 账户中的事件（包括 Amazon Keyspaces 的事件），请创建跟踪。通过跟踪记录，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪时，此跟踪应用于所有 AWS 区域。此跟踪记录在 AWS 分区中记录所有区域中的事件，并将日志文件传送至您指定的 Amazon S3 桶。此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。

有关更多信息，请参阅《AWS CloudTrail 用户指南》中的以下主题：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件
- 从多个账户接收 CloudTrail 日志文件

每个事件或日志条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 AWS Identity and Access Management (IAM) 用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其它 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

CloudTrail 中的 Amazon Keyspaces 数据定义语言 (DDL) 信息

在您创建 AWS 账户时，将在该账户上启用 CloudTrail。当 Amazon Keyspaces 中发生 DDL 活动时，该活动将自动记录为 CloudTrail 事件，并与其他 AWS 服务事件一起保存在事件历史记录中。下表显示了为 Amazon Keyspaces 记录的 DDL 语句。

CloudTrail **eventName**	语句	CQL 操作	AWS SDK 操作
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`

CloudTrail 中的 Amazon Keyspaces 数据操作语言 (DML) 信息

要在 CloudTrail 中对 Amazon Keyspaces DML 语句启用日志记录，您必须先在 CloudTrail 中对数据面板 API 操作启用日志记录。您可以使用 CloudTrail 控制台选择记录数据事件类型为 Cassandra 表的活动，或者使用 AWS CLI 或 CloudTrail API 操作将 `resources.type` 值设置为 `AWS::Cassandra::Table`，即可开始在新跟踪或现有跟踪中记录 Amazon Keyspaces DML 事件。有关更多信息，请参阅记录数据事件。

下表显示 CloudTrail 可以为 `Cassandra table` 记录的数据事件。

CloudTrail **eventName**	语句	CQL 操作	AWS SDK 操作
Select	DML	SELECT	GetKeyspace ，GetTable, ListKeyspaces ，ListTables ListTagsForResource
Insert	DML	INSERT	没有 AWS SDK 操作可用
更新	DML	UPDATE	没有 AWS SDK 操作可用
Delete	DML	DELETE	没有 AWS SDK 操作可用

了解 Amazon Keyspaces 日志文件条目

CloudTrail 日志文件包含一个或多个日记账条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

下面的示例显示了一个 CloudTrail 日志条目，该条目说明了 CreateKeyspace、DropKeyspace、CreateTable 和 DropTable 操作：

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

以下日志文件显示 SELECT 语句的示例。

```
{
    "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"
    }
}
```

以下日志文件显示 INSERT 语句的示例。

```
{
    "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"
    }
}
```

以下日志文件显示 UPDATE 语句的示例。

```
{
    "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"
        }
}
```

以下日志文件显示 DELETE 语句的示例。

```
{
    "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 兼容）中的安全性

AWS 十分重视云安全性。作为 AWS 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。

安全性是AWS和您的共同责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS负责保护在 AWS Cloud 中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。作为 AWS 合规性计划的一部分，我们的安全措施的有效性定期由第三方审计员进行测试和验证。要了解适用于 Amazon Keyspaces 的合规性计划，请参阅按合规性计划提供的范围内 AWS 服务。
- 云中的安全性 - 您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您组织的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 Amazon Keyspaces 时应用责任共担模式。以下主题说明了如何配置 Amazon Keyspaces 以实现您的安全性和合规性目标。您还将了解如何使用其他 AWS 服务来帮助您监控和保护 Amazon Keyspaces 资源。

主题

Amazon Keyspaces 中的数据保护

AWS 责任共担模式适用于 Amazon Keyspaces（Apache Cassandra 兼容）中的数据保护。如该模式中所述，AWS 负责保护运行所有 AWS Cloud 的全球基础设施。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的博客文章 AWS Shared Responsibility Model and GDPR。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置单个用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与 AWS 资源进行通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务 中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准 (FIPS) 第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括使用控制台、API、AWS CLI 或 AWS SDK 处理 Amazon Keyspaces 或其他 AWS 服务时。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，我们强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

主题

Amazon Keyspaces 中的静态加密

Amazon Keyspaces（Apache Cassandra 兼容）静态加密使用存储在 AWS Key Management Service (AWS KMS) 中的加密密钥对所有静态数据进行加密，以增强安全性。此功能减少保护敏感数据时涉及的操作负担和复杂性。通过静态加密，您可以构建安全敏感型应用程序，以满足针对数据保护的严格合规性和法规要求。

Amazon Keyspaces 静态加密使用 256 位高级加密标准 (AES-256) 来加密数据。这有助于保护您的数据，防止对底层存储进行未经授权的访问。

Amazon Keyspaces 以透明方式加密和解密表数据。Amazon Keyspaces 使用信封加密和密钥层次结构来保护数据加密密钥。它与 AWS KMS 集成，用于存储和管理根加密密钥。有关加密密钥层次结构

的更多信息，请参阅the section called “工作原理”。有关 AWS KMS 概念（如信封加密）的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 AWS KMS 管理服务概念。

创建新表时，您可以选择以下 AWS KMS 密钥（KMS 密钥）之一：

- AWS 拥有的密钥：这是默认加密类型。此密钥由 Amazon Keyspaces 拥有（不另外收费）。
- 客户自主管理型密钥：此密钥存储在您的账户中，由您创建、拥有和管理。您对客户自主管理型密钥拥有完全控制权（收取 AWS KMS 费用）。

您可以随时在 AWS 拥有的密钥和客户自主管理型密钥之间切换。您可以在创建新表时指定客户自主管理型密钥，也可以通过控制台或使用 CQL 语句以编程方式更改现有表的 KMS 密钥。要了解如何操作，请参阅静态加密：如何使用客户自主管理型密钥加密 Amazon Keyspaces 中的表。

使用默认选项 AWS 拥有的密钥的静态加密不额外收费。但是，使用客户自主管理型密钥需支付 AWS KMS 费用。有关定价的更多信息，请参阅 AWS KMS 定价。

Amazon Keyspaces 静态加密已在所有 AWS 区域推出，包括 AWS 中国（北京）和 AWS 中国（宁夏）区域。有关更多信息，请参阅静态加密：它在 Amazon Keyspaces 中如何运作。

主题

静态加密：它在 Amazon Keyspaces 中如何运作

Amazon Keyspaces（Apache Cassandra 兼容）静态加密使用 256 位高级加密标准 (AES-256) 来加密数据。这有助于保护您的数据，防止对底层存储进行未经授权的访问。默认情况下，Amazon Keyspaces 表中的所有客户数据都进行了静态加密，服务器端加密是透明的，这意味着不需要对应用程序进行更改。

静态加密集成 AWS Key Management Service (AWS KMS)，管理用于加密表的加密密钥。在创建新表或更新现有表时，您可以选择以下 AWS KMS 密钥选项之一：

- AWS 拥有的密钥：这是默认加密类型。此密钥由 Amazon Keyspaces 拥有（不另外收费）。
- 客户自主管理型密钥：此密钥存储在您的账户中，由您创建、拥有和管理。您对客户自主管理型密钥拥有完全控制权（收取 AWS KMS 费用）。

AWS KMS 密钥（KMS 密钥）

静态加密使用 AWS KMS 密钥保护您的所有 Amazon Keyspaces 数据。默认情况下，Amazon Keyspaces 使用 AWS 拥有的密钥，即在 Amazon Keyspaces 服务账户中创建并管理的多租户加密密钥。

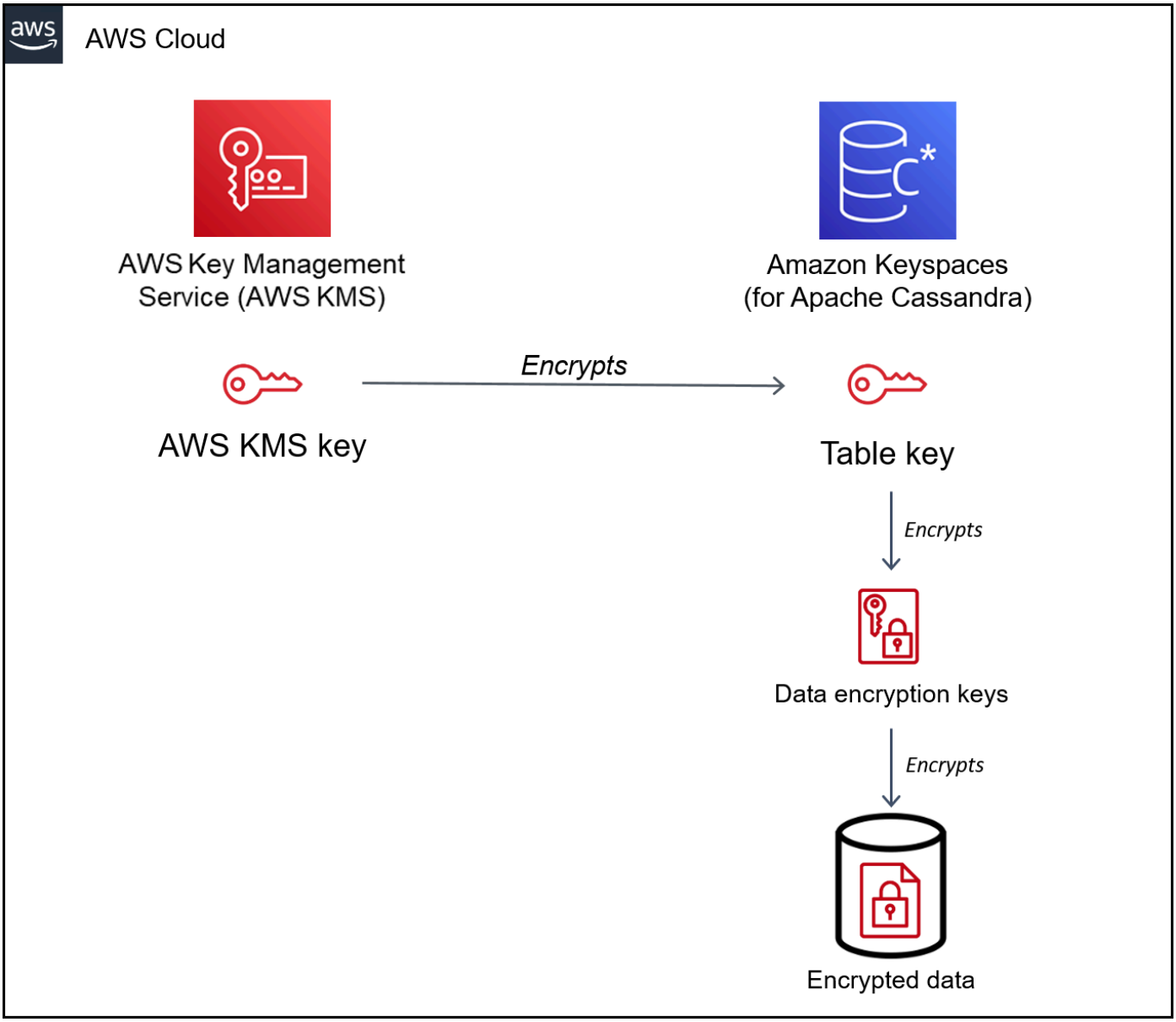
但是，您可以使用 AWS 账户中的客户自主管理型密钥加密 Amazon Keyspaces 表。您可以为键空间中的每个表选择不同的 KMS 密钥。您为表选择的 KMS 密钥也用于加密其所有元数据和可恢复备份。

您可以在创建或更新表时为表选择 KMS 密钥。您可以随时在 Amazon Keyspaces 控制台中或使用 ALTER TABLE 语句更改表的 KMS 密钥。切换 KMS 密钥的过程是无缝的，不需要停机，也不会降低服务质量。

密钥层次结构

Amazon Keyspaces 使用密钥层次结构来加密数据。在这个密钥层次结构中，KMS 密钥是根密钥。它用于加密和解密 Amazon Keyspaces 表加密密钥。表加密密钥用于加密 Amazon Keyspaces 在内部使用的加密密钥（用于在执行读取和写入操作时加密和解密数据）。

利用加密密钥层次结构，您可以对 KMS 密钥进行更改，而无需重新加密数据，也不会影响应用程序和正在进行的数据操作。



表密钥

Amazon Keyspaces 表密钥用作密钥加密密钥。Amazon Keyspaces 使用表密钥来保护内部数据加密密钥，后者用于加密存储在表、日志文件和可恢复备份中的数据。Amazon Keyspaces 会为表中的每个底层结构生成唯一的数据加密密钥。但是，多个表行可能受同一个数据加密密钥的保护。

首次将 KMS 密钥设置为客户自主管理型密钥时，AWS KMS 会生成一个数据密钥。AWS KMS 数据密钥是指 Amazon Keyspaces 中的表密钥。

当您访问加密表时，Amazon Keyspaces 会向 AWS KMS 发送请求以使用 KMS 密钥解密表密钥。然后，它会使用明文表密钥来解密 Amazon Keyspaces 数据加密密钥，并使用明文数据加密密钥来解密表数据。

Amazon Keyspaces 在 AWS KMS 外部使用和存储表密钥及数据加密密钥。它会借助高级加密标准 (AES) 加密和 256 位加密密钥保护所有密钥。然后，它会将加密密钥与加密数据存储在一起，这样您就可以根据需要使用它们来解密表数据。

表密钥缓存

为了避免针对每个 Amazon Keyspaces 操作调用 AWS KMS，Amazon Keyspaces 会针对每个连接将明文表密钥缓存在内存中。如果 Amazon Keyspaces 在处于不活动状态 5 分钟后收到了针对缓存表密钥的请求，它会向 AWS KMS 发送新请求以解密表密钥。此调用将捕获自上次请求解密表密钥以来对 AWS KMS 或 AWS Identity and Access Management (IAM) 中的 KMS 密钥的访问策略所做的任何更改。

信封加密

如果您更改了表的客户自主管理型密钥，Amazon Keyspaces 会生成新的表密钥。然后，它会使用新的表密钥来重新加密数据加密密钥。它还会使用新的表密钥来加密用于保护可恢复备份的先前表密钥。此过程称为信封加密。这将确保即使您轮换了客户自主管理型密钥，您仍然可以访问可恢复备份。有关信封加密的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的信封加密。

主题

AWS 拥有的密钥

AWS 拥有的密钥不存储在 AWS 账户中。它们是 AWS 拥有和管理的用于多个 AWS 账户的 KMS 密钥集合的一部分。AWS 服务可以使用 AWS 拥有的密钥保护数据。

您无法查看、管理或使用 AWS 拥有的密钥，也无法审计其使用情况。但是无需执行任何工作或更改任何计划即可保护用于加密数据的密钥。

使用 AWS 拥有的密钥无需支付月费或使用费，并且它们不会计入您账户的 AWS KMS 限额。

客户托管密钥

客户自主管理型密钥是您在 AWS 账户中创建、拥有和管理的密钥。您对此类 KMS 密钥拥有完全控制权。

使用客户托管密钥可获得以下功能：

- 您可以创建和管理客户自主管理型密钥，包括设置和维护密钥策略、IAM 策略和授权来控制对客户自主管理型密钥的访问。您可以启用和禁用客户自主管理型密钥、启用和禁用自动密钥轮换，以及计划在客户自主管理型密钥不再使用时删除客户自主管理型密钥。您可以为自己管理的客户自主管理型密钥创建标签和别名。
- 您可以使用具有导入的密钥材料的客户托管密钥，或者您拥有和管理的自定义密钥存储中的客户托管密钥。
- 您可以使用 AWS CloudTrail 和 Amazon CloudWatch Logs 跟踪 Amazon Keyspaces 代表您发送到 AWS KMS 的请求。有关更多信息，请参阅the section called “步骤 6：使用 AWS CloudTrail 配置监控”。

客户自主管理型密钥会针对每个 API 调用产生费用，并且此类 KMS 密钥具有 AWS KMS 限额。有关更多信息，请参阅 AWS KMS 资源或请求限额。

当您将客户自主管理型密钥指定为表的根加密密钥时，系统将使用创建备份时为表指定的加密密钥对可恢复备份进行加密。如果表的 KMS 密钥进行了轮换，则密钥信封加密可确保最新的 KMS 密钥有权访问所有可恢复备份。

Amazon Keyspaces 必须有权访问您的客户自主管理型密钥才能让您有权访问您的表数据。如果加密密钥的状态设置为“已禁用”或将按计划删除，Amazon Keyspaces 将无法加密或解密数据。因此，您无法对表执行读取和写入操作。一旦服务检测到您的加密密钥无法访问，Amazon Keyspaces 会立即向您发送电子邮件通知以提醒您。

您必须在七天内恢复对加密密钥的访问权限，否则 Amazon Keyspaces 会自动删除您的表。作为预防措施，Amazon Keyspaces 会在删除表之前为表数据创建可恢复备份。Amazon Keyspaces 会将可恢复备份保留 35 天。35 天后，您将无法再恢复表数据。您无需为可恢复备份付费，但需要支付标准恢复费用。

您可以使用这个可恢复备份将数据恢复到新表。要启动恢复操作，必须启用用于表的最后一个客户自主管理型密钥，且 Amazon Keyspaces 必须有权访问它。

> **Note**
>
> 当您创建使用客户自主管理型密钥进行加密的表时，如果该密钥在创建过程完成之前不可访问或将按计划删除，则会发生错误。创建表操作将失败，并且您将收到电子邮件通知。

静态加密使用注意事项

在 Amazon Keyspaces 中使用静态加密时，请注意以下事项。

- 服务器端静态加密已在所有 Amazon Keyspaces 表上启用且无法禁用。整个表都进行了静态加密，您无法选择特定的列或行进行加密。
- 默认情况下，Amazon Keyspaces 使用单服务默认密钥（AWS 拥有的密钥）来加密您的所有表。如果此密钥不存在，系统会为您创建一个。服务默认密钥无法禁用。
- 静态加密仅加密持久存储介质上的静态数据。如果对于正在传输的数据或正在使用的数据而言，数据安全性很重要，则必须执行额外措施：
  - 传输中数据：Amazon Keyspaces 中的所有数据都在传输中加密。默认情况下，与 Amazon Keyspaces 的通信将通过安全套接字层 (SSL)/传输层安全性协议 (TLS) 加密进行保护。
  - 正在使用的数据：在将数据发送到 Amazon Keyspaces 之前使用客户端加密保护数据。
  - 客户自主管理型密钥：表中的静态数据始终使用客户自主管理型密钥进行加密。但是，对多行执行原子更新的操作将在处理过程中使用 AWS 拥有的密钥临时加密数据。这包括范围删除操作以及同时访问静态和非静态数据的操作。
- 一个客户自主管理型密钥最多可以有 50000 个授权。与客户自主管理型密钥关联的每个 Amazon Keyspaces 表使用 2 个授权。删除表会释放一个授权。第二个授权用于创建表的自动快照，防止在 Amazon Keyspaces 无意中失去对客户自主管理型密钥的访问权限时发生数据丢失。此授权将在删除表后 42 天释放。

静态加密：如何使用客户自主管理型密钥加密 Amazon Keyspaces 中的表

您可以使用控制台或 CQL 语句为新表指定 AWS KMS key 并更新 Amazon Keyspaces 中现有表的加密密钥。以下主题概述如何对新表和现有表实施客户自主管理型密钥。

主题

- 先决条件：使用 AWS KMS 创建客户自主管理型密钥并向 Amazon Keyspaces 授予权限
- 步骤 3：为新表指定客户自主管理型密钥
- 步骤 4：更新现有表的加密密钥
- 步骤 5：在日志中使用 Amazon Keyspaces 加密上下文
- 步骤 6：使用 AWS CloudTrail 配置监控

先决条件：使用 AWS KMS 创建客户自主管理型密钥并向 Amazon Keyspaces 授予权限

必须先在 AWS Key Management Service (AWS KMS) 中创建客户自主管理型密钥并授权 Amazon Keyspaces 使用该密钥，然后才能使用该密钥保护 Amazon Keyspaces 表。

步骤 1：使用 AWS KMS 创建客户自主管理型密钥

要创建用于保护 Amazon Keyspaces 表的客户自主管理型密钥，您可以按照使用控制台或 AWS API 创建对称加密 KMS 密钥中的步骤进行操作。

步骤 2：授权使用您的客户自主管理型密钥

要选择一个客户自主管理型密钥来保护 Amazon Keyspaces 表，必须先让针对该密钥的策略为 Amazon Keyspaces 授予代表您使用该密钥的权限。您可以全面控制针对客户自主管理型密钥的策略和授权。您可以在密钥策略、IAM 策略或授权中提供这些权限。

Amazon Keyspaces 无需额外授权即可使用默认的 AWS 拥有的密钥来保护 AWS 账户中的 Amazon Keyspaces 表。

以下主题介绍如何使用允许 Amazon Keyspaces 表使用客户自主管理型密钥的 IAM 策略和授权来配置所需的权限。

主题

- 针对客户自主管理型密钥的密钥策略
- 示例密钥策略
- 使用授权来授权 Amazon Keyspaces

针对客户自主管理型密钥的密钥策略

如果选择客户自主管理型密钥来保护 Amazon Keyspaces 表，那么 Amazon Keyspaces 将获得代表做出此选择的主体使用客户自主管理型密钥的权限。该主体（用户或角色）必须拥有 Amazon Keyspaces 所需的客户自主管理型密钥权限。

Amazon Keyspaces 至少需要以下客户自主管理型密钥权限：

- kms:Encrypt
- kms:Decrypt
- kms:ReEncrypt*（用于 kms:ReEncryptFrom 和 kms:ReEncryptTo）
- kms:GenerateDataKey*（用于 kms:GenerateDataKey 和 kms:GenerateDataKeyWithoutPlaintext）
- kms:DescribeKey
- kms:CreateGrant

示例密钥策略

例如，以下示例密钥策略仅提供所需的权限。该策略具有以下效果：

- 允许 Amazon Keyspaces 在加密操作中使用客户自主管理型密钥并创建授权，但仅当它代表账户中拥有 Amazon Keyspaces 使用权限的主体行事时才可如此。如果策略语句中指定的主体无权使用 Amazon Keyspaces，调用将失败，即使调用来自 Amazon Keyspaces 服务也是如此。
- 仅当 Amazon Keyspaces 代表策略语句中所列主体发出请求时，kms:ViaService 条件键才允许权限。这些委托人不能直接调用这些操作。请注意，`kms:ViaService` 值 `cassandra.*.amazonaws.com`在“区域”位置中有一个星号 (*)。Amazon Keyspaces 要求权限与任何特定 AWS 区域无关。
- 为客户自主管理型密钥管理员（可代入 `db-team` 角色的用户）授予对客户自主管理型密钥的只读访问权限，以及撤销授权的权限，包括 Amazon Keyspaces 保护表所需的授权。
- 为 Amazon Keyspaces 授予对客户自主管理型密钥的只读访问权限。在此情况下，Amazon Keyspaces 可以直接调用这些操作，而不必代表账户主体行事。

在使用示例密钥策略之前，请将示例委托人替换为 AWS 账户 中的实际委托人。

