aws

开发人员指南

Amazon Timestream

Amazon Timestream: 开发人员指南

Table of Contents

亚马逊 Timestream 有什么用？ LiveAnalytics

Amazon Timestream for LiveAnalytics 是一个快速、可扩展、完全托管、专门构建的时间序列数据库，可轻松存储和分析每天数万亿个时间序列数据点。Timestream for 通过将最新数据保存在内存中，并根据用户定义的策略将历史数据移动到成本优化的存储层，为您 LiveAnalytics 节省管理时间序列数据生命周期的时间和成本。Timestream for LiveAnalytics 的专用查询引擎允许您同时访问和分析最新数据和历史数据，而无需指定其位置。Amazon Timestream LiveAnalytics 具有内置的时间序列分析功能，可帮助您近乎实时地识别数据中的趋势和模式。的 Timestream LiveAnalytics 是无服务器的，可以自动向上或向下扩展以调整容量和性能。由于您无需管理底层基础架构，因此可以专注于优化和构建应用程序。

Timestream LiveAnalytics 还与用于数据收集、可视化和机器学习的常用服务集成。你可以将数据发送到亚马逊 Timestream 供 LiveAnalytics 使用 AWS IoT Core、亚马逊 Kinesis、亚马逊和开源 T MSK elegraf。您可以通过使用 Amazon QuickSight、Grafana 和商业智能工具对数据进行可视化。JDBC您还可以使用 SageMaker 带有 Timestream 的 Amazon LiveAnalytics 进行机器学习。

LiveAnalytics 关键优势的时间流

Amazon Timestream 的主要优势是： LiveAnalytics

- 具有自动缩放功能的无服务器 ——使用 Amazon Timestream LiveAnalytics，无需管理服务器，也无需配置容量。随着应用程序需求的变化，Timestream for LiveAnalytics 会自动扩展以调整容量。
- 数据生命周期管理—— Amazon Timestream 用于 LiveAnalytics 简化数据生命周期管理的复杂流程。它提供存储分层，包括用于存储最新数据的内存存储和用于存储历史数据的磁性存储。Amazon Timestream 可根据用户可配置的策略自动将数据从存储器传输到磁性存储。
- 简化数据访问—— 借助 Amazon Timestream for LiveAnalytics，您无需再使用不同的工具来访问最新和历史数据。Amazon Timestream LiveAnalytics 的专用查询引擎可以透明地跨存储层访问和组合数据，而无需您指定数据位置。
- 专为时间序列而构建- 您可以使用内置的时间序列函数快速分析时间序列数据SQL，用于平滑、近似和插值。Timestream LiveAnalytics 还支持高级聚合、窗口函数以及复杂的数据类型，例如数组和行。
- 始终加密—— Amazon Timestream for LiveAnalytics 可确保您的时间序列数据始终处于加密状态，无论是静态数据还是传输中的数据。Amazon Timestream LiveAnalytics 还允许您指定 AWS KMS 客户托管密钥 (CMK)，用于加密磁性存储中的数据。

- 高可用性- Amazon Timestream 通过自动复制数据并在单个区域内至少 3 个不同的可用区分配资源，确保您的写入和读取请求的高可用性。 AWS 有关更多信息，请参阅 Timestream 服务等级协议。
- 耐久性- Amazon Timestream 通过在单个区域内的不同可用区自动复制内存和磁性存储数据，确保数据的持久性。 AWS 在确认您的写入请求完成之前，您的所有数据都将写入磁盘。

用例的时间 LiveAnalytics 流

Timestream 的用例列表越来越多，例如 LiveAnalytics ：

- 监控指标以提高应用程序的性能和可用性。
- 存储和分析工业遥测数据，以简化设备管理和维护。
- 跟踪一段时间内用户与应用程序的互动。
- 存储和分析物联网传感器数据。

开始使用 Timestream LiveAnalytics

我们建议您首先阅读以下部分：

- 教程- 创建填充了示例数据集的数据库并运行示例查询。
- 概念 Amazon Timestream LiveAnalytics - 学习 LiveAnalytics 概念的基本时间流。
- 访问 Timestream LiveAnalytics- 要了解如何通过控制台访问 Tim LiveAnalytics estream AWS CLI，或API。
- 配额- 了解您可以预置的 LiveAnalytics 组件的 Timestream 数量配额。

要了解如何快速开始为 Timestream 开发应用程序 LiveAnalytics，请参阅以下内容：

- 使用 AWS SDKs
- 查询语言参考

工作方式

以下各节概述了 Amazon Timestream for Live Analytics 服务组件及其交互方式。

阅读本简介后，请参阅相关访问 Timestream LiveAnalytics章节，了解如何使用控制台 AWS CLI、或SDKs访问实时分析的 Timestream。

主题

概念 Amazon Timestream LiveAnalytics

时间序列数据是在一个时间间隔内记录的一系列数据点。此类数据用于测量随时间推移而变化的事件。示例包括以下内容。

- 一段时间内的股票价格
- 一段时间内的温度测量
- CPU一段时间内EC2实例的利用率

对于时间序列数据，每个数据点都由一个时间戳、一个或多个属性以及随时间变化的事件组成。这些数据可用于深入了解应用程序的性能和运行状况，检测异常并识别优化机会。例如， DevOps 工程师可能希望查看衡量基础设施性能指标变化的数据。制造商可能希望跟踪物联网传感器数据，以测量整个设施的设备变化。在线营销人员可能想分析点击流数据，以捕捉用户在一段时间内浏览网站的方式。由于时间序列数据是从多个来源生成的，因此需要以近乎实时的方式进行经济实惠的收集，因此需要高效的存储来帮助组织和分析数据。

以下是 Timestream 的 LiveAnalytics关键概念。

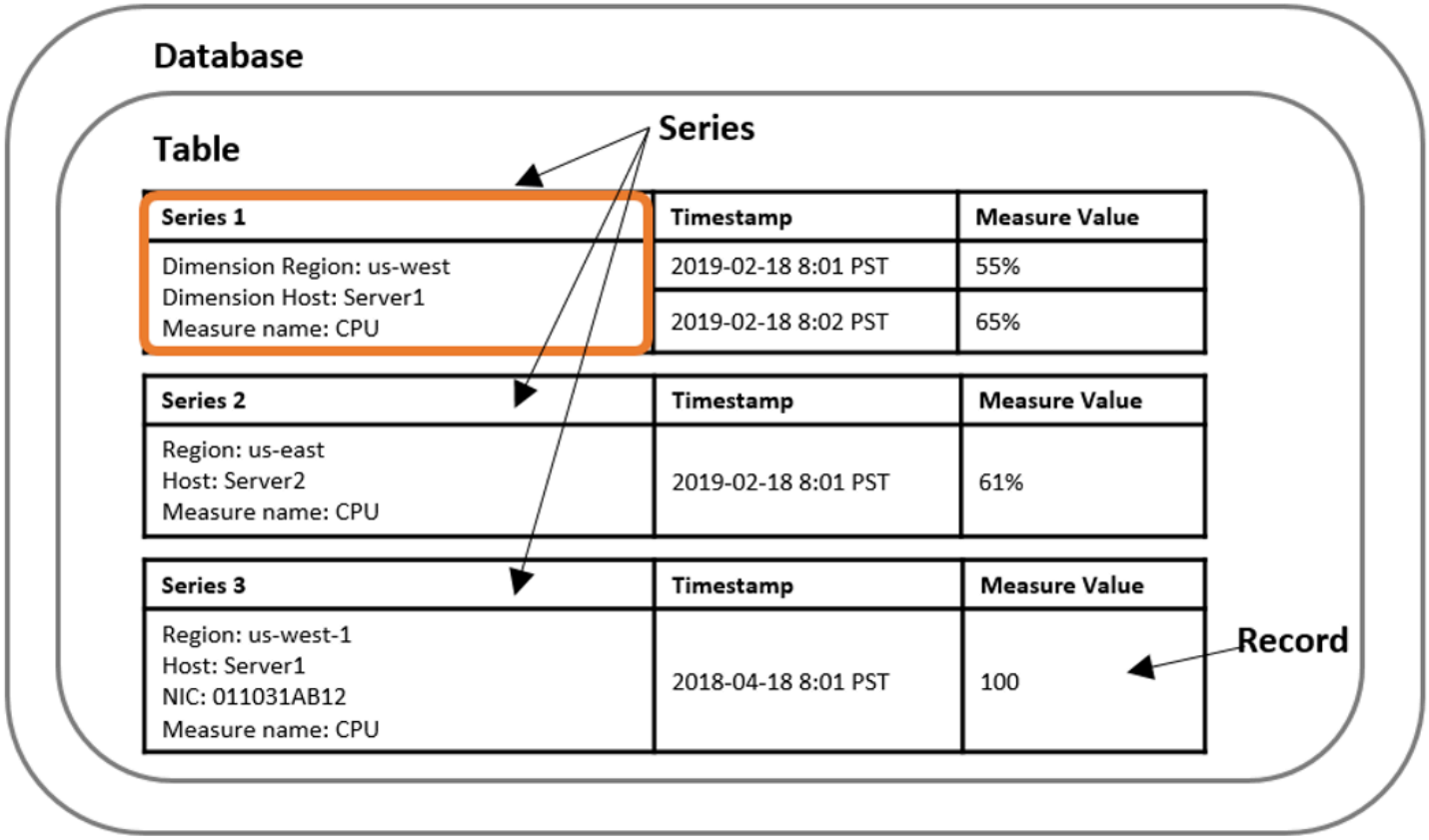
- 时间序列-在一个时间间隔内记录的一个或多个数据点（或记录）的序列。例如，一段时间内的股票价格、一段时间内EC2实例的内存使用率以及物联网传感器随时间推移的温度/压力读数。CPU
- 记录-时间序列中的单个数据点。
- 维度-描述时间序列元数据的属性。维度由维度名称和维度值组成。考虑以下示例：
  - 将证券交易所视为维度时，维度名称为 “证券交易所”，维度值为 “NYSE”