```
{
  "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": "*"
    }
  ]
}
```

使用授权来授权 Amazon Keyspaces

除密钥策略之外，Amazon Keyspaces 还使用授权来设置对客户自主管理型密钥的权限。要查看对您账户中的客户自主管理型密钥的授权，请使用 ListGrants 操作。Amazon Keyspaces 不需要授权或任何其他权限即可使用 AWS 拥有的密钥来保护您的表。

Amazon Keyspaces 在执行后台系统维护和持续数据保护任务时使用授予权限。它还使用授权来生成表密钥。

每个授权特定于一个表。如果该账户包含使用同一客户自主管理型密钥加密的多个表，则每个表都有每种类型的授权。该授权受到 Amazon Keyspaces 加密上下文的约束，其中包括表名称和 AWS 账户 ID。该授权包含在不再需要授权时停用授权的权限。

要创建授权，Amazon Keyspaces 必须拥有代表创建加密表的用户调用 `CreateGrant` 的权限。

该密钥策略还可以允许账户撤销对客户自主管理型密钥的授权。但是，如果您撤销对某个活动的加密表的授权，Amazon Keyspaces 将无法保护和维护该表。

步骤 3：为新表指定客户自主管理型密钥

按照以下步骤，使用 Amazon Keyspaces 控制台或 CQL 指定新表的客户自主管理型密钥。

使用客户自主管理型密钥创建加密表（控制台）

1. 登录 AWS Management Console并打开 Amazon Keyspaces 控制台：https://console.aws.amazon.com/keyspaces/home。
2. 在导航窗格中，选择 Tables (表)，然后选择 Create table (创建表)。
3. 在创建表页面的表详细信息部分中，选择一个键空间并为新表提供一个名称。
4. 在架构部分，为您的表创建架构。
5. 在表设置部分，选择自定义设置。
6. 继续进入加密设置。

   在此步骤中，您将为表选择加密设置。

   在静态加密部分的选择 AWS KMS key下，选择选择其他 KMS 密钥（高级）选项，然后在搜索字段中选择一个 AWS KMS key或输入 Amazon 资源名称 (ARN)。

   > **Note**
   >
   > 如果您选择的密钥不可访问或缺少所需的权限，请参阅《AWS Key Management Service 开发人员指南》中的密钥访问问题排查。

7. 选择 Create (创建) 以创建加密表。

创建使用客户自主管理型密钥进行静态加密 (CQL) 的新表

要创建使用客户自主管理型密钥进行静态加密 (CQL) 的新表，可以使用 CREATE TABLE 语句，如以下示例所示。确保将密钥 ARN 替换为一个有效密钥的 ARN，该有效密钥拥有授予 Amazon Keyspaces 的权限。

```
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'
            }
    };
```

如果您收到 `Invalid Request Exception`，则需要确认客户自主管理型密钥有效，并且 Amazon Keyspaces 拥有所需的权限。要确认密钥已正确配置，请参阅《AWS Key Management Service 开发人员指南》中的密钥访问问题排查。

步骤 4：更新现有表的加密密钥

您还可以随时使用 Amazon Keyspaces 控制台或 CQL，在 AWS 拥有的密钥和客户自主管理型 KMS 密钥之间更改现有表的加密密钥。

使用新的客户自主管理型密钥更新现有表（控制台）

1. 登录 AWS Management Console并打开 Amazon Keyspaces 控制台：https://console.aws.amazon.com/keyspaces/home。
2. 在导航窗格中，选择表。
3. 选择要更新的表，然后选择其他设置选项卡。
4. 在静态加密部分中，选择管理加密以编辑表的加密设置。

   在选择 AWS KMS key下，选择选择其他 KMS 密钥（高级）选项，然后在搜索字段中选择一个 AWS KMS key或输入 Amazon 资源名称 (ARN)。

   > **Note**
   >
   > 如果您选择的密钥无效，请参阅《AWS Key Management Service 开发人员指南》中的密钥访问问题排查。

   或者，您也可以为使用客户自主管理型密钥加密的表选择 AWS 拥有的密钥。
5. 选择保存更改以保存对表所做的更改。

更新用于现有表的加密密钥

要更改现有表的加密密钥，请使用 `ALTER TABLE` 语句指定用于静态加密的客户自主管理型密钥。确保将密钥 ARN 替换为一个有效密钥的 ARN，该有效密钥拥有授予 Amazon Keyspaces 的权限。

```
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'
```

```
                    }
        };
```

如果您收到 `Invalid Request Exception`，则需要确认客户自主管理型密钥有效，并且 Amazon Keyspaces 拥有所需的权限。要确认密钥已正确配置，请参阅《AWS Key Management Service 开发人员指南》中的密钥访问问题排查。

要将加密密钥改回使用 `ALTER TABLE`的默认静态加密选项，可以使用 AWS 拥有的密钥 语句，如以下示例所示。

```
ALTER TABLE my_keyspace.my_table WITH CUSTOM_PROPERTIES = {
                'encryption_specification':{
                     'encryption_type' : 'AWS_OWNED_KMS_KEY'
                   }
        };
```

步骤 5：在日志中使用 Amazon Keyspaces 加密上下文

加密上下文 是一组包含任意非机密数据的键值对。在请求中包含加密上下文以加密数据时，AWS KMS 以加密方式将加密上下文绑定到加密的数据。要解密数据，您必须传入相同的加密上下文。

Amazon Keyspaces 在所有 AWS KMS 加密操作中使用相同的加密上下文。如果您使用客户自主管理型密钥来保护 Amazon Keyspaces 表，则可以使用加密上下文在审计记录和日志中标识客户自主管理型密钥的使用。它还以明文形式显示在日志中，例如在 AWS CloudTrail 和 Amazon CloudWatch Logs 的日志中。

在它对 AWS KMS 的请求中，Amazon Keyspaces 使用包含三个键值对的加密上下文。

```
"encryptionContextSubset": {
    "aws:cassandra:keyspaceName": "my_keyspace",
    "aws:cassandra:tableName": "mytable"
    "aws:cassandra:subscriberId": "111122223333"
}
```

- 键空间：第一个键值对标识包含 Amazon Keyspaces 要加密的表的键空间。键是 `aws:cassandra:keyspaceName`。值是键空间的名称。

  ```
  "aws:cassandra:keyspaceName": "<keyspace-name>"
  ```

  例如：

```
"aws:cassandra:keyspaceName": "my_keyspace"
```

- 表：第二个键值对标识 Amazon Keyspaces 要加密的表。键是 `aws:cassandra:tableName`。值为表的名称。

  ```
  "aws:cassandra:tableName": "<table-name>"
  ```

  例如：

  ```
  "aws:cassandra:tableName": "my_table"
  ```

- 账户：第三个键值对标识 AWS 账户。键是 `aws:cassandra:subscriberId`。该值为账户 ID。

  ```
  "aws:cassandra:subscriberId": "<account-id>"
  ```

  例如：

  ```
  "aws:cassandra:subscriberId": "111122223333"
  ```

步骤 6：使用 AWS CloudTrail 配置监控

如果您使用客户自主管理型密钥来保护 Amazon Keyspaces 表，则可以使用 AWS CloudTrail 日志跟踪 Amazon Keyspaces 代表您发送到 AWS KMS 的请求。

本节将讨论 `GenerateDataKey`、`DescribeKey`、`Decrypt` 和 `CreateGrant` 请求。此外，Amazon Keyspaces 还使用 RetireGrant 操作在您删除表时删除授权。

GenerateDataKey

Amazon Keyspaces 会创建一个唯一表密钥来加密静态数据。它将 GenerateDataKey 请求发送到指定表 KMS 密钥的 AWS KMS 中。

记录 `GenerateDataKey` 操作的事件与以下示例事件类似。用户是 Amazon Keyspaces 服务账户。参数包括客户自主管理型密钥的 Amazon 资源名称 (ARN)、需要 256 位密钥的密钥说明符以及标识键空间、表和 AWS 账户的加密上下文。

```
{
    "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 使用 DescribeKey 操作来确定所选 KMS 密钥是否存在于账户和区域中。

记录 `DescribeKey` 操作的事件与以下示例事件类似。用户是 Amazon Keyspaces 服务账户。参数包括客户自主管理型密钥的 ARN 和需要 256 位密钥的密钥说明符。

```
{
    "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

当您访问 Amazon Keyspaces 表时，Amazon Keyspaces 需要解密表密钥，以便解密层次结构中位于其下方的密钥。然后，解密表中的数据。为了解密表密钥，Amazon Keyspaces 会向 AWS KMS 发送一个 Decrypt 请求，该请求将指定表的 KMS 密钥。

记录 `Decrypt` 操作的事件与以下示例事件类似。用户是您的 AWS 账户 中正在访问表的委托人。参数包括加密表密钥（作为加密文字 blob）以及标识表和 AWS 账户的加密上下文。AWS KMS 将从加密文字中获取客户自主管理型密钥的 ID。

```
{
    "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

如果您使用客户自主管理型密钥来保护 Amazon Keyspaces 表，则 Amazon Keyspaces 会使用授权以允许服务执行持续数据保护和维护以及持久性任务。AWS 拥有的密钥不需要这些授权。

Amazon Keyspaces 创建的授权特定于表。CreateGrant 请求的委托人是创建了表的用户。

记录 `CreateGrant` 操作的事件与以下示例事件类似。参数包括表的客户自主管理型密钥的 ARN、被授权主体和停用主体（Amazon Keyspaces 服务）以及授权涵盖的操作。它还包括要求所有加密操作都使用指定加密上下文的约束。

```
{
    "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 中的传输中加密

Amazon Keyspaces 只接受使用传输层安全性协议 (TLS) 的安全连接。传输中加密可在数据进出 Amazon Keyspaces 时对其进行加密，从而提供额外一层数据保护。组织策略、行业或政府法规以及合规性要求通常要求使用传输中加密，以提高应用程序在通过网络传输数据时的数据安全性。

要了解如何使用 TLS 加密与 Amazon Keyspaces 的 `cqlsh` 连接，请参阅the section called “如何为 TLS 手动配置 `cqlsh` 连接”。要了解如何将 TLS 加密用于客户端驱动程序，请参阅the section called “使用 Cassandra 客户端驱动程序”。

Amazon Keyspaces 中的互联网络流量隐私

本主题介绍 Amazon Keyspaces（Apache Cassandra 兼容）如何保护本地应用程序与 Amazon Keyspaces 之间的连接，以及 Amazon Keyspaces 与同一 AWS 区域内的其他 AWS 资源之间的连接。

服务与本地客户端和应用之间的流量

私有网络和 AWS 之间有两种连接方式：

- AWS Site-to-Site VPN 连接。有关更多信息，请参阅 AWS Site-to-Site VPN 用户指南的 什么是 AWS Site-to-Site VPN？。
- AWS Direct Connect 连接。有关更多信息，请参阅 AWS Direct Connect 用户指南的 什么是 AWS Direct Connect？。

作为一项托管式服务，Amazon Keyspaces（Apache Cassandra 兼容）受 AWS 全球网络安全保护。有关 AWS 安全服务以及 AWS 如何保护基础架构的信息，请参阅 AWS 云安全。要按照基础设施安全最佳实践设计您的 AWS 环境，请参阅《安全性支柱 AWS Well-Architected Framework》中的 基础设施保护。

您可以使用 AWS 发布的 API 调用，通过网络访问 Amazon Keyspaces。客户端必须支持以下内容：

- 传输层安全性协议（TLS） 我们要求使用 TLS 1.2，建议使用 TLS 1.3。

- 具有完全向前保密（PFS）的密码套件，例如 DHE（Ephemeral Diffie-Hellman）或 ECDHE（Elliptic Curve Ephemeral Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

Amazon Keyspaces 支持两种对客户端请求进行身份验证的方法。第一种方法使用特定于服务的凭证，这种凭证是为特定 IAM 用户生成的基于密码的凭证。您可以使用 IAM 控制台、AWS CLI、或 AWS API 来创建和管理密码。有关更多信息，请参阅将 IAM 与 Amazon Keyspaces 结合使用。

第二种方法使用适用于 Cassandra 的开源 DataStax Java 驱动程序的身份验证插件。这一插件让 IAM 用户、角色和联合身份能够使用AWS签名版本 4 流程 (Sigv4) 向 Amazon Keyspaces（Apache Cassandra 兼容）API 请求添加身份验证信息。有关更多信息，请参阅the section called “用于 AWS 身份验证的 IAM 证书”。

同一区域中 AWS 资源之间的流量

接口 VPC 端点可以在 Amazon VPC 中运行的虚拟私有云 (VPC) 与 Amazon Keyspaces 之间实现私有通信。接口 VPC 终端节点由 AWS PrivateLink 支持，这是一种实现 VPC 与 AWS 服务之间私有通信的 AWS 服务。AWS PrivateLink 通过对 VPC 中的私有 IP 使用弹性网络接口启用此功能，以便网络流量不会离开 Amazon 网络。接口 VPC 终端节点不需要 Internet 网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。有关更多信息，请参阅 Amazon Virtual Private Cloud 和 接口 VPC 端点 (AWS PrivateLink)。有关示例策略，请参阅 the section called “将接口 VPC 端点用于 Amazon Keyspaces”。

AWS Identity and Access Management 适用于 Amazon Keyspaces

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以通过身份验证（登录）和授权（具有权限）使用 Amazon Keyspaces 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- Amazon Keyspaces 如何与 IAM 配合使用

- Amazon Keyspaces 基于身份的策略示例
- 适用于 Amazon Keyspaces 的 AWS 托管式策略
- Amazon Keyspaces 身份和访问问题排查
- 对 Amazon Keyspaces 使用服务相关角色

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 Amazon Keyspaces 中所做的工作。

服务用户：如果您使用 Amazon Keyspaces 服务来完成工作，您的管理员将为您提供所需的凭证和权限。随着您使用 Amazon Keyspaces 的更多功能来完成工作，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Amazon Keyspaces 中的功能，请参阅 Amazon Keyspaces 身份和访问问题排查。

服务管理员：如果您负责公司的 Amazon Keyspaces 资源，您可能具有 Amazon Keyspaces 的完全访问权限。您有责任确定您的服务用户应访问哪些 Amazon Keyspaces 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关贵公司如何将 IAM 与 Amazon Keyspaces 搭配使用的更多信息，请参阅 Amazon Keyspaces 如何与 IAM 配合使用。

IAM 管理员：如果您是 IAM 管理员，您可能需要详细了解如何编写策略以管理对 Amazon Keyspaces 的访问。要查看可在 IAM 中使用的 Amazon Keyspaces 基于身份的策略示例，请参阅 Amazon Keyspaces 基于身份的策略示例。

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的 多重身份验证 和《IAM 用户指南》中的 在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以根用户身份登录的任务的完整列表，请参阅 IAM 用户指南 中的 需要根用户凭证的任务。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，我们建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅 IAM 用户指南中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- Federated user access（联合用户访问）– 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的 为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon QLDB 中运行应用程序或在 Simple Storage Service（Amazon S3）中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅 IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。管理员随后可以向角色添加 IAM policy，用户可以代入角色。

IAM policy 定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅 IAM 用户指南中的创建 IAM policy。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Simple Storage Service（Amazon S3）存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅 Amazon Simple Storage Service 开发人员指南 中的访问控制列表（ACL）概览。

其它策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界 – 权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅 IAM 用户指南中的 IAM 实体的权限边界。
- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有特征，则可对任意或全部账户应用服务控制策略（SCP）。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅 AWS Organizations 用户指南中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合身份用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

Amazon Keyspaces 如何与 IAM 配合使用

在使用 IAM 管理对 Amazon Keyspaces 的访问之前，您应该了解哪些 IAM 功能可用于 Amazon Keyspaces。要全面了解 Amazon Keyspaces 和其他 AWS 服务如何与 IAM 配合使用，请参阅 IAM 用户指南中与 IAM 配合使用的AWS 服务。

主题

- Amazon Keyspaces 基于身份的策略
- Amazon Keyspaces 基于资源的策略
- 基于 Amazon Keyspaces 标签的授权
- Amazon Keyspaces IAM 角色

Amazon Keyspaces 基于身份的策略

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。Amazon Keyspaces 支持特定的操作、资源和条件键。要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素参考。

要查看 Amazon Keyspaces 服务特定的资源和操作以及可用于 IAM 权限策略的条件上下文键，请参阅《服务授权参考》中的 Amazon Keyspaces（Apache Cassandra 兼容）的操作、资源和条件键。

操作

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源 执行操作，以及在什么 条件 下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

Amazon Keyspaces 中的策略操作在操作前面使用以下前缀：`cassandra:`。例如，要授予某人使用 Amazon Keyspaces CREATE CQL 语句创建 Amazon Keyspaces 键空间的权限，您应将 `cassandra:Create` 操作纳入其策略中。策略语句必须包含 `Action` 或 `NotAction` 元素。Amazon Keyspaces 定义了一组自己的操作，以描述您可以使用该服务执行的任务。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": [
      "cassandra:CREATE",
      "cassandra:MODIFY"
          ]
```

要查看 Amazon Keyspaces 操作的列表，请参阅《服务授权参考》中的 Amazon Keyspaces（Apache Cassandra 兼容）定义的操作。

资源

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源 执行操作，以及在什么条件 下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

在 Amazon Keyspaces 中，键空间和表可以用于 IAM 权限的 Resource 元素中。

Amazon Keyspaces 键空间资源具有以下 ARN：

```
arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/${KeyspaceName}/
```

Amazon Keyspaces 表资源具有以下 ARN：

```
arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/${KeyspaceName}/table/
${tableName}
```

有关 ARN 格式的更多信息，请参阅 A mazon 资源名称 (ARN) 和 AWS 服务命名空间。

例如，要在语句中指定 mykeyspace 键空间，请使用以下 ARN：

```
"Resource": "arn:aws:cassandra:us-east-1:123456789012:/keyspace/mykeyspace/"
```

要指定属于特定账户的所有键空间，请使用通配符 (*)：

```
"Resource": "arn:aws:cassandra:us-east-1:123456789012:/keyspace/*"
```

无法对特定资源执行某些 Amazon Keyspaces 操作，例如，用于创建资源的操作。在这些情况下，您必须使用通配符 （*)。

```
"Resource": "*"
```

要使用标准驱动程序以编程方式连接到 Amazon Keyspaces，主体必须具有对系统表的 SELECT 访问权限，因为大多数驱动程序会在连接时读取系统键空间/表。例如，要向 IAM 用户授予 mykeyspace

中 mytable 的 SELECT 权限，主体必须具有读取 mytable 和 system keyspace 的权限。要在单个语句中指定多个资源，请使用逗号分隔 ARN。

```
"Resource": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
```

要查看 Amazon Keyspaces 资源类型及其 ARN 的列表，请参阅《服务授权参考》中的 Amazon Keyspaces（Apache Cassandra 兼容）定义的资源。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定义的操作。

条件键

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅 IAM 用户指南中的 IAM policy 元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

Amazon Keyspaces 定义了自己的一组条件键，还支持使用一些全局条件键。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

所有 Amazon Keyspaces 操作都支持 aws:RequestTag/${TagKey}、aws:ResourceTag/${TagKey} 和 aws:TagKeys 条件键。有关更多信息，请参阅 the section called “ 基于标签的 Amazon Keyspaces 资源访问”。

要查看 Amazon Keyspaces 条件键的列表，请参阅《服务授权参考》中的 Amazon Keyspaces（Apache Cassandra 兼容）的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon Keyspaces（Apache Cassandra 兼容）定义的操作。

示例

要查看 Amazon Keyspaces 基于身份的策略的示例，请参阅Amazon Keyspaces 基于身份的策略示例。

Amazon Keyspaces 基于资源的策略

Amazon Keyspaces 不支持基于资源的策略。要查看详细的基于资源的策略页面示例，请参阅 https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html。

基于 Amazon Keyspaces 标签的授权

您可以使用标签管理对 Amazon Keyspaces 资源的访问。要管理基于标签的资源访问，您可以使用 `cassandra:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。有关标记 Amazon Keyspaces 资源的更多信息，请参阅the section called “使用标签”。

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅 基于标签的 Amazon Keyspaces 资源访问。

Amazon Keyspaces IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的实体。

将临时凭证用于 Amazon Keyspaces

您可以使用临时凭证进行联合身份登录，来担任 IAM 角色或担任跨账户角色。您可以通过调用AssumeRole或之类的 AWS STS API 操作来获取临时安全证书GetFederationToken。

Amazon Keyspaces 支持使用 Github 存储库中提供的 AWS 签名版本 4 (Sigv4) 身份验证插件的临时证书，适用于以下语言：

- 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。

有关实现身份验证插件以编程方式访问 Amazon Keyspaces 的示例和教程，请参阅。the section called “使用 Cassandra 客户端驱动程序”

服务相关角色

服务相关角色允许 AWS 服务访问其他服务中的资源以代表您完成操作。服务相关角色显示在 IAM 账户中，并归该服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理 Amazon Keyspaces 服务相关角色的详细信息，请参阅the section called “使用服务相关角色”。

服务角色

Amazon Keyspaces 不支持服务角色。

Amazon Keyspaces 基于身份的策略示例

默认情况下，IAM 用户和角色没有创建或修改 Amazon Keyspaces 资源的权限。他们也无法使用控制台、CQLSH 或 AWS API 执行任务。 AWS CLI IAM 管理员必须创建 IAM 策略，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的 IAM 用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的在 JSON 选项卡上创建策略。

主题

- 策略最佳实践
- 使用 Amazon Keyspaces 控制台
- 允许用户查看他们自己的权限
- 访问 Amazon Keyspaces 表
- 基于标签的 Amazon Keyspaces 资源访问

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon Keyspaces 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略 或 工作职能的AWS 托管式策略。

- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM policy，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM policy 语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

使用 Amazon Keyspaces 控制台

Amazon Keyspaces 不需要特定权限即可访问亚马逊密钥空间控制台。您至少需要具有只读权限才能列出和查看您的 Amazon Keyspaces 资源的详细信息。 AWS 账户如果您创建的基于身份的策略比所需的最低权限更严格，则无法为具有该策略的实体（IAM 用户或角色）正常运行控制台。

两个 AWS 托管策略可供实体使用 Amazon Keyspaces 控制台访问权限。

- AmazonKeyspacesReadOnlyAccess_v2 — 此政策授予对 Amazon Keyspaces 的只读访问权限。
- AmazonKeyspacesFullAccess— 此政策授予使用 Amazon Keyspaces 的权限，并具有对所有功能的完全访问权限。

有关 Amazon Keyspaces 托管式策略的更多信息，请参阅 the section called “AWS 托管式策略”。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
```

```
    "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": "*"
        }
    ]
}
```

访问 Amazon Keyspaces 表

以下是授予对 Amazon Keyspaces 系统表的只读 (SELECT) 访问权限的策略示例。对于所有示例，请将 Amazon 资源名称 (ARN) 中的地区和账户 ID 替换为您自己的地区和账户 ID。

> **Note**
>
> 要使用标准驱动程序进行连接，用户必须至少具有对系统表的 SELECT 访问权限，因为大多数驱动程序在连接时读取系统键空间/表。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

以下示例策略在密钥空间mykeyspace中添加了对用户表mytable的只读访问权限。

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

以下示例策略分配对用户表的读/写访问权限和对系统表的读取访问权限。

> **Note**
>
> 系统表始终是只读的。

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

以下示例策略允许用户在键空间 mykeyspace 中创建表。

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

基于标签的 Amazon Keyspaces 资源访问

您可以在基于身份的策略中使用条件，以便基于标签控制对 Amazon Keyspaces 资源的访问。这些策略控制账户中键空间和表的可见性。请注意，与通过 Cassandra 驱动程序和开发者工具调用

Cassandra 查询语言 (CQL) API 相比，使用 AWS SDK 发出请求时，基于标签的系统表权限的行为有所不同。

- 要在使用基于标签的访问权限时使用 AWS SDK 发出 `List` 和 `Get` 请求，调用者需要具有对系统表的读取权限。例如，需要 `Select` 操作权限才能通过 `GetTable` 操作从系统表读取数据。如果调用者对特定表只有基于标签的访问权限，则需要对系统表进行额外访问的操作将失败。
- 为了与既定的 Cassandra 驱动程序行为兼容，当您通过 Cassandra 驱动程序和开发人员工具使用 Cassandra 查询语言 (CQL) API 调用对系统表执行操作时，系统不会强制实施基于标签的授权策略。

以下示例说明如何创建一个策略，该策略授予用户查看表的权限（如果表的 `Owner` 包含该用户的用户名的值）。在此示例中，您还授予对系统表的读取权限。

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

您可以将该策略附加到您账户中的 IAM 用户。如果一个名为 `richard-roe` 的用户尝试查看 Amazon Keyspaces 表，则您必须为该表添加 `Owner=richard-roe` 或 `owner=richard-roe` 标签。否则，他将被拒绝访问。条件标签键 `Owner` 匹配 `Owner` 和 `owner`，因为条件键名称不区分大小写。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。

以下策略授予用户创建带有标签的表的权限（如果表的 `Owner` 包含该用户的用户名的值）。

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

适用于 Amazon Keyspaces 的 AWS 托管式策略

AWS 托管式策略是由 AWS 创建和管理的独立策略。AWS 托管式策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住，AWS 托管式策略可能不会为您的特定使用场景授予最低权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管式策略中定义的权限。如果 AWS 更新在 AWS 托管式策略中定义的权限，则更新会影响该策略所附加到的所有主体身份（用户、组和角色）。当新的 AWS 服务启动或新的 API 操作可用于现有服务时，AWS 最有可能更新 AWS 托管式策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

AWS 托管式策略：AmazonKeyspacesReadOnlyAccess_v2

您可以将 AmazonKeyspacesReadOnlyAccess_v2 策略附加得到 IAM 身份。

该策略授予对 Amazon Keyspaces 的只读访问权限，并包括通过私有 VPC 端点进行连接时所需的权限。

权限详细信息

此策略包含以下权限。

- Amazon Keyspaces：提供对 Amazon Keyspaces 的只读访问权限。
- Application Auto Scaling：允许主体从 Application Auto Scaling 查看配置。这是必需权限，以便用户可以查看附加到表的自动扩展策略。
- CloudWatch：允许主体查看在 CloudWatch 中配置的指标数据和警报。这是必需权限，以便用户可以查看可计费表大小以及已为表配置的 CloudWatch 警报。
- AWS KMS：允许主体查看 AWS KMS 中配置的密钥。这是必需权限，以便用户可以查看他们在其账户中创建和管理的 AWS KMS 密钥，从而确认分配给 Amazon Keyspaces 的密钥是已启用的对称加密密钥。
- Amazon EC2：允许主体通过 VPC 端点连接到 Amazon Keyspaces，以查询 Amazon EC2 实例上的 VPC，从而获取端点和网络接口信息。必须具有对 Amazon EC2 实例的这种只读访问权限，这样 Amazon Keyspaces 才能在用于连接负载均衡的 system.peers 表中查找和存储可用的接口 VPC 端点。

要查看 JSON 格式的策略，请参阅 AmazonKeyspacesReadOnlyAccess_v2。

AWS 托管式策略：AmazonKeyspacesReadOnlyAccess

您可以将 AmazonKeyspacesReadOnlyAccess 策略附加得到 IAM 身份。

此策略将授予对 Amazon Keyspaces 的只读访问权限。

权限详细信息

此策略包含以下权限。

- `Amazon Keyspaces`：提供对 Amazon Keyspaces 的只读访问权限。
- `Application Auto Scaling`：允许主体从 Application Auto Scaling 查看配置。这是必需权限，以便用户可以查看附加到表的自动扩展策略。
- `CloudWatch`：允许主体查看在 CloudWatch 中配置的指标数据和警报。这是必需权限，以便用户可以查看可计费表大小以及已为表配置的 CloudWatch 警报。
- `AWS KMS`：允许主体查看 AWS KMS 中配置的密钥。这是必需权限，以便用户可以查看他们在其账户中创建和管理的 AWS KMS 密钥，从而确认分配给 Amazon Keyspaces 的密钥是已启用的对称加密密钥。

要查看 JSON 格式的策略，请参阅 AmazonKeyspacesReadOnlyAccess。

AWS 托管式策略：AmazonKeyspacesFullAccess

您可以将 `AmazonKeyspacesFullAccess` 策略附加得到 IAM 身份。

此策略授予管理权限，允许您的管理员不受限制地访问 Amazon Keyspaces。

权限详细信息

此策略包含以下权限。

- `Amazon Keyspaces`：允许主体访问任何 Amazon Keyspaces 资源并执行所有操作。
- `Application Auto Scaling`：允许主体创建、查看和删除 Amazon Keyspaces 表的自动扩展策略。这是必需权限，以便管理员可以管理 Amazon Keyspaces 表的自动扩展策略。
- `CloudWatch`：让主体能够查看可计费表大小以及创建、查看和删除 Amazon Keyspaces 自动扩展策略的 CloudWatch 警报。这是必需权限，以便管理员可以查看可计费表大小并创建 CloudWatch 控制面板。
- `IAM`：允许 Amazon Keyspaces 在启用以下功能时自动使用 IAM 创建服务相关角色：
  - `Application Auto Scaling`：当管理员为表启用 Application Auto Scaling 时，Amazon Keyspaces 会创建一个服务相关角色来代表您执行自动扩展操作。
  - `Amazon Keyspaces Multi-Region Replication`：当管理员创建多区域键空间时，系统会自动创建一个服务相关角色来代表您将数据复制到选定的 AWS 区域。

  有关服务相关角色的更多信息，请参阅 the section called “使用服务相关角色”。

- AWS KMS：允许主体查看 AWS KMS 中配置的密钥。这是必需权限，以便用户可以查看他们在其账户中创建和管理的 AWS KMS 密钥，从而确认分配给 Amazon Keyspaces 的密钥是已启用的对称加密密钥。
- Amazon EC2：允许主体通过 VPC 端点连接到 Amazon Keyspaces，以查询 Amazon EC2 实例上的 VPC，从而获取端点和网络接口信息。必须具有对 Amazon EC2 实例的这种只读访问权限，这样 Amazon Keyspaces 才能在用于连接负载均衡的 `system.peers` 表中查找和存储可用的接口 VPC 端点。

要查看 JSON 格式的策略，请参阅 AmazonKeyspacesFullAccess。

适用于 Amazon Keyspaces 的 AWS 托管式策略的更新

查看有关适用于 Amazon Keyspaces 的 AWS 托管式策略更新的详细信息（自此服务开始跟踪这些更改以来）。要获得有关此页面更改的自动提示，请订阅 文档历史记录 页面上的 RSS 源。

更改	说明	日期
AmazonKeyspacesFullAccess：对现有策略的更新	Amazon Keyspaces 为通过接口 VPC 端点连接到 Amazon Keyspaces 的客户端添加了新的只读权限，以便访问 Amazon EC2 实例来查询网络信息。 Amazon Keyspaces 将可用的接口 VPC 端点存储在 `system.peers` 表中，以实现连接负载均衡。有关更多信息，请参阅 the section called “使用接口 VPC 端点”。	2023 年 10 月 3 日
AmazonKeyspacesReadOnlyAccess_v2：新策略	Amazon Keyspaces 创建了一个新策略，为通过接口 VPC 端点连接到 Amazon Keyspaces	2023 年 9 月 12 日

更改	说明	日期
	的客户端添加只读权限，以便访问 Amazon EC2 实例来查找网络信息。 Amazon Keyspaces 将可用的接口 VPC 端点存储在 `system.peers` 表中，以实现连接负载均衡。有关更多信息，请参阅 the section called “使用接口 VPC 端点”。	
AmazonKeyspacesFullAccess：对现有策略的更新	Amazon Keyspaces 添加了新权限，以允许 Amazon Keyspaces 在管理员创建多区域键空间时创建服务相关角色。 Amazon KeysSpaces 使用服务相关角色代表您执行数据复制任务。有关更多信息，请参阅 the section called “多区域复制”。	2023 年 6 月 5 日
AmazonKeyspacesReadOnlyAccess：对现有策略的更新	Amazon KeysSpaces 添加了新的权限，以允许用户使用 CloudWatch 查看表的可计费大小。 Amazon Keyspaces 与 Amazon CloudWatch 集成，允许您监控可计费表大小。有关更多信息，请参阅 the section called “Amazon Keyspaces 指标与维度”。	2022 年 7 月 7 日

更改	说明	日期
AmazonKeyspacesFullAccess：对现有策略的更新	Amazon KeysSpaces 添加了新的权限，以允许用户使用 CloudWatch 查看表的可计费大小。 Amazon Keyspaces 与 Amazon CloudWatch 集成，允许您监控可计费表大小。有关更多信息，请参阅 the section called “Amazon Keyspaces 指标与维度”。	2022 年 7 月 7 日
AmazonKeyspacesReadOnlyAccess：对现有策略的更新	Amazon Keyspaces 添加了新的权限，以允许用户查看已为 Amazon Keyspaces 静态加密配置的 AWS KMS 密钥。 Amazon Keyspaces 静态加密与 AWS KMS 集成，可保护和管理用于加密静态数据的加密密钥。为便于查看为 Amazon Keyspaces 配置的 AWS KMS 密钥，我们添加了只读权限。	2021 年 6 月 1 日
AmazonKeyspacesFullAccess：对现有策略的更新	Amazon Keyspaces 添加了新的权限，以允许用户查看已为 Amazon Keyspaces 静态加密配置的 AWS KMS 密钥。 Amazon Keyspaces 静态加密与 AWS KMS 集成，可保护和管理用于加密静态数据的加密密钥。为便于查看为 Amazon Keyspaces 配置的 AWS KMS 密钥，我们添加了只读权限。	2021 年 6 月 1 日

更改	说明	日期
Amazon Keyspaces 开始跟踪更改	Amazon Keyspaces 开始跟踪其 AWS 托管式策略的更改。	2021 年 6 月 1 日

Amazon Keyspaces 身份和访问问题排查

您可以使用以下信息，帮助诊断和修复在使用 Amazon Keyspaces 和 IAM 时可能遇到的常见问题。

主题

我无权在 Amazon Keyspaces 中执行操作

如果 AWS Management Console 告诉您您无权执行某项操作，则必须联系管理员寻求帮助。管理员是指提供用户名和密码的人员。

当 `mateojackson` IAM 用户尝试使用控制台查看#的详细信息但不具有 `cassandra:Select` 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 cassandra:Select on resource: mytable
```