  - 将 AWS 区域视为维度时，维度名称为 “区域”，维度值为 “us-east-1”
  - 对于物联网传感器，维度名称为 “设备 ID”，维度值为 “12345”
- 测量-记录所测量的实际值。例如股票价格、CPU或内存利用率以及温度或湿度读数。度量由度量名称和度量值组成。考虑以下示例：
  - 对于股票价格，度量名称为 “股票价格”，衡量值是某个时间点的实际股价。
  - 对于CPU利用率，度量名称为 “CPU利用率”，度量值为实际CPU利用率。

  可以在 Timestream 中将度量建模 LiveAnalytics 为多度量或单度量记录。有关更多信息，请参阅 多度量记录与单度量记录。
- 时间戳-表示何时为给定记录收集度量。timestream LiveAnalytics 支持纳秒粒度的时间戳。
- 表-一组相关时间序列的容器。
- 数据库-表格的顶级容器。

概念时间流摘要 LiveAnalytics

一个数据库包含 0 个或多个表。每个表包含 0 个或多个时间序列。每个时间序列都由给定时间间隔内按指定粒度的一系列记录组成。每个时间序列都可以使用其元数据或维度、其数据或度量以及时间戳来描述。

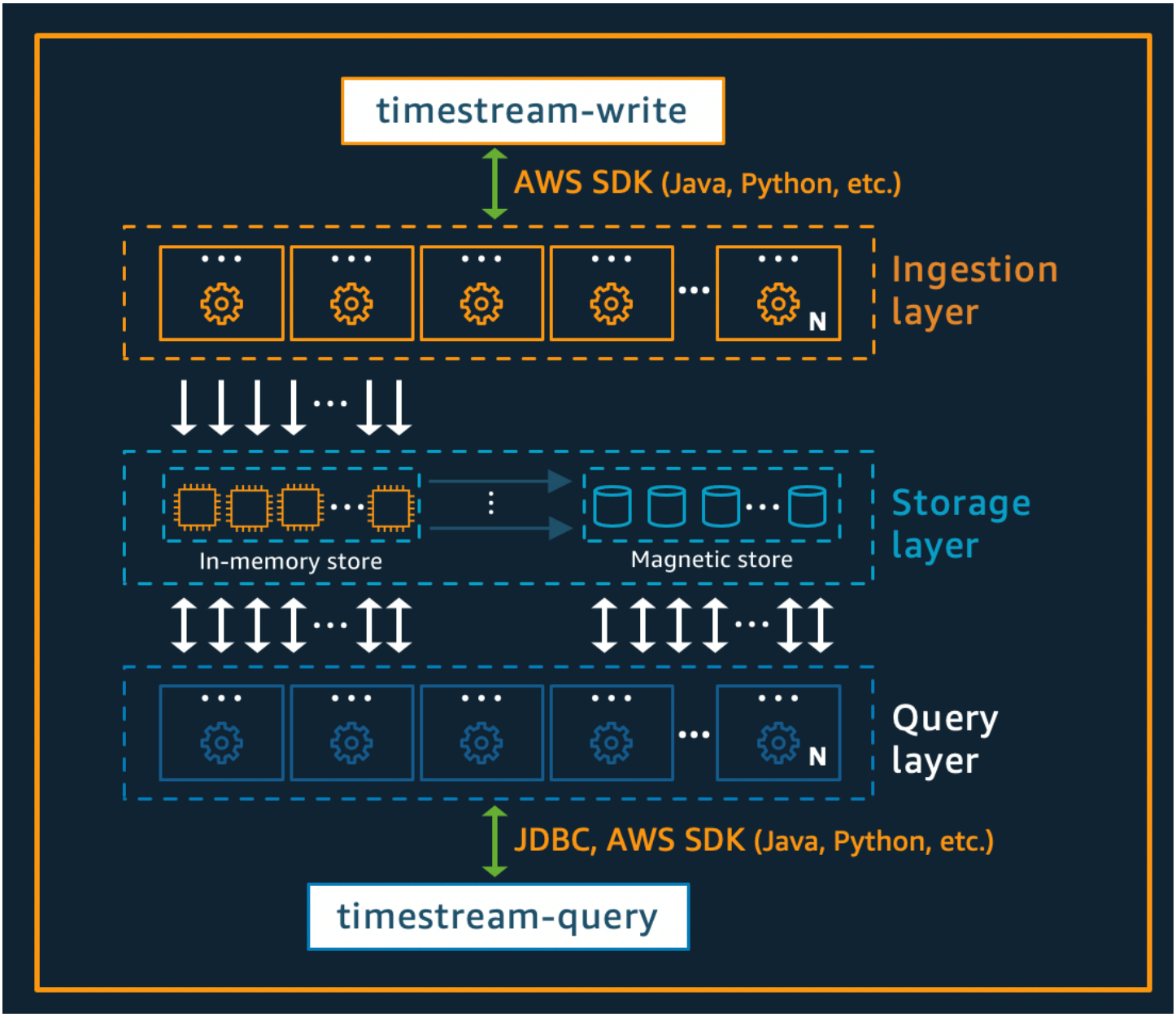


架构

用于实时分析的 Amazon Timestream 是从头开始设计的，旨在大规模收集、存储和处理时间序列数据。其无服务器架构支持完全分离的数据摄取、存储和查询处理系统，这些系统可以独立扩展。这种设计简化了每个子系统，可以更轻松地实现坚定不移的可靠性，消除扩展瓶颈，并减少相关系统故障的机会。随着系统的扩展，这些因素中的每一个都变得越来越重要。

主题

- 写架构
- 存储架构
- 查询架构
- 蜂窝架构



写架构

在写入时间序列数据时，Amazon Timestream for Live Analytics 会将表、分区的写入路由到处理高吞吐量数据写入的容错内存存储实例。反过来，内存存储在独立的存储系统中实现了持久性，该存储系统将数据复制到三个可用区（AZs）。复制基于法定人数，因此节点或整个可用区的丢失不会中断写入可用性。其他内存存储节点近乎实时地同步到数据以提供查询。读取器副本节点AZs也跨越多个节点，以确保高读取可用性。