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 `cassandra:Select` 操作访问 *`mytable`* 资源。

我修改了 IAM 用户或角色，但更改没有立即生效

对于已与 Amazon Keyspaces 建立连接的应用程序，IAM 政策更改最多可能需要 10 分钟才能生效。当应用程序建立新连接时，IAM 策略更改会立即生效。如果您对现有 IAM 用户或角色进行了修改，但修改尚未立即生效，请等待 10 分钟，或者断开连接并重新连接到 Amazon Keyspaces。

我无法使用 Amazon Keyspaces point-in-time 恢复 (PITR) 来恢复表

如果您正在尝试使用恢复 (PITR) point-in-time 恢复 Amazon Keyspaces 表，但看到恢复过程已开始但未成功完成，则可能没有配置恢复过程所需的所有必要权限。您必须联系您的管理员寻求帮助，并要求该人员更新您的策略，以允许您还原 Amazon Keyspaces 中的表。

除了用户权限之外，Amazon Keyspaces 可能还需要在还原过程中代表您的主体执行操作的权限。如果表使用客户自主管理型密钥加密，或者您使用限制传入流量的 IAM 策略，就会出现这种情况。例如，如果您在 IAM 策略中使用条件键将源流量限制到特定端点或 IP 范围，则还原操作会失败。要允许 Amazon Keyspaces 代表您的主体执行表恢复操作，必须在 IAM 策略中添加 `aws:ViaAWSService` 全局条件键。

有关恢复表所需的权限的更多信息，请参阅the section called “还原权限。”。

我无权执行 iam：PassRole

如果您收到一个错误消息，其中指明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 Amazon Keyspaces。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的 IAM 用户尝试使用控制台在 Amazon Keyspaces 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我是管理员并希望允许其他人访问 Amazon Keyspaces

要允许其他人访问 Amazon Keyspaces，您必须为需要访问权限的人员或应用程序创建一个 IAM 实体（用户或角色）。它们将使用该实体的凭证访问 AWS。然后，您必须将策略附加到实体，以便在 Amazon Keyspaces 中为他们（它们）授予正确的权限。

要立即开始使用，请参阅《IAM 用户指南》中的创建您的第一个 IAM 委派用户和组。

我想允许我以外的人访问我的 Amazon Keyspaces 资源 AWS 账户

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon Keyspaces 是否支持这些功能，请参阅 Amazon Keyspaces 如何与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过联合身份验证提供访问权限，请参阅 IAM 用户指南中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

对 Amazon Keyspaces 使用服务相关角色

Amazon Keyspaces（Apaces Cassandra 兼容）使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon Keyspaces 直接相关。服务相关角色是由 Amazon Keyspaces 预定义的，并包含服务代表您调用其他 AWS 服务所需的所有权限。

主题

- 使用角色实现 Amazon Keyspaces 应用程序自动扩缩
- 使用角色进行 Amazon Keyspaces 多区域复制

使用角色实现 Amazon Keyspaces 应用程序自动扩缩

Amazon Keyspaces（Apaces Cassandra 兼容）使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon Keyspaces 直接相关。服务相关角色是由 Amazon Keyspaces 预定义的，并包含服务代表您调用其他 AWS 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon Keyspaces，因为您不必手动添加必要的权限。Amazon Keyspaces 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon Keyspaces 可以代入该角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其它 IAM 实体。

只有在首先删除服务相关角色的相关资源后，才能删除该角色。这将保护您的 Amazon Keyspaces 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的 AWS 服务，并查找服务相关角色列中显示为是的服务。选择是，可转到查看该服务的服务相关角色文档的链接。

Amazon Keyspaces 的服务相关角色权限

Amazon Keyspaces 使用名为AWSServiceRoleForApplicationAutoScaling_CassandraTable的服务相关角色允许 Auto Scaling 代表您调用亚马逊密钥空间和亚马逊。 CloudWatch

AWSServiceRoleForApplicationAutoScaling_CassandraTable 服务相关角色信任以下服务来代入该角色：

- `cassandra.application-autoscaling.amazonaws.com`

角色权限策略允许 Application Auto Scaling 在指定的 Amazon Keyspaces 资源上完成以下操作：

- 操作：`arn:*:cassandra:*:*:/keyspace/system/table/*` 上的 `cassandra:Select`
- 操作：资源 `arn:*:cassandra:*:*:/keyspace/system_schema/table/*` 上的 `cassandra:Select`
- 操作：资源 `arn:*:cassandra:*:*:/keyspace/system_schema_mcs/table/*` 上的 `cassandra:Select`
- 操作：资源 `arn:*:cassandra:*:*:"*"` 上的 `cassandra:Alter`

为 Amazon Keyspaces 创建服务相关角色

您无需手动为 Amazon Keyspaces 自动扩展创建服务相关角色。当您使用AWS Management Console、CQL、或 AWS API 在表上启用 Amazon Keyspaces 自动扩展时，Application Auto Scaling 会为您创建服务相关角色。AWS CLI

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您为表启用 Amazon Keyspaces 自动扩展时，Application Auto Scaling 会再次为您创建服务相关角色。

> **Important**
>
> 如果您在其他使用此角色支持的功能的服务中完成某个操作，此服务相关角色可以出现在您的账户中。要了解更多信息，请参阅我的 AWS 账户中出现新角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您为表启用 Amazon Keyspaces 自动应用程序扩展时，Application Auto Scaling 会再次为您创建服务相关角色。

为 Amazon Keyspaces 编辑服务相关角色

Amazon Keyspaces 不允许您编辑 AWSServiceRoleForApplicationAutoScaling_CassandraTable 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除适用于 Amazon Keyspaces 的服务相关角色

如果您不再需要使用某个需要服务相关角色的特征或服务，我们建议您删除该角色。这样您就没有未被主动监控或维护的未使用实体。但是，您必须首先在所有 AWS 区域 区域中对账户中的所有表禁用自动扩展，然后才能手动删除服务相关角色。要在 Amazon Keyspaces 表上禁用自动扩展，请参阅修改或禁用 Amazon Keyspaces 自动扩展设置。

> **Note**
>
> 如果在您尝试修改资源时 Amazon Keyspaces 自动扩展使用该角色，则取消注册可能会失败。如果发生这种情况，请等待几分钟后重试。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台AWS CLI、或 AWS API 删除 AWSServiceRoleForApplicationAutoScaling_CassandraTable 服务相关角色。有关更多信息，请参见《IAM 用户指南》中的删除服务相关角色。

> **Note**
>
> 要删除 Amazon Keyspaces 自动扩展所使用的服务相关角色，您必须首先禁用账户中所有表的自动扩展。

Amazon Keyspaces 服务相关角色支持的区域

Amazon Keyspaces 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 Amazon Keyspaces 服务端点。

使用角色进行 Amazon Keyspaces 多区域复制

Amazon Keyspaces（Apaces Cassandra 兼容）使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon Keyspaces 直接相关。服务相关角色是由 Amazon Keyspaces 预定义的，并包含服务代表您调用其他 AWS 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon Keyspaces，因为您不必手动添加必要的权限。Amazon Keyspaces 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon Keyspaces 可以代入该角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其它 IAM 实体。

只有在首先删除服务相关角色的相关资源后，才能删除该角色。这将保护您的 Amazon Keyspaces 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的 AWS 服务，并查找服务相关角色列中显示为是的服务。选择是，可转到查看该服务的服务相关角色文档的链接。

Amazon Keyspaces 的服务相关角色权限

Amazon Keyspaces 使用名为的服务相关角色AWSServiceRoleForAmazonKeyspacesReplication允许 Amazon Keyspaces 代表您复制对多区域表的所有副本的写入。

AWSServiceRoleForAmazonKeyspacesReplication 服务相关角色信任以下服务来代入该角色：

- `replication.cassandra.amazonaws.com`

名为的角色权限策略 KeyspacesReplicationServiceRolePolicy 允许 Amazon Keyspaces 完成以下操作：

- 操作：`cassandra:Select`
- 操作：`cassandra:SelectMultiRegionResource`
- 操作：`cassandra:Modify`
- 操作：`cassandra:ModifyMultiRegionResource`

尽管 Amazon Keyspaces 服务相关角色 AWSServiceRoleForAmazonKeyspacesReplication 为策略中指定的亚马逊资源名称 (ARN) “arn: *” 提供了权限：“操作：”，但亚马逊密钥空间会提供您账户的 ARN。

您必须配置允许用户、组或角色创建、编辑或删除服务相关角色的权限。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon Keyspaces 创建服务相关角色

您不能手动创建服务相关角色。当您在 AWS Management Console、AWS CLI 或 AWS API 中创建多区域键空间时，Amazon Keyspaces 会为您创建服务相关角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建多区域键空间时，Amazon Keyspaces 将再次为您创建服务相关角色。

为 Amazon Keyspaces 编辑服务相关角色

Amazon Keyspaces 不允许您编辑 AWSServiceRoleForAmazonKeyspacesReplication 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除适用于 Amazon Keyspaces 的服务相关角色

如果不再需要使用某个需要服务相关角色的特征或服务，建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须首先在所有 AWS 区域中对账户中的所有表禁用自动扩展，然后才能手动删除服务相关角色。

清除服务相关角色

必须先删除服务相关角色使用的多区域键空间和表，然后才能使用 IAM 删除该角色。

> **Note**
>
> 如果在您试图删除资源时，Amazon Keyspaces 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

删除 AWSServiceRoleForAmazonKeyspacesReplication （控制台）使用的 Amazon Keyspaces 资源

1. 访问 https://console.aws.amazon.com/keyspaces/home，登录 AWS Management Console并打开 Amazon Keyspaces 控制台。
2. 从左侧面板中选择 Keyspaces。

3. 从列表中选择所有多区域键空间。
4. 选择删除确认删除操作，然后选择删除键空间。

您也可以使用以下任何一种方法以编程方式删除多区域键空间。

- Cassandra 查询语言 (CQL) ??? 语句。
- AWS CLI 的 delete-keyspace 操作。
- Amazon Keyspaces API 的DeleteKeyspace操作。

手动删除服务相关角色

使用 IAM 控制台、AWS CLI 或 AWS API 删除 AWSServiceRoleForAmazonKeyspacesReplication 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Amazon Keyspaces 服务相关角色支持的区域

Amazon Keyspaces 并非在提供该服务的每个区域中都支持使用服务相关角色。您可以在以下区域使用该 AWSServiceRoleForAmazonKeyspacesReplication 角色。

区域名称	区域标识	Amazon Keyspaces 支持
美国东部（弗吉尼亚州北部）	us-east-1	是
美国东部（俄亥俄州）	us-east-2	是
美国西部（北加利福尼亚）	us-west-1	是
美国西部（俄勒冈州）	us-west-2	是
亚太地区（孟买）	ap-south-1	是
亚太地区（大阪）	ap-northeast-3	是
亚太地区（首尔）	ap-northeast-2	是
亚太地区（新加坡）	ap-southeast-1	是
亚太地区（悉尼）	ap-southeast-2	是

区域名称	区域标识	Amazon Keyspaces 支持
亚太地区（东京)	ap-northeast-1	是
加拿大（中部）	ca-central-1	是
欧洲地区（法兰克福）	eu-central-1	是
欧洲地区（爱尔兰）	eu-west-1	是
欧洲地区（伦敦）	eu-west-2	是
欧洲地区（巴黎）	eu-west-3	是
南美洲（圣保罗）	sa-east-1	是
AWS GovCloud （美国东部）	us-gov-east-1	否
AWS GovCloud （美国西部）	us-gov-west-1	否

Amazon Keyspaces（Apache Cassandra 兼容）的合规性验证

作为多个 AWS 合规性计划的一部分，第三方审计员将评测 Amazon Keyspaces（Apache Cassandra 兼容）的安全性和合规性。其中包括：

- ISO/IEC 27001:2013、27017:2015、27018:2019 和 ISO/IEC 9001:2015。有关更多信息，请参阅 AWS ISO 和 CSA STAR 认证和服务。
- 系统和组织控制 (SOC)
- 支付卡行业 (PCI)
- 联邦风险与授权管理项目 (FedRAMP)（高）
- 健康保险流通与责任法案 (HIPAA)

要了解某个 AWS 服务 是否在特定合规性计划范围内，请参阅合规性计划范围内的 AWS 服务 ，然后选择您感兴趣的合规性计划。有关常规信息，请参阅AWS 合规性计划。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅在 AWS Artifact 中下载报告、。

您使用 AWS 服务 的合规性责任取决于数据的敏感度、贵公司的合规性目标以及适用的法律法规。AWS 提供以下资源来帮助满足合规性：

- 安全性与合规性快速入门指南——这些部署指南讨论了架构注意事项，并提供了在 AWS 上部署以安全性和合规性为重点的基准环境的步骤。
- Amazon Web Services 上的 HIPAA 安全性和合规性架构设计 – 该白皮书介绍了公司如何使用 AWS 创建符合 HIPAA 标准的应用程序。

> **Note**
>
> 并非所有 AWS 服务 都符合 HIPAA 要求。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合规性资源 – 此业务手册和指南集合可能适用于您的行业和位置。
- AWS 客户合规指南：从合规角度了解责任共担模式。这些指南总结了保护 AWS 服务 的最佳实践，并将指南映射到跨多个框架的安全控制，包括美国国家标准与技术研究院（NIST）、支付卡行业安全标准委员会（PCI）和国际标准化组织（ISO）。
- AWS Config 开发人员指南中的使用规则评估资源 – 此 AWS Config 服务评测您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub——此 AWS 服务 向您提供 AWS 中安全状态的全面视图。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实操。有关受支持服务及控制的列表，请参阅 Security Hub 控制参考。
- AWS Audit Manager：此 AWS 服务 可帮助您持续审核您的 AWS 使用情况，以简化管理风险以及与相关法规和行业标准的合规性的方式。

Amazon Keyspaces 的韧性和灾难恢复

AWS全球基础设施围绕AWS 区域和可用区构建。AWS 区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

为了实现持久性和高可用性，Amazon Keyspaces 会在同一个 AWS 区域中的多个 AWS 可用区自动复制数据三次。

有关 AWS 区域 和可用区的更多信息，请参阅 AWS 全球基础设施。

除了 AWS 全球基础设施之外，Amazon Keyspaces 还提供多种功能，以帮助支持您的数据恢复能力和备份需求。

多区域复制

Amazon Keyspaces 提供多区域复制功能，适用于需要跨更大的地理距离复制数据或应用程序的用例。您最多可以跨您选择的六个不同的 AWS 区域复制您的 Amazon Keyspaces 表。有关更多信息，请参阅多区域复制。

时间点故障恢复 (PITR)

PITR 通过持续备份 Amazon Keyspaces 表数据，保护它免受意外写入或删除操作的影响。有关更多信息，请参阅 Point-in-time recovery for Amazon Keyspaces。

Amazon Keyspaces 中的基础设施安全性

作为一项托管式服务，Amazon Keyspaces（Apache Cassandra 兼容）受 AWS 全球网络安全保护。有关 AWS 安全服务以及 AWS 如何保护基础架构的信息，请参阅 AWS 云安全。要按照基础设施安全最佳实践设计您的 AWS 环境，请参阅《安全性支柱 AWS Well-Architected Framework》中的 基础设施保护。

您可以使用 AWS 发布的 API 调用，通过网络访问 Amazon Keyspaces。客户端必须支持以下内容：

- 传输层安全性协议（TLS） 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密（PFS）的密码套件，例如 DHE（Ephemeral Diffie-Hellman）或 ECDHE（Elliptic Curve Ephemeral Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

Amazon Keyspaces 支持两种对客户端请求进行身份验证的方法。第一种方法使用特定于服务的凭证，这种凭证是为特定 IAM 用户生成的基于密码的凭证。您可以使用 IAM 控制台、AWS CLI、或 AWS API 来创建和管理密码。有关更多信息，请参阅将 IAM 与 Amazon Keyspaces 结合使用。

第二种方法使用适用于 Cassandra 的开源 DataStax Java 驱动程序的身份验证插件。这一插件让 IAM 用户、角色和联合身份能够使用 AWS 签名版本 4 流程 (Sigv4) 向 Amazon Keyspaces（Apache Cassandra 兼容）API 请求添加身份验证信息。有关更多信息，请参阅 the section called “用于 AWS 身份验证的 IAM 证书”。

您可以使用接口 VPC 端点来防止 Amazon VPC 和 Amazon Keyspaces 之间的流量离开 Amazon 网络。接口 VPC 端点由 AWS PrivateLink 提供支持，后者是一种 AWS 技术，可将弹性网络接口与 Amazon VPC 中的私有 IP 结合使用，以实现 AWS 服务之间的私有通信。有关更多信息，请参阅 the section called “使用接口 VPC 端点”。

将 Amazon Keyspaces 与接口 VPC 端点结合使用

接口 VPC 端点可以在 Amazon VPC 中运行的虚拟私有云 (VPC) 与 Amazon Keyspaces 之间实现私有通信。接口 VPC 终端节点由提供支持 AWS PrivateLink，这是一项支持 VPC 和 AWS 服务之间私有通信的 AWS 服务。

AWS PrivateLink 通过在您的 VPC 中使用带有私有 IP 地址的 elastic network interface 来实现这一点，这样网络流量就不会离开亚马逊网络。接口 VPC 端点不需要互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。有关更多信息，请参阅 Amazon Virtual Private Cloud 和 接口 VPC 端点 (AWS PrivateLink)。

主题

- 将接口 VPC 端点用于 Amazon Keyspaces
- 使用接口 VPC 端点信息填充 system.peers 表条目
- 控制 Amazon Keyspaces 对接口 VPC 端点的访问
- 可用性
- VPC 终端节点策略和 Amazon Keyspaces point-in-time 恢复 (PITR)
- 常见错误和警告

将接口 VPC 端点用于 Amazon Keyspaces

您可以创建接口 VPC 端点，这样 Amazon Keyspaces 和 Amazon VPC 资源之间的流量就会通过接口 VPC 端点开始流动。要开始使用，请按照步骤创建接口端点。接下来，编辑与您在上一步中创建的终端节点关联的安全组，并为端口 9142 配置入站规则。有关详细信息，请参阅添加、删除和更新规则。

有关通过 VPC 终端节点配置与 Amazon Keyspaces 的连接的 step-by-step 教程，请参阅。the section called “连接 VPC 终端节点”要了解如何为与 VPC 中 AWS 账户 不同应用程序分开的 Amazon Keyspaces 资源配置跨账户访问权限，请参阅。the section called “跨账户访问权限”

使用接口 VPC 端点信息填充 `system.peers` 表条目

Apache Cassandra 驱动程序使用该 `system.peers` 表来查询有关集群的节点信息。Cassandra 驱动程序使用节点信息对连接进行负载均衡并重试操作。Amazon Keyspaces 会自动为通过公共端点连接的客户端填充 `system.peers` 表中的九个条目。

为了向通过接口 VPC 端点连接的客户端提供类似的功能，Amazon Keyspaces 会在您账户的 `system.peers` 表格中填充一个条目，用于显示 VPC 端点可用的每个可用区。要在 `system.peers` 表中查找和存储可用的接口 VPC 端点，Amazon Keyspaces 要求您向用于连接到 Amazon Keyspaces 的 IAM 实体授予访问权限，以查询您的 VPC 的端点和网络接口信息。

> **Important**
>
> 使用可用的接口 VPC 终点填充 `system.peers` 表可以改善负载均衡并增加读/写吞吐量。建议使用接口 VPC 端点访问 Amazon Keyspaces 的所有客户端都这样做，Apache Spark 也需要这样做。

要向用于连接 Amazon Keyspaces 的 IAM 实体授予权限以查找必要的接口 VPC 端点信息，可以更新现有的 IAM 角色或用户策略，或创建新的 IAM 策略，如以下示例所示。

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

> **Note**
>
> 托管式策略 AmazonKeyspacesReadOnlyAccess_v2 和 AmazonKeyspacesFullAccess 包含允许 Amazon Keyspaces 访问 Amazon EC2 实例以读取可用接口 VPC 端点信息所需的权限。

要确认策略设置是否正确，请查询 system.peers 表以查看网络信息。如果该 system.peers 表为空，则可能表示策略未成功配置，或者您已超出 DescribeNetworkInterfaces 和 DescribeVPCEndpoints API 操作的请求速率限额。DescribeVPCEndpoints 属于 Describe* 类别，被视为非变异操作。DescribeNetworkInterfaces 属于未过滤和未分页的非变异操作的子集，并且适用不同的限额。有关更多信息，请参阅 Amazon EC2 API 参考中的请求令牌桶大小和重新填充率。

如果您看到一张空表，请在几分钟后重试，以排除请求速率限额问题。要验证您是否正确配置了 VPC 端点，请参阅the section called “VPC 端点连接错误”。如果您的查询返回表中的结果，说明您的策略配置正确。

控制 Amazon Keyspaces 对接口 VPC 端点的访问

VPC 端点策略使您能够通过两种方式控制对资源的访问：

- IAM 策略 – 您可以控制允许通过特定 VPC 端点访问 Amazon Keyspaces 的请求、用户或组。您可以通过在附加到 IAM 用户、组或角色的策略中使用条件键来完成此操作。
- VPC 策略 – 您可以通过向 VPC 端点附加策略来控制哪些端点可以访问您的 Amazon Keyspaces 资源。要限制仅允许通过特定 VPC 终端节点的流量来访问特定键空间或表，请编辑限制资源访问的现有 IAM 策略并添加该 VPC 终端节点。

下面是访问 Amazon Keyspaces 资源的端点策略示例。

- IAM 策略示例：限制对特定 Amazon Keyspaces 表的所有访问，除非流量来自指定的 VPC 端点 – 此示例策略可以附加到 IAM 用户、角色或组。它限制对指定 Amazon Keyspaces 表的访问，除非传入流量来自指定的 VPC 端点。

```
{
   "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**
>
> 要限制对特定表的访问，还必须包括对系统表的访问权限。系统表为只读。

- VPC 策略示例：只读访问 – 此示例策略可以附加到 VPC 端点。（有关更多信息，请参阅控制对 Amazon VPC 资源的访问）。它限制通过所连接的 VPC 端点对 Amazon Keyspaces 资源进行只读访问的操作。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "cassandra:Select"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- VPC 策略示例：限制对特定 Amazon Keyspaces 表的访问 – 此示例策略可以附加到 VPC 端点。它限制通过所附加到的 VPC 端点访问特定表。

```
{
   "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**
>
> 要限制对特定表的访问，还必须包括对系统表的访问权限。系统表为只读。

可用性

Amazon Keyspaces 支持在所有提供服务 AWS 区域 的地方使用接口 VPC 终端节点。有关更多信息，请参阅 ???。

VPC 终端节点策略和 Amazon Keyspaces point-in-time 恢复 (PITR)

如果您使用带有条件键的 IAM 策略来限制传入流量，则表恢复操作可能会失败。例如，如果您使用 `aws:SourceVpce` 条件键将源流量限制到特定 VPC 端点，则表恢复操作将失败。要允许 Amazon Keyspaces 代表您的主体执行恢复操作，您必须在 IAM 策略中添加 `aws:ViaAWSService` 条件键。当任何 AWS 服务使用委托人的证书发出请求时，`aws:ViaAWSService`条件密钥允许访问。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件键。下面是一个示例策略。

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

常见错误和警告

如果您使用的是 Amazon Virtual Private Cloud 并连接到 Amazon Keyspaces，可能会看到以下警告。

```
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.
```

之所以出现此警告，是因为 `system.peers` 表包含 Amazon Keyspaces 有权查看的所有 Amazon VPC 端点的条目，包括您通过其连接的 Amazon VPC 端点。您可以放心地忽略这一警告。

有关其他错误，请参阅the section called “VPC 端点连接错误”。

Amazon Keyspaces 中的配置和漏洞分析

AWS 负责处理基本安全任务，如来宾操作系统 (OS) 和数据库补丁、防火墙配置和灾难恢复等。这些流程已通过相应第三方审核和认证。有关更多详细信息，请参阅以下资源：

- 责任共担模式
- Amazon Web Services：安全过程概述（白皮书）

Amazon Keyspaces 的安全最佳实践

Amazon Keyspaces（Apache Cassandra 兼容）提供了在您开发和实施自己的安全策略时需要考虑的大量安全功能。以下最佳实践是一般指导原则，并不代表完整安全解决方案。这些最佳实践可能不适合您的环境或不满足您的环境要求，请将其视为有用的考虑因素而不是惯例。

主题

Amazon Keyspaces 的预防性安全最佳实践

以下安全最佳实践属于预防性实践，因为它们可以帮助您预测和预防 Amazon Keyspaces 中的安全事件。

使用静态加密

Amazon Keyspaces 使用 AWS Key Management Service (AWS KMS) 中存储的加密密钥对存储在表中的所有用户数据进行静态加密。保护您的数据免受未经授权访问，为基础存储提供额外一层数据保护。

默认情况下，Amazon Keyspaces 使用 AWS 拥有的密钥来加密您的所有表。如果此密钥不存在，系统会为您创建一个。服务默认密钥无法禁用。

或者，您可以使用客户自主管理型密钥进行静态加密。有关更多信息，请参阅 Amazon Keyspaces Encryption at Rest。

使用 IAM 角色对 Amazon Keyspaces 的访问进行身份验证

对于访问 Amazon Keyspaces 的用户、应用程序和其他 AWS 服务，必须在 AWS API 请求中包含有效的 AWS 凭证。不应直接在应用程序或 EC2 实例中存储 AWS 凭证。这些长期凭证不会自动轮

换，如果外泄，可能造成重大业务影响。利用 IAM 角色，可以获得临时访问密钥，用于访问 AWS 服务和资源。

有关更多信息，请参阅 IAM 角色。

使用 IAM 策略进行 Amazon Keyspaces 基本授权

在授予权限时，您要决定谁获得权限，获得哪些 Amazon Keyspaces API 的权限，以及您允许对这些资源执行的具体操作。实施最低权限对于减小安全风险以及错误或恶意意图造成的影响至关重要。

将权限策略附加到 IAM 身份（即用户、组和角色），从而授予对 Amazon Keyspaces 资源执行操作的权限。

可以通过以下方法实现：

- AWS 托管式（预定义）策略
- 客户管理型策略

使用 IAM 策略条件进行精细访问控制

在 Amazon Keyspaces 中授予权限时，可以指定确定权限策略如何生效的条件。实施最低权限对于减小安全风险以及错误或恶意意图造成的影响至关重要。

使用 IAM 策略授予权限时，可以指定条件。例如，您可以执行以下操作：

- 授予权限，允许用户对特定键空间或表进行只读访问。
- 根据用户的身份授予权限，允许用户对某个表进行写入访问。

有关更多信息，请参阅 Identity-Based Policy Examples。

考虑客户端加密

如果您将敏感或机密数据存储在 Amazon Keyspaces 中，您可能希望将该数据加密到尽可能靠近其源的位置，以便在该数据的整个生命周期内对其进行保护。加密传输中和静态敏感数据有助于确保明文数据不会提供给任何第三方。

Amazon Keyspaces 的检测性安全最佳实践

以下安全最佳实践属于检测性实践，因为它们可以帮助您检测潜在的安全漏洞和事件。

使用 AWS CloudTrail 监控 AWS Key Management Service (AWS KMS) AWS KMS 密钥的使用情况

如果您使用客户自主管理型AWS KMS密钥进行静态加密，AWS CloudTrail 将记录该密钥的使用情况。CloudTrail 按照帐户上执行的记录操作，显示用户活动。CloudTrail 记录有关每个操作的重要信息，包括谁发出请求、所使用的服务、执行的操作、操作的参数以及 AWS 服务返回的响应元素。这些信息有助于跟踪 AWS 资源更改，解决运营问题。利用 CloudTrail，可以更轻松确保符合内部策略和法规标准。

可以使用 CloudTrail 审计密钥使用情况。CloudTrail 创建日志文件，其中包含账户的 AWS API 调用和相关事件历史记录。这些日志文件包含通过控制台、AWS SDK 和命令行工具，以及通过集成的 AWS 服务发出的所有 AWS KMS API 请求。可以使用这些日志文件获取 AWS KMS 密钥使用时间、请求的操作、请求者的身份、发出请求的 IP 地址等信息。有关更多信息，请参见用 AWS CloudTrail 记录 AWS Key Management Service API 调用和 AWS CloudTrail 用户指南。

使用 CloudTrail 监控 Amazon Keyspaces 数据定义语言 (DDL) 操作

CloudTrail 按照帐户上执行的记录操作，显示用户活动。CloudTrail 记录有关每个操作的重要信息，包括谁发出请求、所使用的服务、执行的操作、操作的参数以及 AWS 服务返回的响应元素。这些信息有助于跟踪 AWS 资源更改和排查运营问题。利用 CloudTrail，可以更轻松确保符合内部策略和法规标准。

所有 Amazon Keyspaces DDL 操作都会自动记录在 CloudTrail 中。DDL 操作让您可以创建和管理 Amazon Keyspaces 键空间和表。

当 Amazon Keyspaces 中发生活动时，该活动将与事件历史记录中的其他 AWS 服务事件一起记录在 CloudTrail 事件中。有关更多信息，请参阅 Logging Amazon Keyspaces operations by using AWS CloudTrail。您可以在 AWS 账户 中查看、搜索和下载最新事件。有关更多信息，请参阅 AWS CloudTrail 用户指南中的使用 CloudTrail 事件历史记录查看事件。

要持续记录 AWS 账户中的事件（包括 Amazon Keyspaces 的事件），请创建跟踪。跟踪记录让 CloudTrail 可以将日志文件发送至 Amazon Simple Storage Service (Amazon S3) 存储桶。在控制台创建跟踪记录时，该跟踪记录默认应用于所有 AWS 区域。跟踪记录 AWS 分区所有区域的事件，将日志文件传送至指定的 S3 存储桶。此外，可以配置其他 AWS 服务，进一步分析和应对 CloudTrail 日志中收集的事件数据。

标记 Amazon Keyspaces 资源以进行标识和自动化

可以将自己的元数据以标签形式分配给 AWS 资源。每个标签都是一个简单标注，包含一个客户定义的键和一个可选值，方便管理、搜索和筛选资源。

标签可实现分组控制。尽管没有固有类型的标签，但利用标签，可以根据用途、所有者、环境或其他条件分类资源。下面是一些示例：

- 访问：用于基于标签控制对 Amazon Keyspaces 资源的访问。有关更多信息，请参阅 the section called “基于 Amazon Keyspaces 标签的授权”。
- 安全性：用于确定数据保护设置等要求。
- 机密性：资源支持的特定数据机密等级的标识符。
- 环境 – 用于区分开发、测试和生产基础设施。

有关更多信息，请参阅 AWS tagging strategies 和 Adding tags and labels to resources。

Amazon Keyspaces（Apache Cassandra 兼容）的 CQL 语言参考

在连接到 Amazon Keyspaces（Apache Cassandra 兼容）端点后，您可以通过 Cassandra 查询语言 (CQL) 来使用数据库。CQL 在许多方面与结构化查询语言 (SQL) 相似。

主题

Amazon Keyspaces 中的 Cassandra 查询语言 (CQL) 元素

了解 Amazon Keyspaces 支持的 Cassandra 查询语言 (CQL) 元素，包括标识符、常量、术语和数据类型。

主题

标识符

标识符（或名称）用于标识表、列和其他对象。标识符可以带引号，也可以不带引号。以下情况将适用。

```
identifier          ::=  unquoted_identifier | quoted_identifier
unquoted_identifier ::=  re('[a-zA-Z][a-zA-Z0-9_]*')
quoted_identifier   ::=  '"' (any character where " can appear if doubled)+ '"'
```