Timestream for Live Analytics 支持将数据直接写入磁存储，适用于生成吞吐量较低的延迟数据的应用程序。迟到的数据是指时间戳早于当前时间的数据。与内存存储中的高吞吐量写入类似，写入磁存储的数据将跨三次复制，AZs并且复制基于法定人数。

无论数据是写入内存还是磁性存储，Timestream for Live Analytics 都会在将数据写入存储之前自动对其进行索引和分区。一个 Timestream for Live Analytics 表可能有数百、数千甚至数百万个分区。各个分区不直接相互通信，也不共享任何数据（无共享架构）。相反，表的分区是通过高度可用的分区跟踪和索引服务来跟踪的。这提供了另一种关注点分离，专为最大限度地减少系统故障的影响并大大降低相关故障的可能性而设计。

存储架构

当数据存储在 Timestream for Live Analytics 中时，将根据随数据一起写入的上下文属性按时间顺序和跨时间组织数据。拥有除时间之外还能分割 “空间” 的分区方案对于大规模扩展时间序列系统非常重要。这是因为大多数时间序列数据是在当前时间或大约当前时间写入的。因此，仅按时间进行分区并不能很好地分配写入流量或允许在查询时对数据进行有效修剪。这对于极大规模的时间序列处理非常重要，它使Timestream for Live Analytics能够以无服务器方式比当今其他领先系统高出几个数量级。生成的分区之所以被称为 “切片”，是因为它们代表二维空间的分区（设计为大小相似）。Live Analytics 表的时间流从单个分区（切片）开始，然后根据吞吐量的需要在空间维度中进行拆分。当切片达到一定大小时，它们会在时间维度上进行拆分，以便随着数据大小的增长实现更好的读取并行性。

Timestream for Live Analytics 旨在自动管理时间序列数据的生命周期。Timestream for Live Analytics 提供两个数据存储——一个内存存储和一个经济实惠的磁性存储。它还支持配置表级策略以自动跨商店传输数据。传入的高吞吐量数据写入存储器中，在内存存储中，数据会针对写入进行优化，也可以在当前时间左右执行读取，以便为仪表板供电和提醒类型查询。当写入、警报和仪表板制作需求的主要时间范围过去时，允许数据自动从内存存储流向磁性存储以优化成本。Timestream for Live Analytics 允许为此目的在内存存储上设置数据保留政策。为迟到的数据写入的数据会直接写入磁存储。

一旦数据在磁存储中可用（由于内存存储保留期到期或由于直接写入磁存储），就会将其重组为一种针对大容量数据读取进行了高度优化的格式。磁性存储器还具有数据保留策略，如果存在时间阈值使数据超过其用处，则可以配置该策略。当数据超过为磁性存储保留策略定义的时间范围时，会自动将其删除。因此，使用 Timestream for Live Analytics，除了某些配置外，数据生命周期管理可以在幕后无缝进行。

查询架构

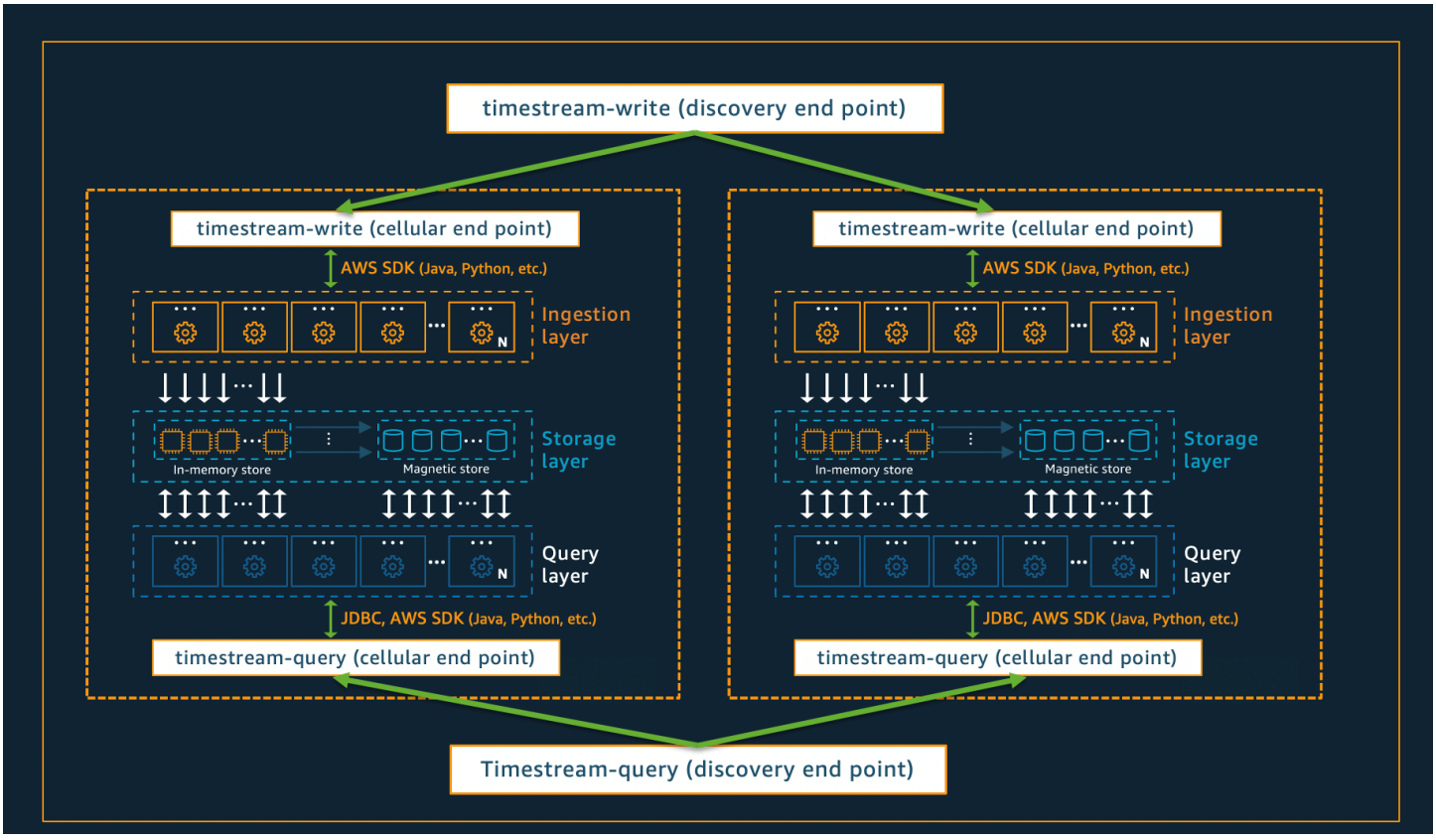
Live Analytics 查询的时间流以一种SQL语法表示，该语法具有针对特定时间序列的支持（特定于时间序列的数据类型和函数）的扩展，因此对于已经熟悉的开发人员来说，学习曲线很容易。SQL然后，查询由自适应分布式查询引擎处理，该引擎使用来自切片跟踪和索引服务的元数据，在发出查询时无缝访

问和合并跨数据存储的数据。这将带来一种能引起客户共鸣的体验，因为它将 Rube Goldberg 的许多复杂性整合为一个简单而熟悉的数据库抽象。

查询由专门的工作人员队伍运行，其中参与运行给定查询的工作人员数量由查询的复杂性和数据大小决定。对大型数据集进行复杂查询的性能是通过在查询运行时队列和系统的存储队列上的大规模并行性实现的。快速高效地分析海量数据的能力是Timestream for Live Analytics的最大优势之一。运行超过 TB 甚至 PB 数据的单个查询可能会让成千上万台计算机同时处理所有数据。

蜂窝架构

为了确保 Timestream for Live Analytics 可以为您的应用程序提供几乎无限的扩展，同时确保 99.99% 的可用性，该系统还使用蜂窝架构进行设计。Timestream for Live Analytics 没有将整个系统扩展为多个较小的副本，称为单元格。这允许对细胞进行全面测试，并防止一个细胞中的系统问题影响给定区域中任何其他细胞的活性。虽然 Timestream for Live Analytics 旨在支持每个区域的多个单元格，但请考虑以下虚构场景，即一个区域中有 2 个单元。



在上面描述的场景中，数据摄取和查询请求首先分别由发现端点处理，分别用于数据摄取和查询。然后，发现端点识别包含客户数据的单元，并将请求定向到该单元的相应摄取或查询端点。使用时 SDKs，这些端点管理任务将以透明方式为您处理。

> **Note**
>
> 将VPC端点与 Timestream 一起用于实时分析或直接访问用于实时分析的 Timestream 的RESTAPI操作时，您需要直接与蜂窝终端节点进行交互。有关如何执行此操作的指导，请参阅VPC端点以获取有关如何设置VPC端点的说明，以及有关直接调用RESTAPI操作的说明，请参阅端点发现模式。

写入

您可以从联网设备、IT 系统和工业设备收集时间序列数据，并将其写入 Timestream 进行实时分析。当时间序列属于同一个表时，Timestream for Live Analytics 允许您在单个写入请求中写入来自单个时间序列的数据点和/或来自多个序列的数据点。为方便起见，Timestream for Live Analytics 为您提供了一个灵活的架构，该架构可根据您在调用数据库写入数据时指定的度量值的维度名称和数据类型，自动检测 Timestream for Live Analytics 表的列名和数据类型。您也可以将批量数据写入 Timestream 以进行实时分析。

> **Note**
>
> Live Analytics 的 Timestream 支持读取的最终一致性语义。这意味着，当您在将一批数据写入 Timestream for Live Analytics 后立即查询数据时，查询结果可能无法反映最近完成的写入操作的结果。结果可能还包括一些陈旧的数据。同样，在写入具有一个或多个新维度的时间序列数据时，查询可以在短时间内返回列的部分子集。如果您在短时间后重复这些查询请求，则结果应返回最新数据。

您可以使用、、或通过AWS SDKsAWS CLI、、AWS Lambda、AWS IoT Core适用于 Apache Flink的亚马逊托管服务Amazon KinesisAmazon MSK、和写入数据开源 Telegraf。

主题

- 数据类型
- 没有预先定义架构
- 写入数据（插入和向上移动）
- 读取的最终一致性
- 使用批量写入 WriteRecords API
- 批量加载

- 在 WriteRecords API操作和批量加载之间进行选择

数据类型

Live Analytics 的 Timestream 支持以下数据类型进行写入。

数据类型	描述
BIGINT	表示 64 位有符号整数。
BOOLEAN	代表逻辑的两个真值，即真值和假值。
DOUBLE	实现二进制浮点运算IEEE标准 754 的 64 位可变精度。 **Note** 有查询语言函数`Infinity`和`NaN`双精度值可以在查询中使用。但是你不能将这些值写入 Timestream。
VARCHAR	可变长度字符数据，最大长度可选。最大限制为 2 KB。
MULTI	多度量记录的数据类型。此数据类型包括一个或多个类型为`BIGINT`、`BOOLEAN`、`DOUBLE`、`VARCHAR`和的度量`TIMESTAMP` 。
TIMESTAMP	使用纳秒精度时间表示时间中的实例UTC，跟踪自 Unix 时间以来的时间。目前仅支持多度量记录（即在该类型的度量值内`MULTI`）使用此数据类型。 *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.*sssssssss* 将支持时间戳写入`1970-01-01 00:00:00.000000000` 到的范围内。`2262-04-11 23:47:16.854775807`

没有预先定义架构

在将数据发送到 Amazon Timestream 进行实时分析之前，您必须使用用于实时分析的时间流或用于实时分析SDKs操作的时间流创建数据库和表。 AWS Management Console API有关更多信息，请参阅创建 数据库 和创建表。创建表时，您无需预先定义架构。Amazon Timestream for Live Analytics 会

根据发送的数据点的度量和维度自动检测架构，因此您无需再离线更改架构即可使其适应快速变化的时间序列数据。

写入数据（插入和向上移动）

Amazon Timestream for Live Analytics 中的写入操作使您能够插入和更新数据。默认情况下，在 Amazon Timestream for Live Analytics 中写入数据遵循第一个写入者获胜的语义，即数据仅存储为附加，重复的记录将被拒绝。虽然第一个写入者获胜语义可以满足许多时间序列应用程序的要求，但在某些情况下，应用程序需要以等性方式更新现有记录和/或使用最后一个写入者获胜语义写入数据，在这种情况下，版本最高的记录存储在服务中。为了解决这些情况，Amazon Timestream for Live Analytics 提供了更新数据的能力。Upsert 是一种在记录不存在时将记录插入系统的操作，或者在记录存在时更新该记录的操作。当记录更新时，它会以等性方式更新。

没有记录级别的删除操作。但是可以删除表和数据库。

将数据写入存储器和磁性存储器

Amazon Timestream for Live Analytics 能够将数据直接写入内存存储和磁性存储。内存存储针对高吞吐量数据写入进行了优化，而磁存储则针对较低吞吐量的延迟数据写入进行了优化。

迟到的数据是指时间戳早于当前时间且超出内存存储保留期的数据。必须通过启用磁存储写入表来明确启用将迟到的数据写入磁存储的功能。此外，`MagneticStoreRejectedDataLocation`是在创建表时定义的。要写入磁性存储，的调用者`WriteRecords`必须拥有在创建表`MagneticStoreRejectedDataLocation`期间对中指定的 S3 存储桶的`S3:PutObject`权限。有关更多信息，请参阅CreateTableWriteRecords、和PutObject。

使用单度记录和多度量记录写入数据

Amazon Timestream for Live Analytics 提供了使用两种类型的记录写入数据的功能，即单量记录和多度量记录。

单项测量记录

单项测量记录使您可以为每条记录发送一个度量。当使用这种格式将数据发送到 Timestream 进行实时分析时，实时分析的 Timestream 会为每条记录创建一个表格行。这意味着，如果一台设备发出 4 个指标，并且每个指标都作为单一指标记录发送，则 Timestream for Live Analytics 将在表中创建 4 行来存储这些数据，并且每行都会重复设备属性。如果您想监控应用程序中的单个指标，或者您的应用程序不同时发出多个指标，则建议使用这种格式。

多重测量记录

使用多度量记录，您可以在单个表格行中存储多个度量，而不是在每个表行中存储一个度量。因此，多指标记录使您能够将现有数据从关系数据库迁移到用于实时分析的 Amazon Timestream，只需进行最少的更改。

您还可以在一次写入请求中批处理比单一测量记录更多的数据。这提高了数据写入吞吐量和性能，还降低了数据写入成本。这是因为在写入请求中批量处理更多数据可以让 Amazon Timestream for Live Analytics 在单个写入请求中识别更多可重复的数据（如果适用），并且只需为重复的数据收取一次费用。

主题

多重测量记录

借助多重测量记录，您可以将时间序列数据以更紧凑的格式存储在存储器和磁性存储器中，这有助于降低数据存储成本。此外，紧凑的数据存储有助于编写更简单的查询以进行数据检索，提高查询性能并降低查询成本。

此外，多度量记录还支持在时间序列记录中存储多个时间戳TIMESTAMP的数据类型。TIMESTAMP多度量记录中的属性支持 future 或过去的时间戳。因此，多度量记录有助于提高性能、降低成本和简化查询，并为存储不同类型的关联度量提供了更大的灵活性。