常量

定义以下常量。

```
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+
```

术语

术语表示受支持的值的类型。术语由以下内容定义。

```
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
```

数据类型

Amazon Keyspaces 支持以下数据类型：

字符串类型

数据类型	描述
ascii	表示 ASCII 字符串。
text	表示 UTF-8 编码的字符串。
varchar	表示 UTF-8 编码的字符串（varchar 是 text 的别名）。

数字类型

数据类型	描述
bigint	表示 64 位有符号长整型。
counter	表示 64 位有符号整数计数器。有关更多信息，请参阅 the section called “计数器”。
decimal	表示可变精度小数。
double	表示 64 位 IEEE 754 浮点。
float	表示 32 位 IEEE 754 浮点。
int	表示 32 位有符号整数。
varint	表示任意精度的整数。

计数器

counter 列包含 64 位有符号整数。计数器值通过 the section called “UPDATE” 语句进行递增或递减，并且无法直接设置。这使得 counter 列对于跟踪计数很有用。例如，您可以使用计数器来跟踪日志文件中的条目数或某个社交网络上的文章被查看的次数。以下限制适用于 counter 列：

- 类型 counter 的列不能是表的 primary key 的一部分。
- 在包含一个或多个类型 counter 的列的表中，所有列的类型都必须为 counter。

如果计数器更新失败（例如，因超时或与 Amazon Keyspaces 的连接断开），客户端将不知道计数器值是否已更新。如果重试更新，则可能会再次应用对计数器值的更新。

Blob 类型

数据类型	描述
blob	表示任意字节。

布尔值类型

数据类型	描述
boolean	表示 true 或 false。

与时间相关的类型

数据类型	描述
timestamp	64 位有符号整数，表示自纪元（1970 年 1 月 1 日 00:00:00 GMT）以来的日期和时间，以毫秒为单位。
timeuuid	表示版本 1 UUID。

集合类型

数据类型	描述
list	表示文本元素的有序集合。
map	表示键/值对的无序集合。
set	表示一个或多个文本元素的无序集合。

您将通过在尖括号中使用集合类型后跟其他数据类型（如 TEXT 或 INT）来声明集合列。您可以创建包含 TEXT 的 SET 的列，也可以创建 TEXT 和 INT 键值对的 MAP，如以下示例所示。

```
SET <TEXT>
MAP <TEXT, INT>
```

非冻结集合可让您对每个单独的集合元素进行更新。系统将为各个元素存储客户端时间戳和存活时间 (TTL) 设置。

当您对集合类型使用 FROZEN 关键字时，集合的值将被序列化为单个不可变值，并且 Amazon Keyspaces 会将它们视为 BLOB。这是一个冻结集合。INSERT 或 UPDATE 语句将覆盖整个冻结集合。您无法对冻结集合中的单个元素进行更新。

客户端时间戳和存活时间 (TTL) 设置适用于整个冻结集合，而不是单个元素。Frozen 集合列可以是表的 PRIMARY KEY 的一部分。

您可以嵌套冻结集合。例如，如果 MAP 使用了 FROZEN 关键字，您可以在 SET 中定义 MAP，如以下示例所示。

```
SET <FROZEN> <MAP <TEXT, INT>>>
```

默认情况下，Amazon Keyspaces 支持嵌套最多五个级别的冻结集合。有关更多信息，请参阅 the section called “Amazon Keyspaces 服务限额”。有关 Apache Cassandra 的功能差异的更多信息，请参阅 the section called “FROZEN 集合”。有关 CQL 语法的更多信息，请参阅 the section called “CREATE TABLE” 和 the section called “ALTER TABLE”。

元组类型

tuple 数据类型表示一组有界的文本元素。您可以使用元组作为 user defined type 的替代项。您无需将 FROZEN 关键字用于元组。这是因为元组始终处于冻结状态，您无法单独更新元素。

其他类型

数据类型	描述
inet	一个表示 IPv4 或 IPv6 格式的 IP 地址的字符串。

静态

在包含聚类列的 Amazon Keyspaces 表中，您可以使用STATIC关键字创建任何类型的静态列。

下面是一个示例语句。

```
my_column INT STATIC
```

有关使用静态列的更多信息，请参阅the section called “静态列”。

Amazon Keyspaces 数据类型的 JSON 编码

Amazon Keyspaces 提供与 Apache Cassandra 相同的 JSON 数据类型映射。下表介绍了 Amazon Keyspaces 在 `INSERT JSON` 语句中接受的数据类型，以及 Amazon Keyspaces 在使用 `SELECT JSON` 语句返回数据时使用的数据类型。

对于单字段数据类型（如 `float`、`int`、`UUID` 和 `date`），您还可以将数据作为 `string` 插入。对于复合数据类型和集合（如 `tuple`、`map` 和 `list`），您还可以将数据作为 JSON 或编码 `JSON string` 插入。

JSON 数据类型	**INSERT JSON** 语句中接受的数据类型	**SELECT JSON** 语句中返回的数据类型	注意
`ascii`	`string`	`string`	使用 JSON 字符转义 `\u`。
`bigint`	`integer, string`	`integer`	字符串必须是有效的 64 位整数。
`blob`	`string`	`string`	字符串应以 `0x` 开头，后跟偶数个十六进制数字。
`boolean`	`boolean, string`	`boolean`	String 必须为 `true` 或 `false`。
`date`	`string`	`string`	格式为 `YYYY-MM-DD` 的日期，时区为 UTC。
`decimal`	`integer, float, string`	`float`	在客户端解码器中，浮点精度可以超过 32 位或 64 位 IEEE-754。
`double`	`integer, float, string`	`float`	字符串必须是有效的整数或浮点数。

JSON 数据类型	**INSERT JSON** 语句中接受的数据类型	**SELECT JSON** 语句中返回的数据类型	注意
float	integer, float, string	float	字符串必须是有效的整数或浮点数。
inet	string	string	IPv4 或 IPv6 地址。
int	integer, string	integer	字符串必须是有效的 32 位整数。
list	list, string	list	使用原生 JSON 列表表示形式。
map	map, string	map	使用原生 JSON 映射表示形式。
smallint	integer, string	integer	字符串必须是有效的 16 位整数。
set	list, string	list	使用原生 JSON 列表表示形式。
text	string	string	使用 JSON 字符转义 \u。
time	string	string	一天中的时间，格式为 HH-MM-SS[.fffffffff] 。
timestamp	integer, string	string	时间戳。字符串常量可让您将时间戳存储为日期。将返回格式为 YYYY-MM-DD HH:MM:SS.SSS 的日期戳。

JSON 数据类型	**INSERT JSON** 语句中接受的数据类型	**SELECT JSON** 语句中返回的数据类型	注意
timeuuid	string	string	类型 1 UUID。有关 UUID 格式的信息，请参阅constants。
tinyint	integer, string	integer	字符串必须是有效的 8 位整数。
tuple	list, string	list	使用原生 JSON 列表表示形式。
uuid	string	string	有关 UUID 格式的信息，请参阅constants。
varchar	string	string	使用 JSON 字符转义 \u。
varint	integer, string	integer	可变长度；在客户端解码器中，可能会溢出 32 位或 64 位整数。

Amazon Keyspaces 中的 DDL 语句（数据定义语言）

数据定义语言 (DDL) 是一组用于管理 Amazon Keyspaces（Apache Cassandra 兼容）中的数据结构（如键空间和表）的 Cassandra 查询语言 (CQL) 语句。可以使用 DDL 创建这些数据结构，在创建数据结构后对其进行修改，以及在不再使用数据结构时将其删除。Amazon Keyspaces 以异步方式执行 DDL 操作。有关如何确认异步操作已完成的更多信息，请参阅the section called “异步创建与删除键空间和表”。

支持以下 DDL 语句：

- CREATE KEYSPACE
- ALTER KEYSPACE
- DROP KEYSPACE

- CREATE TABLE
- ALTER TABLE
- RESTORE TABLE
- DROP TABLE

主题

- Keyspaces
- 表

Keyspaces

键空间 对与一个或多个应用程序相关的表进行分组。就关系数据库管理系统 (RDBMS) 而言，键空间与数据库、表空间或类似的结构大致相似。

> **Note**
>
> 在 Apache Cassandra 中，键空间将决定如何在多个存储节点之间复制数据。但是，Amazon Keyspaces 是一项完全托管式服务：系统将为您管理其存储层的详细信息。因此，Amazon Keyspaces 中的键空间只是逻辑结构，并且与底层物理存储无关。

有关 Amazon Keyspaces 键空间的配额限制和约束的信息，请参阅 限额。

密钥空间的语句

- CREATE KEYSPACE
- ALTER KEYSPACE
- DROP KEYSPACE

CREATE KEYSPACE

使用 `CREATE KEYSPACE` 语句可创建新的键空间。

语法

```
create_keyspace_statement ::=
    CREATE KEYSPACE [ IF NOT EXISTS ] keyspace_name
```

```
    WITH options
```

其中：

- *keyspace_name* 是要创建的键空间的名称。
- 选项 为下列一个或多个项：
  - REPLICATION：表示键空间复制策略的映射：
    - SingleRegionStrategy：用于单区域键空间。（必需）
    - NetworkTopologyStrategy— 指定至少两个，最多六个 AWS 区域。每个区域的复制因子为 3。（可选）
  - DURABLE_WRITES：由于对 Amazon Keyspaces 的写入操作始终是持久的，因此此选项不是必需的。但是，如果指定此选项，则值必须为 true。
  - TAGS：创建资源时要附加到资源的键值对标签的列表。（可选）

示例

创建键空间，如下所示。

```
CREATE KEYSPACE my_keyspace
    WITH REPLICATION = {'class': 'SingleRegionStrategy'} and TAGS ={'key1':'val1',
 'key2':'val2'} ;
```

要创建多区域键空间，请指定 NetworkTopologyStrategy 并包含最少两个、最多六个 AWS 区域。每个区域的复制因子为 3。

```
CREATE KEYSPACE my_keyspace
    WITH REPLICATION = {'class':'NetworkTopologyStrategy', 'us-east-1':'3', 'ap-
southeast-1':'3','eu-west-1':'3'};
```

ALTER KEYSPACE

使用 ALTER KEYSPACE 可在键空间中添加或删除标签。

语法

```
alter_keyspace_statement ::=
    ALTER KEYSPACE keyspace_name
    [[ADD | DROP] TAGS
```

其中：

- *keyspace_name* 是要更改的键空间的名称。
- TAGS：要在键空间中添加或删除的键值对标签的列表。

示例

更改键空间，如下所示。

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

DROP KEYSPACE

使用 DROP KEYSPACE 语句可删除键空间，包括其所有内容（如表）。

语法

```
drop_keyspace_statement ::=
    DROP KEYSPACE [ IF EXISTS ] keyspace_name
```

其中：

- keyspace_name 是要删除的键空间的名称。

示例

```
DROP KEYSPACE "myGSGKeyspace";
```

表

表是 Amazon Keyspaces 中的主要数据结构。表中的数据按行和列进行组织。这些列的子集用于通过指定分区键来确定分区（以及最终的数据放置）。

可以将另一组列定义为聚类列，这意味着它们可以作为谓词参与查询执行。

默认情况下，将创建具有按需 吞吐容量的新表。您可以更改新表和现有表的容量模式。有关读/写容量吞吐量模式的更多信息，请参阅 the section called “读/写容量模式”。

对于处于预配模式的表，您可以配置可选AUTOSCALING_SETTINGS。有关 Amazon Keyspaces 自动扩展和可用选项的更多信息，请参阅。the section called “使用 CQL”

有关 Amazon Keyspaces 表的配额限制和约束的信息，请参阅 限额。

表格的语句

- CREATE TABLE
- ALTER TABLE
- RESTORE TABLE
- DROP TABLE

CREATE TABLE

使用 CREATE TABLE 语句可创建新表。

语法

```
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) )*
```

其中：

- *table_name* 是要创建的表的名称。
- *column_definition* 包含以下各项：
  - *column_name*：列的名称。
  - *cql_type*：一种 Amazon Keyspaces 数据类型（参见数据类型）。
  - *FROZEN*：将 collection 类型（例如 LIST、SET 或 MAP）的此列指定为已冻结。冻结集合将被序列化为单个不可变值，并被视为 BLOB。有关更多信息，请参阅 the section called “集合类型”。
  - *STATIC*：将此列指定为静态。静态列存储由同一分区中的所有行共享的值。
  - *PRIMARY KEY*：将此列指定为表的主键。
- *primary_key* 包含以下各项：
  - partition_key
  - clustering_columns
- *partition_key*:
  - 分区键可以是单个列，也可以是由两列或多个列组成的复合值。主键的分区键部分是必需的，它决定了 Amazon Keyspaces 存储数据的方式。
- *clustering_columns*:
  - 主键的可选聚类列部分决定如何在每个分区中聚类和排序数据。
- *table_options* 包含以下各项：
  - *CLUSTERING ORDER BY*：表上的默认 CLUSTERING ORDER 包含 ASC（升序）排序方向的聚类键。指定它可覆盖默认排序行为。
  - *CUSTOM_PROPERTIES*：特定于 Amazon Keyspaces 的设置的映射。
    - capacity_mode：指定表的读/写吞吐容量模式。选项为 throughput_mode:PAY_PER_REQUEST 和 throughput_mode:PROVISIONED。预置容量模式要求将 read_capacity_units 和 write_capacity_units 作为输入。默认值为 throughput_mode:PAY_PER_REQUEST。
    - client_side_timestamps：指定为表启用还是禁用客户端时间戳。选项为 {'status': 'enabled'} 和 {'status': 'disabled'}。如果未指定，则默认为 status:disabled。为表启用客户端时间戳后便无法禁用该设置。
    - encryption_specification：为静态加密指定加密选项。如果未指定，则默认为 encryption_type:AWS_OWNED_KMS_KEY。加密选项客户托管密钥要求输入 Amazon 资源名称 (ARN) 格式的 AWS KMS 密钥作为输入:: kms_key_identifier:ARN。kms_key_identifier:ARN

- `point_in_time_recovery`：指定是启用还是禁用表的 point-in-time 还原。选项为 `status:enabled` 和 `status:disabled`。如果未指定，则默认为 `status:disabled`。
- `replica_updates`：指定多区域表中特定于的 AWS 区域设置。对于多区域表，您可以根据不同的方式配置表的读取容量。 AWS 区域您可以通过配置以下参数来做到这一点。有关更多信息以及示例，请参阅 the section called “创建具有预置容量模式和 auto Scaling (CQL) 的多区域表”。
  - `region`— 具有以下设置 AWS 区域 的表副本的：
    - `read_capacity_units`
- TTL：为表启用存活时间自定义设置。要启用，请使用 `status:enabled`。默认值为 `status:disabled`。为表启用 TTL 后便无法禁用该设置。

- *`AUTOSCALING_SETTINGS`*包括置备模式下的表的以下可选设置。有关更多信息以及示例，请参阅 the section called “使用 CQL 创建具有自动缩放功能的新表”。
  - `provisioned_write_capacity_autoscaling_update`:
    - `autoscaling_disabled`— 要为写入容量启用 auto 缩放，请将该值设置为`false`。默认值为 `true`。（可选）
    - `minimum_units`— 表应随时准备支持的最低写入吞吐量级别。该值必须介于 1 和您账户的最大每秒吞吐量配额（默认为 40,000）之间。
    - `maximum_units`— 表应随时准备支持的最大写入吞吐量级别。该值必须介于 1 和您账户的最大每秒吞吐量配额（默认为 40,000）之间。
    - `scaling_policy`— Amazon Keyspaces 支持目标跟踪政策。auto Scaling 目标是表的预配置写入容量。
      - `target_tracking_scaling_policy_configuration`— 要定义目标跟踪策略，必须定义目标值。有关目标跟踪和冷却时间的更多信息，请参阅 App lication Auto Scaling 用户指南中的目标跟踪扩展策略。
        - `target_value`— 表格的目标利用率。Amazon Keyspaces 自动扩展可确保已用容量与预配置容量的比率保持在该值或接近该值。您将 `target_value` 定义为百分比。20 到 90 之间的双打。（必需）
        - `scale_in_cooldown`— 两次缩放活动之间的冷却时间（以秒为单位），可让表格在另一个缩放活动开始之前稳定下来。如果未提供任何值，则默认值为 0。（可选）
        - `scale_out_cooldown`— 两次扩展活动之间的冷却时间（以秒为单位），可让表格在另一个扩展活动开始之前稳定下来。如果未提供任何值，则默认值为 0。（可选）

- disable_scale_in: Aboolean，它指定该表scale-in是禁用还是启用。默认情况下，此参数处于禁用状态。要开启scale-in，请将该boolean值设置为FALSE。这意味着代表您自动缩小餐桌的容量。（可选）

- provisioned_read_capacity_autoscaling_update:
  - autoscaling_disabled— 要为读取容量启用 auto 缩放，请将该值设置为false。默认值为 true。（可选）
  - minimum_units— 表应随时准备支持的最低吞吐量级别。该值必须介于 1 和您账户的最大每秒吞吐量配额（默认为 40,000）之间。
  - maximum_units— 表应随时准备支持的最大吞吐量级别。该值必须介于 1 和您账户的最大每秒吞吐量配额（默认为 40,000）之间。
  - scaling_policy— Amazon Keyspaces 支持目标跟踪政策。auto Scaling 目标是表的预配置读取容量。
    - target_tracking_scaling_policy_configuration— 要定义目标跟踪策略，必须定义目标值。有关目标跟踪和冷却时间的更多信息，请参阅 App lication Auto Scaling 用户指南中的目标跟踪扩展策略。
      - target_value— 表格的目标利用率。Amazon Keyspaces 自动扩展可确保已用容量与预配置容量的比率保持在该值或接近该值。您将 target_value 定义为百分比。20 到 90 之间的双打。（必需）
      - scale_in_cooldown— 两次缩放活动之间的冷却时间（以秒为单位），可让表格在另一个缩放活动开始之前稳定下来。如果未提供任何值，则默认值为 0。（可选）
      - scale_out_cooldown— 两次扩展活动之间的冷却时间（以秒为单位），可让表格在另一个扩展活动开始之前稳定下来。如果未提供任何值，则默认值为 0。（可选）
      - disable_scale_in: Aboolean，它指定该表scale-in是禁用还是启用。默认情况下，此参数处于禁用状态。要开启scale-in，请将该boolean值设置为FALSE。这意味着代表您自动缩小餐桌的容量。（可选）