优势

以下是使用多重测量记录的好处。

- 性能和成本-多指标记录使您能够在单个写入请求中写入多个时间序列度量。这增加了写入吞吐量并降低了写入成本。借助多重测量记录，您可以以更紧凑的方式存储数据，这有助于降低数据存储成本。多度量记录的紧凑数据存储减少了查询处理的数据。这旨在提高整体查询性能并帮助降低查询成本。
- 查询简单性-使用多度量记录，您无需在查询中编写复杂的公用表表达式 (CTEs) 即可读取具有相同时间戳的多个度量。这是因为度量以列的形式存储在单个表行中。因此，多重测量记录可以编写更简单的查询。
- 灵活的数据建模 — 您可以使用TIMESTAMP数据类型和多度量记录将未来的时间戳写入 Timestream 以进行实时分析。除了记录中的时间字段外，多度量记录还可以具有多个TIMESTAMP数据类型的属性。TIMESTAMP在多度量记录中，属性可以具有未来或过去的时间戳，其行为类似于时间字段，唯一的不同是实时分析的 Timestream 不对多度量记录TIMESTAMP中的类型值进行索引。

使用案例

您可以将多测量记录用于任何时间序列应用程序，该应用程序在任何给定时间从同一设备生成多个测量值。以下是一些示例应用程序。

- 一种在给定时间生成数百个指标的视频流媒体平台。
- 生成血氧水平、心率和脉搏等测量值的医疗设备。
- 工业设备，例如石油钻井平台，可生成指标、温度和天气传感器。
- 使用一个或多个微服务架构的其他应用程序。

示例：监控视频流应用程序的性能和运行状况

假设一个在 200 个EC2实例上运行的视频流应用程序。您想使用 Amazon Timestream for Live Analytics 来存储和分析应用程序发出的指标，这样您就可以了解应用程序的性能和运行状况，快速识别异常情况，解决问题并发现优化机会。

我们将使用单测量记录和多度量记录对这种情况进行建模，然后比较/对比这两种方法。对于每种方法，我们都做出以下假设。

- 每个EC2实例每秒发出四个度量（video_startup_time、rebuffering_ratio、video_playback_failers 和 average_frame_rate）和四个维度（device_id、device_type、os_version和区域）。
- 您想在存储器中存储 6 小时的数据，在磁性存储中存储 6 个月的数据。
- 为了识别异常情况，您设置了 10 次查询，每分钟运行一次，以识别过去几分钟内的任何异常活动。您还构建了一个包含八个小部件的仪表板，用于显示最近 6 小时的数据，因此您可以有效地监控应用程序。该仪表板可在任何给定时间由五个用户访问，并且每小时自动刷新一次。

使用单项测量记录

数据建模：使用单一测量记录，我们将为四个测量值（视频启动时间、重新缓冲比率、视频播放失败和平均帧速率）中的每一个创建一条记录。每条记录将有四个维度（device_id、device_type、os_version 和区域）和一个时间戳。

写入：当您将数据写入 Amazon Timestream 进行实时分析时，记录的构造方式如下。

```
public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
```

```
    List<Dimension> dimensions = new ArrayList<>();

    final Dimension device_id = new
Dimension().withName("device_id").withValue("12345678");
    final Dimension device_type = new
Dimension().withName("device_type").withValue("iPhone 11");
    final Dimension os_version = new
Dimension().withName("os_version").withValue("14.8");
    final Dimension region = new Dimension().withName("region").withValue("us-east-1");

    dimensions.add(device_id);
    dimensions.add(device_type);
    dimensions.add(os_version);
    dimensions.add(region);

    Record videoStartupTime = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_startup_time")
        .withMeasureValue("200")
        .withMeasureValueType(MeasureValueType.BIGINT)
        .withTime(String.valueOf(time));
    Record rebufferingRatio = new Record()
        .withDimensions(dimensions)
        .withMeasureName("rebuffering_ratio")
        .withMeasureValue("0.5")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));
    Record videoPlaybackFailures = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_playback_failures")
        .withMeasureValue("0")
        .withMeasureValueType(MeasureValueType.BIGINT)
        .withTime(String.valueOf(time));
    Record averageFrameRate = new Record()
        .withDimensions(dimensions)
        .withMeasureName("average_frame_rate")
        .withMeasureValue("0.5")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    records.add(videoStartupTime);
    records.add(rebufferingRatio);
    records.add(videoPlaybackFailures);
```

```
    records.add(averageFrameRate);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

存储单项测量记录时，数据的逻辑表示方式如下。

时间	device_id	设备类型	os_version	region	measure_name	measure_value::bigint	measure_value::double
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	video_startup_time	200	
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	rebuffering_ration		0.5

时间	device_id	设备类型	os_version	region	measure_name	measure_value::bigint	measure_value::double
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	视频播放失败	0	
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	平均帧速率		0.85
2021-09-07 21:53:44.00	12345678	iPhone 11	14.8	us-east-1	video_startup_time	500	
2021-09-07 21:53:44.00	12345678	iPhone 11	14.8	us-east-1	rebuffering_ration		1.5
2021-09-07 21:53:44.00	12345678	iPhone 11	14.8	us-east-1	视频播放失败	10	
2021-09-07 21:53:44.00	12345678	iPhone 11	14.8	us-east-1	平均帧速率		0.2

查询：您可以编写一个查询，检索过去 15 分钟内收到的具有相同时间戳的所有数据点，如下所示。

```
with cte_video_startup_time as ( SELECT time, device_id, device_type, os_version,
 region, measure_value::bigint as video_startup_time FROM table where time >= ago(15m)
 and measure_name=”video_startup_time”),
cte_rebuffering_ratio as ( SELECT time, device_id, device_type, os_version, region,
 measure_value::double as rebuffering_ratio FROM table where time >= ago(15m) and
 measure_name=”rebuffering_ratio”),
cte_video_playback_failures as ( SELECT time, device_id, device_type, os_version,
 region, measure_value::bigint as video_playback_failures FROM table where time >=
 ago(15m) and measure_name=”video_playback_failures”),
cte_average_frame_rate as ( SELECT time, device_id, device_type, os_version, region,
 measure_value::double as average_frame_rate FROM table where time >= ago(15m) and
 measure_name=”average_frame_rate”)
SELECT a.time, a.device_id, a.os_version, a.region, a.video_startup_time,
 b.rebuffering_ratio, c.video_playback_failures, d.average_frame_rate FROM
 cte_video_startup_time a, cte_buffering_ratio b, cte_video_playback_failures c,
 cte_average_frame_rate d WHERE
a.time = b.time AND a.device_id = b.device_id AND a.os_version = b.os_version AND
 a.region=b.region AND
a.time = c.time AND a.device_id = c.device_id AND a.os_version = c.os_version AND
 a.region=c.region AND
a.time = d.time AND a.device_id = d.device_id AND a.os_version = d.os_version AND
 a.region=d.region
```

工作量成本：这种工作量的成本估计为每月373.23美元，如果有单一测量记录

使用多度量记录

数据建模：对于多度量记录，我们将创建一个包含所有四个度量（视频启动时间、重新缓冲比率、视频播放失败率和平均帧速率）、所有四个维度（device_id、deversion_type、os_version和区域）以及时间戳的记录。

写入：当您将数据写入 Amazon Timestream 进行实时分析时，记录的构造方式如下。

```
public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();

    final Dimension device_id = new
 Dimension().withName("device_id").withValue("12345678");
```

```
    final Dimension device_type = new
Dimension().withName("device_type").withValue("iPhone 11");
    final Dimension os_version = new
Dimension().withName("os_version").withValue("14.8");
    final Dimension region = new Dimension().withName("region").withValue("us-east-1");

    dimensions.add(device_id);
    dimensions.add(device_type);
    dimensions.add(os_version);
    dimensions.add(region);

    Record videoMetrics = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_metrics")
        .withTime(String.valueOf(time));
        .withMeasureValueType(MeasureValueType.MULTI)
        .withMeasureValues(
          new MeasureValue()
         .withName("video_startup_time")
         .withValue("0")
         .withValueType(MeasureValueType.BIGINT),
         new MeasureValue()
 .withName("rebuffering_ratio")
         .withValue("0.5")
         .withType(MeasureValueType.DOUBLE),
          new MeasureValue()
         .withName("video_playback_failures")
         .withValue("0")
         .withValueType(MeasureValueType.BIGINT),
  new MeasureValue()
          .withName("average_frame_rate")
         .withValue("0.5")
         .withValueType(MeasureValueType.DOUBLE))

    records.add(videoMetrics);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
```

```
      System.out.println("WriteRecords Status: " +
 writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

存储多度量记录时，数据的逻辑表示方式如下。

时间	device_id	设备类型	os_version	region	measure_name	video_startup_time	rebuffering_ratio	视频_播放失败	平均帧速率
2021-09-7 21:48:440	1234567	iPhone 11	14.8	us-east-1	视频指标	200	0.5	0	0.85
2021-09-7 21:53:440	1234567	iPhone 11	14.8	us-east-1	视频指标	500	1.5	10	0.2

查询：您可以编写一个查询，检索过去 15 分钟内收到的具有相同时间戳的所有数据点，如下所示。

```
SELECT time, device_id, device_type, os_version, region, video_startup_time,
 rebuffering_ratio, video_playback_failures, average_frame_rate FROM table where time
 >= ago(15m)
```

工作负载成本：如果有多项记录，工作量成本估计为127.43美元。

> **Note**
>
> 在这种情况下，使用多指标记录可将每月的总体估计支出减少2.5倍，数据写入成本降低3.3倍，存储成本降低3.3倍，查询成本降低1.2倍。

使用过去或将来存在的时间戳写入数据

Timestream for Live Analytics 提供了通过几种不同的机制写入具有位于内存存储保留窗口之外的时间戳的数据的能力。

- 磁性存储写入 — 您可以通过磁存储写入将迟到的数据直接写入磁存储。要使用磁存储写入，必须先为表启用磁存储写入。然后，您可以使用与向内存存储中写入数据相同的机制将数据提取到表中。Amazon Timestream for Live Analytics 将根据其时间戳自动将数据写入磁性存储。

  > **Note**
  >
  > 磁存储的 write-to-read延迟可能长达 6 小时，这与将数据写入内存存储不同，后者的 write-to-read延迟在亚秒范围内。

- TIMESTAMP度量的数据类型-您可以使用该TIMESTAMP数据类型来存储过去、现在或未来的数据。除了记录中的时间字段外，多度量记录还可以具有多个TIMESTAMP数据类型的属性。TIMESTAMP在多度量记录中，属性可以具有未来或过去的时间戳，其行为类似于时间字段，唯一的不同是实时分析的 Timestream 不对多度量记录TIMESTAMP中的类型值进行索引。

  > **Note**
  >
  > 仅多度量记录支持该TIMESTAMP数据类型。

读取的最终一致性

Live Analytics 的 Timestream 支持读取的最终一致性语义。这意味着，当您在将一批数据写入 Timestream for Live Analytics 后立即查询数据时，查询结果可能无法反映最近完成的写入操作的结果。如果您在短时间后重复这些查询请求，则结果应返回最新数据。

使用批量写入 WriteRecords API

Amazon Timestream for Live Analytics 允许您在单个写入请求中写入来自单个时间序列的数据点和/或来自多个序列的数据点。从性能和成本的角度来看，在一次写入操作中批处理多个数据点是有益的。有关更多详细信息写入，请参阅 “计量和定价” 部分。

> **Note**
>
> 您对实时分析Timestream的写入请求可能会受到限制，因为Live Analytics的时间流会根据应用程序的数据摄取需求进行扩展。如果您的应用程序遇到限制异常，则必须继续以相同（或更高）的吞吐量发送数据，以允许 Timestream for Live Analytics 自动扩展以满足应用程序的需求。

批量加载

通过 Amazon Timestream 的批量加载 LiveAnalytics，您可以将存储在 Amazon S3 中的CSV文件批量提取到 Timestream 中。借助这项新功能，您 LiveAnalytics 无需依赖其他工具或编写自定义代码，即可在 Timestream 中保存数据。您可以使用批量加载来回填具有灵活等待时间的数据，例如查询或分析不需要立即使用的数据。

您可以使用、和 AWS Management Console AWS CLI，创建批量加载任务 AWS SDKs。有关更多信息，请参阅 在控制台中使用批量加载、将批量加载与 AWS CLI 和 将批量加载与 AWS SDKs。

有关批量加载的更多信息，请参阅在 Timestream 中使用批量加载 LiveAnalytics。

在 WriteRecords API操作和批量加载之间进行选择

通过该 WriteRecords API操作，您可以将流式传输时间序列数据写入 Timestream，使其与系统生成的数据相同。 LiveAnalytics 通过使用 WriteRecords，您可以持续实时摄取单个数据点或小批量数据。Timestream for 为您 LiveAnalytics提供了一个灵活的架构，该架构根据您在调用数据库写入数据时指定的数据点的维度名称和数据类型，自动检测 LiveAnalytics 表的 Timestream 的列名和数据类型。

相比之下，批量加载允许使用您定义的数据模型将批处理的时间序列数据从源文件（CSV文件）稳健地导入到 Timestream 中。 LiveAnalytics关于何时对源文件使用批量加载的几个示例，包括 LiveAnalytics 通过概念验证批量导入时间序列数据以评估Timestream，从离线一段时间的物联网设备批量导入时间序列数据，以及将历史时间序列数据从 Amazon S3 迁移到 Timestream LiveAnalytics 有关批量加载的信息，请参见在 Timestream 中使用批量加载 LiveAnalytics。

这两种解决方案都是安全、可靠和高性能的。

WriteRecords 在以下情况下使用：

- 每次请求流式传输少量（小于 10 MB）的数据。
- 填充现有表。
- 从日志流中提取数据。
- 执行实时分析。
- 需要更低的延迟。

在以下情况下使用批量加载：

- 在文件中提取源自 Amazon S3 的大量数据。CSV有关 限制的更多信息，请参阅 配额。
- 填充新表，例如在数据迁移的情况下。
- 用历史数据丰富数据库（提取到新表中）。
- 您的源数据变化缓慢或根本没有变化。
- 您可以灵活地等待时间，因为在资源可用之前，批量加载任务可能处于待处理状态，尤其是在加载大量数据的情况下。Batch Load 适用于不需要随时可供查询或分析的数据以提高清晰度。

存储

Timestream for Live Analytics 存储和组织您的时间序列数据，以优化查询处理时间并降低存储成本。它提供数据存储分层，并支持两个存储层：内存存储和磁性存储。内存存储针对高吞吐量数据写入和快速 point-in-time查询进行了优化。磁存储器针对吞吐量较低的延迟数据写入、长期数据存储和快速分析查询进行了优化。

Timestream for Live Analytics 通过在单个可用区内自动复制不同可用区的内存和磁性存储数据，从而确保数据的持久性。 AWS 区域在确认您的写入请求完成之前，您的所有数据都将写入磁盘。

Timestream for Live Analytics 允许您配置保留策略，将数据从内存存储移动到磁性存储。当数据达到配置值时，Timestream for Live Analytics 会自动将数据移动到磁性存储。您也可以在磁性存储器上设置保留值。当数据从磁性存储中过期时，它将被永久删除。

例如，假设您将内存存储配置为保存一周的数据，将磁性存储配置为存放 1 年的数据。数据的年龄是使用与数据点关联的时间戳计算的。当存储器中的数据保存一周后，它会自动移至磁性存储区。然后，它会保存在磁性存储中一年时间。当数据存在一年后，它就会从用于实时分析的 Timestream 中删除。内存存储和磁性存储的保留值累积定义了您的数据将存储在 Timestream for Live Analytics 中的时间。