- replica_updates：指定多区域表的 AWS 区域 特定自动缩放设置。对于多区域表，您可以根据不同的方式配置表的读取容量。 AWS 区域您可以通过配置以下参数来做到这一点。有关更多信息以及示例，请参阅 the section called “创建具有预置容量模式和 auto Scaling (CQL) 的多区域表”。
  - region— 具有以下设置 AWS 区域 的表副本的：
    - provisioned_read_capacity_autoscaling_update
      - autoscaling_disabled— 要为表的读取容量启用 auto 缩放，请将该值设置为false。默认值为 true。（可选）

> **Note**
>
> 必须为该表的所有副本启用或禁用多区域表的自动缩放。

- `minimum_units`— 表应随时准备支持的最低读取吞吐量级别。该值必须介于 1 和您账户的最大每秒吞吐量配额（默认为 40,000）之间。
- `maximum_units`— 表应随时准备支持的最大读取吞吐量级别。该值必须介于 1 和您账户的最大每秒吞吐量配额（默认为 40,000）之间。
- `scaling_policy`— Amazon Keyspaces 支持目标跟踪政策。auto Scaling 目标是表的预配置读取容量。
  - `target_tracking_scaling_policy_configuration`— 要定义目标跟踪策略，必须定义目标值。有关目标跟踪和冷却时间的更多信息，请参阅 App lication Auto Scaling 用户指南中的目标跟踪扩展策略。
    - `target_value`— 表格的目标利用率。Amazon Keyspaces 自动扩展可确保已消耗的读取容量与预配置读取容量的比率保持在该值或接近该值。您将 `target_value` 定义为百分比。20 到 90 之间的双打。（必需）
    - `scale_in_cooldown`— 两次缩放活动之间的冷却时间（以秒为单位），可让表格在另一个缩放活动开始之前稳定下来。如果未提供任何值，则默认值为 0。（可选）
    - `scale_out_cooldown`— 两次扩展活动之间的冷却时间（以秒为单位），可让表格在另一个扩展活动开始之前稳定下来。如果未提供任何值，则默认值为 0。（可选）
    - `disable_scale_in`: Aboolean，它指定该表`scale-in`是禁用还是启用。默认情况下，此参数处于禁用状态。要开启`scale-in`，请将该`boolean`值设置为`FALSE`。这意味着代表您自动缩小表格的读取容量。（可选）
- `default_time_to_live`：表的默认存活时间设置（以秒为单位）。
- `TAGS`：创建资源时要附加到资源的键值对标签的列表。

- *`clustering_order`* 包含以下各项：
  - *`column_name`*：列的名称。
  - *`ASC | DESC`*：设置升序 (ASC) 或降序 (DESC) 顺序修饰符。如果未指定，则默认顺序为 ASC。

示例

```
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;
```

在使用聚类列的表中，可以在表定义中将非聚类列声明为静态列。有关静态列的更多信息，请参阅 the section called “静态列”。

示例

```
CREATE TABLE "my_keyspace".my_table (
                                            id int,
                                            name text,
                                            region text,
                                            division text,
                                            project text STATIC,
                                            PRIMARY KEY (id,division));
```

ALTER TABLE

使用 ALTER TABLE 语句可添加新列、添加标签或更改表的自定义属性。

语法

```
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
```

其中：

- *table_name* 是要更改的表的名称。
- *column_definition* 是要添加的列和数据类型的名称。
- *column_definition_list* 是放在括号内的列的逗号分隔列表。
- *table_options* 包含以下各项：
  - *CUSTOM_PROPERTIES*：特定于 Amazon Keyspaces 的设置的映射。
    - capacity_mode：指定表的读/写吞吐容量模式。选项为 throughput_mode:PAY_PER_REQUEST 和 throughput_mode:PROVISIONED。预置容量模式要求将 read_capacity_units 和 write_capacity_units 作为输入。默认值为 throughput_mode:PAY_PER_REQUEST。
    - client_side_timestamps：指定为表启用还是禁用客户端时间戳。选项为 {'status': 'enabled'} 和 {'status': 'disabled'}。如果未指定，则默认为 status:disabled。为表启用客户端时间戳后便无法禁用该设置。
    - encryption_specification：为静态加密指定加密选项。选项为 encryption_type:AWS_OWNED_KMS_KEY 和 encryption_type:CUSTOMER_MANAGED_KMS_KEY。加密选项客户自主管理型密钥要求输入 Amazon 资源名称 (ARN) 格式的 AWS KMS 密钥作为输入：kms_key_identifier:ARN。
    - point_in_time_recovery：指定是启用还是禁用表的 point-in-time 还原。选项为 status:enabled 和 status:disabled。默认值为 status:disabled。

    - `replica_updates`：指定多区域表的 AWS 区域 特定设置。对于多区域表，您可以根据不同的方式配置表的读取容量。 AWS 区域您可以通过配置以下参数来做到这一点。有关更多信息以及示例，请参阅 the section called “更新多区域表 (CQL) 的预配置容量和 auto scaling 设置”。
      - `region`— 具有以下设置 AWS 区域 的表副本的：
        - `read_capacity_units`
    - `ttl`：为表启用存活时间自定义设置。要启用，请使用 `status:enabled`。默认值为 `status:disabled`。为表启用 `ttl` 后便无法禁用该设置。
  - *`AUTOSCALING_SETTINGS`*包括已配置表的可选 auto Scaling 设置。有关语法和详细说明，请参见the section called “CREATE TABLE”。有关示例，请参阅the section called “使用 CQL 在现有表上启用自动缩放”。
- `default_time_to_live`：表的默认存活时间设置（以秒为单位）。
- *`TAGS`* 是要附加到资源的键/值对标签的列表。

> **Note**
>
> 使用 ALTER TABLE 时，您只能更改单个自定义属性。您无法在同一语句中组合多个 ALTER TABLE 命令。

示例

以下语句展示如何将列添加到现有表。

```
ALTER TABLE mykeyspace.mytable ADD (ID int);
```

此语句展示如何将两个集合列添加到现有表：

- 包含嵌套冻结集合的冻结集合列 `col_frozen_list`
- 包含嵌套冻结集合的非冻结集合列 `col_map`

```
ALTER TABLE my_Table ADD(col_frozen_list FROZEN<LIST<FROZEN<SET<TEXT>>>>, col_map
 MAP<INT, FROZEN<SET<INT>>>);
```

要更改表的容量模式并指定读取和写入容量单位，可以使用以下语句。

```
ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 10, 'write_capacity_units':
 20}};
```

以下语句为表指定客户自主管理型 KMS 密钥。

```
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'
                  }
         };
```

要为表启用 point-in-time 还原，可以使用以下语句。

```
ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={'point_in_time_recovery':
 {'status': 'enabled'}};
```

要为表设置默认存活时间值（以秒为单位），可以使用以下语句。

```
ALTER TABLE my_table WITH default_time_to_live = 2592000;
```

此语句为表启用自定义存活时间设置。

```
ALTER TABLE mytable WITH CUSTOM_PROPERTIES={'ttl':{'status': 'enabled'}};
```

RESTORE TABLE

使用 RESTORE TABLE 语句可将表恢复到某个时间点。此语句要求在表上启用 point-in-time 恢复。有关更多信息，请参阅 时间点恢复。

语法

```
restore_table_statement ::=
    RESTORE TABLE restored_table_name FROM TABLE source_table_name
                  [ WITH table_options [ , ... ] ];
```

其中：

- *restored_table_name* 是已恢复的表的名称。
- *source_table_name* 是源表的名称。
- *table_options* 包含以下各项：
  - *restore_timestamp* 是 ISO 8601 格式的恢复点时间。如果未指定，则使用当前时间戳。
  - *CUSTOM_PROPERTIES*：特定于 Amazon Keyspaces 的设置的映射。
    - capacity_mode：指定表的读/写吞吐容量模式。选项为 throughput_mode:PAY_PER_REQUEST 和 throughput_mode:PROVISIONED。预置容量模式要求将 read_capacity_units 和 write_capacity_units 作为输入。默认值为源表中的当前设置。
    - encryption_specification：为静态加密指定加密选项。选项为 encryption_type:AWS_OWNED_KMS_KEY 和 encryption_type:CUSTOMER_MANAGED_KMS_KEY。加密选项客户托管密钥要求输入亚马逊资源名称 (ARN) 格式的 AWS KMS 密钥作为输入：。kms_key_identifier:ARN要将使用客户托管密钥加密的表恢复到使用加密的表 AWS 拥有的密钥，Amazon Keyspaces 需要访问源表的 AWS KMS 密钥。
    - point_in_time_recovery：指定是启用还是禁用表的 point-in-time 还原。选项为 status:enabled 和 status:disabled。与创建新表时不同，已恢复的表的默认状态为 status:enabled，因为该设置继承自源表。要对已恢复的表禁用 PITR，必须显式设置 status:disabled。
    - replica_updates：指定多区域表的 AWS 区域 特定设置。对于多区域表，您可以根据不同的方式配置表的读取容量。 AWS 区域您可以通过配置以下参数来做到这一点。
      - region— 具有以下设置 AWS 区域 的表副本的：
        - read_capacity_units
  - *AUTOSCALING_SETTINGS*包括已配置表的可选 auto Scaling 设置。有关详细语法和说明，请参阅the section called “CREATE TABLE”。
  - *TAGS* 是要附加到资源的键/值对标签的列表。

> **Note**
>
> 已删除的表只能恢复到删除时的状态。

示例

```
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

使用 DROP TABLE 语句可从键空间中删除表。

语法

```
drop_table_statement ::=
    DROP TABLE [ IF EXISTS ] table_name
```

其中：

- IF EXISTS 防止 DROP TABLE 在表不存在的情况下失败。（可选）
- *table_name* 是要删除的表的名称。

示例

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

Amazon Keyspaces 中的 DML（数据操作语言）语句

数据操作语言 (DML) 是一组 Cassandra 查询语言 (CQL) 语句，用于管理 Amazon Keyspaces（Apache Cassandra 兼容）表中的数据。可以使用 DML 语句在表中添加、修改或删除数据。

还可以使用 DML 语句查询表中的数据。（请注意，CQL 不支持联接或子查询。）

主题

- SELECT
- INSERT
- UPDATE

- 删除

SELECT

使用 SELECT 语句可查询数据。

语法

```
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 ] )*
```

示例

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

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

有关将 JSON 编码的数据类型映射到 Amazon Keyspaces 数据类型的表，请参阅 the section called “Amazon Keyspaces 数据类型的 JSON 编码”。

使用 **IN** 关键字

IN 关键字指定一个或多个值的相等性。它可以应用于分区键和聚类列。结果按照键在 SELECT 语句中的显示顺序返回。

示例

```
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;
```

有关 IN 关键字以及 Amazon Keyspaces 如何处理该语句的更多信息，请参阅the section called “IN SELECT 语句”。

对结果排序

ORDER BY 子句指定返回结果的排序顺序。它采用列名称列表以及每列的排序顺序作为参数。您只能在排序子句中指定聚类列。不允许指定非聚类列。排序顺序选项是 ASC（用于升序排序顺序）和 DESC（用于降序排序顺序）。如果排序顺序被忽略，则使用聚类列的默认排序方式。有关可能的排序顺序，请参阅the section called “对结果排序”。

示例

```
SELECT name, id, division, manager_id FROM "myGSGKeyspace".employees_tbl WHERE id =
 '012-34-5678' ORDER BY division;
```

将 ORDER BY 与 IN 关键字一起使用时，结果在一个页面内排序。不支持在禁用分页的情况下进行完全重新排序。

TOKEN

您可以将 TOKEN 函数应用于 SELECT 和 WHERE 子句中的 PARTITION KEY 列。使用 TOKEN 函数时，Amazon Keyspaces 会根据 PARTITION_KEY 的映射令牌值（而不是 PARTITION KEY 的值）返回行。

IN 关键字不支持 TOKEN 关系。

示例

```
SELECT TOKEN(id) from my_table;

SELECT TOKEN(id) from my_table WHERE TOKEN(id) > 100 and TOKEN(id) < 10000;
```

TTL 函数

可以将 TTL 函数与 SELECT 语句一起使用，检索为列存储的到期时间（以秒为单位）。如果未设置 TTL 值，该函数将返回 null。

示例

```
SELECT TTL(my_column) from my_table;
```

TTL 函数不能用于多单元格列，例如集合。

WRITETIME 函数

仅当表使用了客户端时间戳时，您才能将 WRITETIME 函数与 SELECT 语句一起使用来检索存储为列值的元数据的时间戳。有关更多信息，请参阅 客户端时间戳。

```
SELECT WRITETIME(my_column) from my_table;
```

WRITETIME 函数不能用于多单元格列，例如集合。

> **Note**
>
> 为了与既定的 Cassandra 驱动程序行为兼容，当您通过 Cassandra 驱动程序和开发人员工具使用 Cassandra 查询语言 (CQL) API 调用对系统表执行操作时，系统不会强制实施基于标签的授权策略。有关更多信息，请参阅 the section called “ 基于标签的 Amazon Keyspaces 资源访问”。

INSERT

使用 INSERT 语句可向表添加行。

语法

```
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 )* ')'
```

示例

```
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') ;
```

更新参数

INSERT 支持以下值作为 update_parameter：

- TTL：以秒为单位的时间值。最大可配置值为 630720000 秒，相当于 20 年。
- TIMESTAMP：一个 bigint 值，表示自标准基准时间（称为epoch：1970 年 1 月 1 日 00:00:00 GMT）以来的微秒数。Amazon Keyspaces 中的时间戳必须介于过去 2 天和未来 5 分钟之间。

示例

```
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 支持

有关将 JSON 编码的数据类型映射到 Amazon Keyspaces 数据类型的表，请参阅 the section called “Amazon Keyspaces 数据类型的 JSON 编码”。

可以使用 JSON 关键字将 JSON 编码的映射作为单行插入。对于表中存在但在 JSON 插入语句中省略的列，请使用 DEFAULT UNSET 保留现有值。使用 DEFAULT NULL 可将 NULL 值写入省略的列的每一行，并覆盖现有值（收取标准写入费用）。DEFAULT NULL 是默认选项。

示例

```
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"}';
```

如果 JSON 数据包含重复键，Amazon Keyspaces 会存储键的最后一个值（类似于 Apache Cassandra）。在以下示例（重复键为 id）中，使用了值 234-56-7890。

示例

```
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

使用 UPDATE 语句可修改表中的行。

语法

```
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
```

示例

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

要递增 counter，请使用以下语法。有关更多信息，请参阅 the section called “计数器”。

```
UPDATE ActiveUsers SET counter = counter + 1  WHERE user = A70FE1C0-5408-4AE3-
BE34-8733E5K09F14 AND action = 'click';
```

更新参数

UPDATE 支持以下值作为 update_parameter：

- TTL：以秒为单位的时间值。最大可配置值为 630720000 秒，相当于 20 年。
- TIMESTAMP：一个 bigint 值，表示自标准基准时间（称为epoch：1970 年 1 月 1 日 00:00:00 GMT）以来的微秒数。Amazon Keyspaces 中的时间戳必须介于过去 2 天和未来 5 分钟之间。

示例

```
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';
```

删除

使用 DELETE 语句可从表中删除行。

语法