这意味着，对于上述场景，从数据到达之时起，数据将存储在 Timestream for Live Analytics 中，总时间为 1 年零 1 周。

> **Note**
>
> 当您升级内存或磁性存储器的保留期时，保留期更改将从那时起生效。例如，如果将内存存储的保留期设置为 2 小时，然后通过更新表保留策略将其更改为 24 小时，则内存存储将能够保存 24 小时的数据，但在进行此更改后 22 小时将填充 24 小时的数据。Live Analytics 的 Timestream 不会从磁性存储中检索数据来填充内存存储。

为确保时间序列数据的安全性，默认情况下，实时分析的 Timestream 中的数据始终处于加密状态。这适用于传输中的数据和静态数据。此外，Timestream for Live Analytics 使您能够使用客户管理的密钥来保护磁性存储中的数据。有关客户托管密钥的更多信息，请参阅AWS KMS keys。

查询

借助 Timestream for Live Analytics DevOps，您可以轻松存储和分析物联网应用的传感器数据、用于设备维护的工业遥测数据以及许多其他用例的指标。Timestream for Live Analytics 中专门构建的自适应查询引擎允许您使用单个语句访问跨存储层的数据。SQL它可以透明地跨存储层访问和组合数据，而无需您指定数据位置。您可以使用SQL查询 Timestream for Live Analytics 中的数据，以便从一个或多个表中检索时间序列数据。您可以访问数据库和表的元数据信息。用于实时分析的 Timestream SQL 还支持用于时序分析的内置函数。您可以参考查询语言参考参考以了解更多详细信息。

Timestream for Live Analytics 旨在实现完全分离的数据摄取、存储和查询架构，其中每个组件都可以独立于其他组件进行扩展（使其能够为应用程序的需求提供几乎无限的扩展）。这意味着，当您的应用程序每天发送数百 TB 的数据或运行数百万个处理少量或大量数据的查询时，Timestream for Live Analytics 不会 “翻转”。随着数据随时间的推移而增长，Timestream for Live Analytics 的查询延迟基本保持不变。这是因为 Timestream for Live Analytics 查询架构可以利用大量的并行性来处理更大的数据量，并自动扩展以满足应用程序的查询吞吐量需求。

数据模型

Timestream 支持两种查询数据模型——平面模型和时间序列模型。

> **Note**
>
> Timestream 中的数据使用平面模型存储，它是查询数据的默认模型。时间序列模型是一个查询时间的概念，用于时间序列分析。

- 平面模型
- 时间序列模型

平面模型

平面模型是 Timestream 的默认查询数据模型。它以表格格式表示时间序列数据。维度名称、时间、度量名称和度量值显示为列。表中的每一行都是一个原子数据点，对应于时间序列中特定时间的测量结果。时间流数据库、表和列有一些命名限制。中描述了这些内容服务限制。

下表显示了一个说明性示例，说明当数据作为单一测量记录发送时，Timestream 如何存储代表EC2实例利用率、内存利用率和网络活动的数据。CPU在这种情况下，维度是区域、可用区、虚拟私有云和EC2实例IDs的实例。衡量标准是EC2实例的CPU利用率、内存利用率和传入的网络数据。区域、az、vpc 和 instance_id 列包含维度值。时间列包含每条记录的时间戳。measure_name 列包含由 CPU 利用率、内存利用率和网络字节数_in表示的度量名称。measure_value:: double 列包含作为双精度发出的测量值（例如CPU利用率和内存利用率）。measure_value:: bigint 列包含以整数形式发出的测量值，例如传入的网络数据。

时间	region	az	vpc	instance_id	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000 000000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	CPU_利用率	35.0	null
2019-12-04 19:00:01.000 000 000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	CPU_利用率	38.2	null
2019-12-04 19:00:02.	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	CPU_利用率	45.3	null

时间	region	az	vpc	instance_id	measure_name	measure_value::double	measure_value::bigint
000 000 000 000							
2019-12-04 19:00:00.000 000000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	54.9	null
2019-12-04 19:00:01.000 000 000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	42.6	null
2019-12-04 19:00:02.000 000 000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	33.3	null
2019-12-04 19:00:00.000 000000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	网络字节	34,400	null
2019-12-04 19:00:01.000 000 000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	网络字节	1500	null

时间	region	az	vpc	instance_id	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:02.000 000 000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	网络字节	6000	null

下表显示了一个说明性示例，说明当数据作为多度CPU量记录发送时，Timestream 如何存储表示EC2实例利用率、内存利用率和网络活动的数据。

时间	region	az	vpc	instance_id	measure_ame	CPU_利用率	memory_ilization	网络字节
2019-12-04 19:00:00.000 00000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	指标	35.0	54.9	34,400
2019-12-04 19:00:01.000 000 000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	指标	38.2	42.6	1500
2019-12-04 19:00:02.000 000 000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	指标	45.3	33.3	6,600

时间序列模型

时间序列模型是一种用于时间序列分析的查询时间结构。它将数据表示为（时间、度量值）对的有序序列。Timestream 支持插值等时间序列函数，使您能够填补数据中的空白。要使用这些函数，必须使用诸如 create_time_series 之类的函数将数据转换为时间序列模型。有关更多查询语言参考详细信息，请参阅。

使用前面的EC2实例示例，以下是以时间序列表示的CPU利用率数据。

region	az	vpc	instance_id	CPU_利用率
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	[{时间：2019-12-04 19:00:00.000 00000 000，值：35}，{时间：2019-12-04 19:00:01.000 000 000 000 000，值：38.2}，{时间：2019-12-04 19:00:00:02.000 000 000 000，值：45.3}]

计划查询

Amazon Timestream for Live Analytics 中的计划查询功能是一种完全托管、无服务器且可扩展的解决方案，用于计算和存储聚合、汇总和其他形式的预处理数据，通常用于为操作仪表板、业务报告、临时分析和其他应用程序提供支持。计划查询可以提高实时分析的性能和成本效益，因此您可以从数据中获得更多见解，并可以继续做出更好的业务决策。

有关定时查询的更多信息，请参阅在 Timestream 中使用预定查询 LiveAnalytics。

时间流计算单元 () TCU

Amazon Timestream for Live Analytics 以 Timestream 计算单位 () 衡量分配给你的计算容量，以满足你的查询需求。TCU一个包TCU含 4 GB vCPUs 和 16 GB 的内存。当你在 Timestream for Live

Analytics 中运行查询时，该服务会根据查询的复杂性和正在处理的数据量TCUs按需分配。查询消耗TCUs的数量决定了相关成本。

> **Note**
>
> 2024 年 4 月 29 日之后加入 AWS 账户 该服务的所有内容都将默认为使用查询TCUs定价。

本主题内容

MaxQuery TCU

此设置指定服务在任何时间点为查询提供服务所使用的最大计算单元数。要运行查询，必须将最小容量设置为 4 TCUs。您可以以 4 TCUs 的倍数设置最大数量，例如 4、8、16、32 等。您只需为用于工作负载的计算资源付费。例如，如果您将最大值设置TCUs为 128，但始终仅使用 8 TCUs。仅在您使用 8 的时间内向您收费TCUs。您账户`MaxQueryTCU`中的默认值设置为 200。使用 AWS Management Console 或UpdateAccountSettingsAPI操作和或，可以在 4 到 1000 之间进行 AWS SDK 调整`MaxQueryTCU` AWS CLI。

我们建议`MaxQueryTCU`为您的账户设置。设置最大TCU限制可限制服务可用于查询工作负载的计算单元数量，从而有助于控制成本。这使您可以更好地预测和管理查询支出。

TCU 的计费

TCU每个按小时计费，精度为每秒，最少为 30 秒。这些计算单位的使用单位为 TCU-小时。

运行查询时，您需要为查询执行期间的TCUs使用量计费，以 TCU-小时为单位。例如：

- 您的工作负载在 3 小时内使用 20 TCUs。您需要支付 60 TCU 小时（20 TCUs x 3 小时）的费用。

- 您的工作负载在 30 分钟内使用 10TCUs，然后在接下TCUs来的 30 分钟内使用 20。您需要支付 15 TCU 小时的费用（ 10 TCUs x 0.5 小时 + 20 TCUs x 0.5 小时 ）。