```
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
```

其中：

- *table_name* 是包含要删除的行的表。

示例

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

DELETE 支持以下值作为 update_parameter：

- TIMESTAMP：一个 bigint 值，表示自标准基准时间（称为epoch：1970 年 1 月 1 日 00:00:00 GMT）以来的微秒数。

Amazon Keyspaces 中的内置函数

Amazon Keyspaces（Apache Cassandra 兼容）支持可在 Cassandra 查询语言 (CQL) 语句中使用的各种内置函数。

主题

标量函数

标量函数对单个值执行计算，并将结果作为单个值返回。Amazon Keyspaces 支持以下标量函数。

函数	描述
blobAsType	返回指定数据类型的值。
cast	将一种原生数据类型转换为另一种原生数据类型。
currentDate	将当前日期/时间作为日期返回。
currentTime	将当前日期/时间作为时间返回。
currentTimestamp	将当前日期/时间作为时间戳返回。
currentTimeUUID	返回 timeuuid 形式的当前日期/时间。
fromJson	将 JSON 字符串转换为所选列的数据类型。

函数	描述
maxTimeuuid	返回时间戳或日期字符串可能的最大 timeuuid。
minTimeuuid	返回时间戳或日期字符串可能的最小 timeuuid。
now	返回新的唯一 timeuuid。支持 INSERT、UPDATE 和 DELETE 语句，并作为 SELECT 语句中 WHERE 子句的一部分。
toDate	将 timeuuid 或时间戳转换为日期类型。
toJson	以 JSON 格式返回所选列的列值。
token	返回分区键的哈希值。
toTimestamp	将 timeuuid 或日期转换为时间戳。
TTL	返回列的到期时间（以秒为单位）。
typeAsBlob	将指定数据类型转换为 blob。
toUnixTimestamp	将 timeuuid 或时间戳转换为 bigInt。
uuid	返回随机版本 4 UUID。支持 INSERT、UPDATE 和 DELETE 语句，并作为 SELECT 语句中 WHERE 子句的一部分。
writetime	返回指定列的值的时间戳。
dateOf	（已弃用）提取 timeuuid 的时间戳，并将该值作为日期返回。
unixTimestampOf	（已弃用）提取 timeuuid 的时间戳，并将值作为原始 64 位整数时间戳返回。

Amazon Keyspaces（Apache Cassandra 兼容）限额

本节介绍了 Amazon Keyspaces（Apache Cassandra 兼容）的当前限额和默认值。

主题

Amazon Keyspaces 服务限额

下表包含 Amazon Keyspaces（Apache Cassandra 兼容）的限额和默认值。服务限额控制台中提供了有关可以调整哪些限额的信息，您也可以在其中申请提高限额。有关配额的更多信息，请联系 AWS Support。

限额	描述	Amazon Keyspaces 默认值
每人的最大密钥空间 AWS 区域	每个区域此订阅者的最大键空间数。您可以在 Serv ice Q uotas 控制台中调整此默认值。	256
每张桌子的最大数量 AWS 区域	每个区域此订阅者在所有键空间中的最大表数。您可以在 Serv ice Q uotas 控制台中调整此默认值。	256
最大表架构大小	表架构的最大大小	350 KB
最大并发 DDL 操作数	每个区域允许此订阅用户执行的并发 DDL 操作的最大数量。	50
每个连接的最大查询数	每秒单个客户端 TCP 连接可处理的最大 CQL 查询数。	3000

限额	描述	Amazon Keyspaces 默认值
最大行大小	行的最大大小，不包括静态列数据。有关更多信息，请参阅 the section called “计算行大小”。	1 MB
INSERT 和 UPDATE 语句中的最大列数	CQL INSERT 或 UPDATE 语句中允许的最大列数。禁用生存时间 (TTL) 后，INSERT 或 UPDATE 语句最多支持 225 个常规列。如果启用 TTL，一次操作最多可以修改 166 个常规列。	225/166
每个逻辑分区的最大静态数据量	逻辑分区中静态数据的最大聚合大小。有关更多信息，请参阅 the section called “计算每个逻辑分区的静态列大小”。	1 MB
每个 IN SELECT 语句的最大子查询数	可在 SELECT 语句中为 IN 关键字使用的子查询的最大数量。您可以在 Serv ice Q uotas 控制台中调整此默认值。	100
每个嵌套冻结集合的最大数量 AWS 区域	对具有集合数据类型的列使用 FROZEN 关键字时支持的最大嵌套集合数量。有关冻结集合的更多信息，请参阅 the section called “集合类型”。要提高嵌套级别，请联系 AWS Support。	5

限额	描述	Amazon Keyspaces 默认值
每秒最大读取吞吐量	可分配给每个区域的表的每秒最大读取吞吐量 [读取请求单位 (RRU) 或读取容量单位 (RCU)]。您可以在 Serv ice Q uotas 控制台中调整此默认值。	40000
每秒最大写入吞吐量	可分配给每个区域的表的每秒最大写入吞吐量 [写入请求单位 (WRU) 或写入容量单位 (WCU)]。您可以在 Serv ice Q uotas 控制台中调整此默认值。	40000
账户级读取吞吐量（预置）	每个区域为该账户分配的最大聚合读取容量单位 (RCU) 数。这仅适用于处于预置读/写容量模式的表。您可以在 Serv ice Q uotas 控制台中调整此默认值。	80,000
账户级写入吞吐量（预置）	每个区域为该账户分配的最大聚合写入容量单位 (WCU) 数。这仅适用于处于预置读/写容量模式的表。您可以在 Serv ice Q uotas 控制台中调整此默认值。	80,000

限额	描述	Amazon Keyspaces 默认值
每个账户每个区域的最大可扩展目标数量	每个区域的账户可扩展目标的最大数量。如果为读取容量启用了自动缩放，则 Amazon Keyspaces 表算作一个可扩展目标；如果为写入容量启用了自动缩放，则视为另一个可扩展目标。您可以在 Application Auto Scaling 的服务配额控制台中通过选择 Amazon Keyspaces 的可扩展目标来调整此默认值。	1500
最大分区键大小	复合分区键的最大大小。为元数据分区键中包含的每一列的原始大小添加最多 3 字节的附加存储空间。	2048 字节
最大聚类键大小	所有聚类列的最大组合大小。为元数据每一聚类列的原始大小添加最多 4 字节的附加存储空间。	850 字节
使用 P Recovery (PITR) oint-in-time 恢复的最大并发表数	每个订阅用户使用 PITR 还原并发表的最大数量为 4。您可以在 Serv ice Q uotas 控制台中调整此默认值。	4
使用 point-in-time 恢复功能恢复的最大数据量 (PITR)	可以在 24 小时内使用 PITR 恢复的最大数据大小。您可以在 Serv ice Q uotas 控制台中调整此默认值。	5 TB

增加或减少吞吐量（对于预调配表）

增加预调配吞吐量

您可以使用控制台ReadCapacityUnits或WriteCapacityUnitsALTER TABLE语句增加或根据需要增加频率。新设置在 ALTER TABLE 操作完成后才会生效。

添加预配置容量时，不能超过每个账户的配额。而且，您可以根据需要增加表的预配置容量。有关每个账户的配额的更多信息，请参阅上一部分 the section called “Amazon Keyspaces 服务限额”。

减少预调配吞吐量

对于 ALTER TABLE 语句中的每个表，您可以减少 ReadCapacityUnits 和/或 WriteCapacityUnits 的值。新设置在 ALTER TABLE 操作完成后才会生效。

每天的任何时间最多可执行 4 次减小操作。天依据通用协调时间 (UTC) 定义。此外，如果过去 1 小时内未执行减小操作，则可以执行额外的减小操作。这实际上将每日的减小操作的最大次数设置为 27 次（在前 1 个小时内为 4 次减小操作，对于一天内的每个后续 1 小时时段，为 1 次减小操作）。

Amazon Keyspaces 静态加密

从表格创建之日起，您最多可以在 24 小时窗口内按表更改 AWS 自有 AWS KMS 密 AWS KMS 钥和客户管理密钥之间的加密选项四次。此外，如果过去 6 小时内未执行任何更改，则可以执行额外的更改。这实际上将每日的更改操作的最大次数设置为 8 次（在前 6 个小时内为 4 次更改操作，对于一天内的每个后续 6 小时时段，为 1 次更改操作）。

即使之前的配额已用完，您也可以根据需要更改加密选项以使用自有 AWS KMS 密钥。 AWS

相关配额如下，但您也可以请求提高配额。要申请增加服务配额，请参阅AWS Support。

Amazon Keyspaces（Apache Cassandra 兼容）的文档历史记录

下表列出了自 Amazon Keyspaces（Apache Cassandra 兼容）上一次发布以来对文档所做的重要更改。要获得本文档的更新通知，您可以订阅 RSS 源。

- 最新文档更新：2024 年 2 月 7 日

变更	说明	日期
从亚马逊 Elastic Kubernetes 服务连接到亚马逊密钥空间 Elastic Kubernetes Service	现在，您可以按照 step-by-step 教程从 Amazon EKS 连接到 Amazon Keyspaces。	2024 年 2 月 7 日
Amazon Keyspaces 为预配置的表自动扩展 API	Amazon Keyspaces 现在提供CQL了 AWS API 支持，用于在预配置容量模式下设置自动扩展。	2024 年 1 月 23 日
Amazon Keyspaces 多区域复制支持预配置表	Amazon Keyspaces 现在支持多区域表的预配置容量模式。	2024 年 1 月 23 日
日志中包含 Amazon Keyspaces 的 DML 活动 CloudTrail	现在，您可以在 AWS CloudTrail中审计 Amazon Keyspaces 数据操作语言 (DML, Data Manipulation Language) API 调用。	2023 年 12 月 20 日
Amazon Keyspaces 支持 FROZEN 关键字	Amazon Keyspaces 现在针对集合数据类型支持 FROZEN 关键字。	2023 年 11 月 15 日
Amazon Keyspaces 托管式策略更新	Amazon Keyspaces 向 AmazonKeyspacesFullAccess 托管式策略增加了新权限，以允许通过接口 VPC 端点连接到 Amazon	2023 年 10 月 3 日

	Keyspaces 的客户端访问 Amazon EC2 实例，从而使用来自 VPC 的网络信息更新 Amazon Keyspaces `system.peers` 表。	
Amazon Keyspaces 托管式策略更新	Amazon Keyspaces 创建了一个新的 `AmazonKeyspacesReadOnlyAccess_v2` 托管式策略，以允许通过接口 VPC 端点连接到 Amazon Keyspaces 的客户端访问 Amazon EC2 实例，从而使用来自 VPC 的网络信息更新 Amazon Keyspaces `system.peers` 表。	2023 年 9 月 12 日
在 Amazon Keyspaces 中创建连接的最佳实践	了解如何改进和优化 Amazon Keyspaces 中的客户端驱动程序配置。	2023 年 6 月 30 日
系统键空间现已针对 Amazon Keyspaces 进行记录	了解存储在系统键空间中的内容以及如何在 Amazon Keyspaces 中查询它们以获取有用信息。	2023 年 6 月 21 日
Amazon Keyspaces 现已支持多区域复制	Amazon Keyspaces 多区域复制通过为您提供更高的容错能力、稳定性和恢复能力来帮助您维护分布在全球的应用程序。	2023 年 6 月 5 日

Amazon Keyspaces 托管式策略更新	Amazon Keyspaces 向 `AmazonKeyspacesFullAccess` 托管式策略增加了新权限，以允许 Amazon Keyspaces 在管理员创建多区域键空间时创建服务相关角色。	2023 年 6 月 5 日
Amazon Keyspaces 支持 IN 关键字	Amazon Keyspaces 现在支持在 SELECT 语句中使用 IN 关键字。	2023 年 4 月 25 日
Amazon Keyspaces 和接口 VPC 端点的跨账户访问	了解如何使用 VPC 端点实施针对 Amazon Keyspaces 的跨账户访问。	2023 年 4 月 20 日
Amazon Keyspaces 支持客户端时间戳	Amazon Keyspaces 客户端时间戳是与 Cassandra 兼容的单元格级时间戳，可在不同的客户端对相同的数据进行更改时帮助分布式应用程序确定写入操作的顺序。	2023 年 3 月 14 日
开始使用 Amazon Keyspaces 和接口 VPC 端点	在本 step-by-step 教程中，学习如何从 VPC 连接到 Amazon Keyspaces。	2023 年 3 月 1 日
优化 Amazon Keyspaces 表的成本	我们提供了最佳实践和指导，帮助您确定优化现有 Amazon Keyspaces 表的成本的策略。	2023 年 2 月 17 日
`Murmur3Partitioner` 现在是默认值	`Murmur3Partitioner` 现在是 Amazon Keyspaces 中的默认分区程序。	2022 年 11 月 17 日
Amazon Keyspaces 现已支持 `Murmur3Partitioner`	`Murmur3Partitioner` 现在可在 Amazon Keyspaces 中使用。	2022 年 11 月 9 日

对空字符串和 blob 值的支持更新	Amazon Keyspaces 现在还支持将空字符串和 blob 值作为聚类列值。	2022 年 10 月 19 日
Amazon Keyspaces 现已在 AWS GovCloud (US)	Amazon Keyspaces 现已在中推出， AWS GovCloud (US) Region 并且在 FedRAMP 高合规性范围内。有关可用端点的信息，请参阅 AWS GovCloud (US) Region FIPS 端点。	2022 年 8 月 4 日
通过亚马逊监控亚马逊 Keyspaces 表存储成本 CloudWatch	Amazon Keyspaces 现在可以帮助您通过该`BillableTableSizeInBytes` CloudWatch 指标监控和跟踪一段时间内的表存储成本。	2022 年 6 月 14 日
Amazon Keyspaces 现已支持 Terraform	您现在可以使用 Terraform 在 Amazon Keyspaces 中执行数据定义语言 (DDL) 操作。	2022 年 6 月 9 日
Amazon Keyspaces `token` 函数支持	Amazon Keyspaces 现在通过使用 `token` 函数来帮助您优化应用程序查询。	2022 年 4 月 19 日
Amazon Keyspaces 与 Apache Spark 集成	Amazon Keyspaces 现在通过使用开源 Spark Cassandra Connector 帮助您更轻松地在 Apache Spark 中读取和写入数据。	2022 年 4 月 19 日
Amazon Keyspaces API 参考	Amazon Keyspaces 支持控制平面操作，以使用软件开发工具包和表来管理密钥空间和表。 AWS AWS CLI API 参考指南详细介绍了支持的控制面板操作。	2022 年 3 月 2 日

如何排查使用 Amazon Keyspaces 时的常见配置问题	详细了解如何解决您在使用 Amazon Keyspaces 时可能遇到的常见配置问题。	2021 年 11 月 22 日
Amazon Keyspaces 支持存活时间 (TTL)	Amazon Keyspaces 存活时间 (TTL) 通过自动让表中的数据过期来帮助您简化应用程序逻辑和优化存储价格。	2021 年 10 月 18 日
使用 DSBulk 将数据迁移到 Amazon Keyspaces	使用 DataStax 批量加载器 (DSBulk) 将数据从 Apache Cassandra 迁移到亚马逊密钥空间的tep-by-step 教程。	2021 年 8 月 9 日
Amazon Keyspaces 支持在 `system.peers` 表中使用 VPC 端点条目	Amazon Keyspaces 让您可以使用可用的接口 VPC 端点信息填充 `system.peers` 表，以改善负载均衡和提高读取/写入吞吐量。	2021 年 7 月 29 日
更新 IAM 托管策略以支持客户托管 AWS KMS 密钥。	Amazon Keyspaces 的 IAM 托管策略现在包括列出和查看存储在中的可用客户托管 AWS KMS 密钥的权限。 AWS KMS	2021 年 6 月 1 日
Amazon Keyspaces 支持客户管理 AWS KMS 的密钥。	Amazon Keyspaces 允许您控制存储在中的客户托管 AWS KMS 密钥， AWS KMS 以便进行静态加密。	2021 年 6 月 1 日
Amazon Keyspaces 支持 JSON 语法	Amazon Keyspaces 通过支持用于 INSERT 和 SELECT 操作的 JSON 语法来帮助您更轻松地读取和写入 JSON 文档。	2021 年 1 月 21 日

Amazon Keyspaces 支持静态列	Amazon Keyspaces 现在通过使用静态列来帮助您在多个行之间高效地更新和存储常用数据。	2020 年 11 月 9 日
NoSQL Workbench 支持 Amazon Keyspaces 的正式版发布	NoSQL Workbench 是一款客户端应用程序，可帮助您更轻松地设计和可视化用于 Amazon Keyspaces 的非关系数据模型。NoSQL Workbench 客户端可用于 Windows、macOS 和 Linux。	2020 年 10 月 28 日
NoSQL Workbench 支持 Amazon Keyspaces 的预览版发布	NoSQL Workbench 是一款客户端应用程序，可帮助您更轻松地设计和可视化用于 Amazon Keyspaces 的非关系数据模型。NoSQL Workbench 客户端可用于 Windows、macOS 和 Linux。	2020 年 10 月 5 日
以编程方式访问 Amazon Keyspaces 的新代码示例	我们将继续增加以编程方式访问 Amazon Keyspaces 的代码示例。支持 Apache Cassandra 版本 3.11.2 的 Java、Python、Go、C# 和 Perl Cassandra 驱动程序的示例现已推出。	2020 年 7 月 17 日
Amazon Keyspaces point-in-time 恢复 (PITR)	Amazon Keyspaces 现在提供 point-in-time 恢复 (PITR) 功能，通过为您提供表数据的连续备份，帮助保护您的表免受意外写入或删除操作的影响。	2020 年 7 月 9 日

Amazon Keyspaces 正式发布	利用 Amazon Keyspaces [以前在预览版中称为 Amazon Managed Apache Cassandra Service (MCS)]，您可以使用 Cassandra 查询语言 (CQL) 代码、获得 Apache 2.0 许可的 Cassandra 驱动程序以及您现在已经使用的开发人员工具。	2020 年 4 月 23 日
Amazon Keyspaces 自动扩缩	Amazon Keyspaces（Apache Cassandra 兼容）与 Application Auto Scaling 集成，通过自动调整吞吐容量来帮助您高效地为可变工作负载预置吞出容量，从而应对实际应用程序流量。	2020 年 4 月 23 日
用于 Amazon Keyspaces 的接口虚拟私有云 (VPC) 端点	Amazon Keyspaces 提供服务和您的 VPC 之间的私有通信，使网络流量不会离开 Amazon 网络。	2020 年 4 月 16 日
基于标签的访问策略	您现在可以在 IAM 策略中使用资源标签来管理对 Amazon Keyspaces 的访问。	2020 年 4 月 8 日
计数器数据类型	Amazon Keyspaces 现在可帮助您使用计数器来协调列值的增量和减量。	2020 年 4 月 7 日
为资源添加标签	Amazon Keyspaces 现在可让您使用标签对资源进行标记和分类。	2020 年 3 月 31 日
AWS CloudFormation 支持	Amazon Keyspaces 现在可帮助您使用 AWS CloudFormation来自动创建和管理资源。	2020 年 3 月 25 日

支持 IAM 角色和策略以及 SigV4 身份验证	添加了有关如何使用 AWS Identity and Access Management (IAM) 管理亚马逊密钥空间的访问权限和实施安全策略以及如何使用适用于 Cassandra 的 DataStax Java 驱动程序的身份验证插件使用 IAM 角色和联合身份以编程方式访问亚马逊密钥空间的信息。	2020 年 3 月 17 日
读取/写入容量模式	Amazon Keyspaces 现在支持两种读取/写入吞出容量模式。读取/写入容量模式控制对读取和写入吞吐量收费的方式以及管理表吞出容量的方式。	2020 年 2 月 20 日
初始版本	本文档介绍了 Amazon Keyspaces（Apache Cassandra 兼容）的初始版本。	2019 年 12 月 3 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。