每TCU小时的定价因而异。 AWS 区域有关更多详情，请参阅亚马逊 Timestream 定价。随着工作负载的增长，该服务会自动将计算容量扩展到指定的最大TCU限制 (MaxQueryTCU)，以保持稳定的性能。该MaxQueryTCU设置充当服务可以扩展到的计算容量的上限。此设置可帮助您控制计算资源的数量及其成本。

正在配置 TCU

当您加入服务时， AWS 账户 每个服务的默认MaxQueryTCU限制为 200。您可以根据需要随时使用 AWS Management Console 或UpdateAccountSettingsAPI操作更新此限制 AWS CLI。 AWS SDK

如果您不确定要配置的值，请监控您账户的QueryTCU指标。该指标可在 AWS Management Console 和 Amazon 中找到 CloudWatch。该指标可让您深入了解每分钟的最大TCUs使用次数。根据历史数据和您对 future 增长的估计，设置MaxQueryTCU以适应使用量的激增。我们建议留出比峰值使用量TCUs高出至少 4-16 的余量。例如，如果您在过去 30 天QueryTCU内的峰值为 128，我们建议将设置在 132 到 144 MaxQueryTCU 之间。

估算所需的计算单位

计算单元可以同时处理查询。要确定所需的计算单元数量，请考虑下表中的一般准则：

并发查询	TCUs
7	4
14	8
21	12

> **Note**
>
> - 这些是一般指导方针，所需的实际计算单元数取决于多个因素，例如：
> - 查询的有效并发性。
> - 查询模式。
> - 扫描的分区数。

- 其他特定于工作负载的特征。
- 本指南适用于扫描最后几分钟到一小时的数据并遵守 Timestream 查询最佳实践和数据建模指南的查询。
- 监控应用程序的性能和QueryTCU指标，以根据需要调整计算单位。

何时增加 MaxQuery TCU

在以下MaxQueryTCU情况下，您应该考虑增加：

- 您的查询消耗量峰值已接近或达到当前配置的最大查询量TCU。我们建议将最大查询数设置为比峰值消耗量TCU至少TCUs高 4-16。
- 您的查询返回了 4xx 错误，且消息 MaxQueryTCU已超出范围。如果您预计工作负载会按计划增加，请重新访问并TCU相应调整配置的最大查询次数。

何时减少 MaxQuery TCU

在以下MaxQueryTCU情况下，您应该考虑减少：

- 您的工作负载具有可预测且稳定的使用模式，并且您对自己的计算使用要求有很好的了解。将最大查询量降TCU至高TCU于峰值消耗量的 4-16 以内，有助于防止意外的使用量和成本。您可以使用UpdateAccountSettingsAPI操作修改该值。
- 随着时间的推移，您的工作负载的峰值使用量有所下降，这可能是由于应用程序的变化或用户行为模式的变化。降低最大值TCU可以帮助减少意外成本。

> **Note**
>
> 根据您当前的使用情况，减少最大TCU限额更改最多可能需要 24 小时才能生效。您只需为查询实际消耗TCUs的费用付费。除非您的工作负载使用TCUs这些TCU限制，否则设置更高的最大查询限制不会影响您的成本。

使用 CloudWatch 指标监控使用情况

为了监控您的TCU使用情况，实时分析的 Timestream 提供了以下 CloudWatch 指标：QueryTCU。该指标指定一分钟内使用的计算单元数，并且每分钟发出一次。您可以选择监控一分钟内TCUs使用的最大值和最小值。您还可以针对此指标设置警报，以实时跟踪您的查询成本。

了解计算单位使用情况的变化

根据多个参数，查询所需的计算资源数量可以增加或减少。例如，数据量、数据提取模式、查询延迟、查询形状、查询效率以及使用实时和分析查询的查询组合。这些参数可能导致工作负载所需的TCU单位更高或更低。在这些参数不变的稳定状态下，您可能会观察到工作负载所需的计算单元数量减少了。因此，这可以降低您的每月费用。

此外，如果您的工作负载或数据中的任何一个参数发生变化，则所需的计算单元数量可能会增加。当 Timestream 收到查询时，Timestream 会根据查询访问的数据分区来决定高效处理查询所需的计算资源数量。

根据您的摄取和查询访问模式，Timestream 会定期优化数据布局。为了提高性能，Timestream 通过将访问量较少的分区组合成单个分区或将热分区拆分为多个分区来执行优化。因此，同一查询使用的计算容量在不同的时间点可能会略有不同。

> **选择使用TCU定价进行查询**
>
> 作为现有用户，您可以一次性选择加入，以便更好地TCUs管理成本，并删除每次查询计量的最小字节数。你可以使用 AWS Management Console 或UpdateAccountSettingsAPI操作和或来选择加入。 AWS SDK AWS CLI在API操作中，将`QueryPricingModel`参数设置为`COMPUTE_UNITS`。
> 选择使用基于计算的定价模式是一种不可逆转的变化。

访问 Timestream LiveAnalytics

您可以使用控制台访问 Time LiveAnalytics stream，CLI或者。API有关访问的 Timestream 的信息 LiveAnalytics，请查看以下内容：

主题

- 注册获取 AWS 账户
- 创建具有管理访问权限的用户
- 提供访问时间 LiveAnalytics 流
- 授予编程式访问权限

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。

要注册 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
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，请为用户分配管理访问权限，并且只使用根用户来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。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

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 root 用户开启多重身份验证 (MFA)。

   有关说明，请参阅《用户指南》中的 “为 AWS 账户 root 用户（控制台）启用虚拟MFA设备” IAM。

创建具有管理访问权限的用户

1. 启用IAM身份中心。

   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，向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。

以具有管理访问权限的用户身份登录

- 要使用您的 Ident IAM ity Center 用户登录URL，请使用您在创建 Ident IAM ity Center 用户时发送到您的电子邮件地址的登录信息。

  有关使用 Ident IAM ity 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。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个遵循应用最低权限权限的最佳实践的权限集。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

提供访问时间 LiveAnalytics 流

访问 Timestream 所需的权限 LiveAnalytics 已授予管理员。对于其他用户，您应使用以下策略向他们授予 Timestream LiveAnalytics 访问权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "timestream:*",
        "kms:DescribeKey",
        "kms:CreateGrant",
        "kms:Decrypt",
        "dbqms:CreateFavoriteQuery",
        "dbqms:DescribeFavoriteQueries",
        "dbqms:UpdateFavoriteQuery",
        "dbqms:DeleteFavoriteQueries",
        "dbqms:GetQueryString",
        "dbqms:CreateQueryHistory",
        "dbqms:UpdateQueryHistory",
        "dbqms:DeleteQueryHistory",
        "dbqms:DescribeQueryHistory",
```

```
        "s3:ListAllMyBuckets"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 有关信息dbqms，请参阅数据库查询元数据服务的操作、资源和条件键。有关信息，kms请参阅密钥管理服务的操作、资源和条件密钥。AWS

授予编程式访问权限

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份 （在IAM身份中心管理的用户）	使用临时证书签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	按照您希望使用的界面的说明进行操作。 • 有关的 AWS CLI，请参阅《AWS Command Line Interface 用户指南》 AWS IAM Identity Center中的 “配置 AWS CLI 要使用”。 • 有关工具和的信息 AWS SDKs AWS APIs，请参阅《工具参考指南》中的 “IAM AWS SDKs身份中心身份验证”。

哪个用户需要编程式访问权限？	目的	方式
IAM	使用临时证书签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	按照IAM用户指南中的将临时证书与 AWS 资源配合使用中的说明进行操作。
IAM	（不推荐使用） 使用长期凭证签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	按照您希望使用的界面的说明进行操作。 • 有关信息 AWS CLI，请参阅《用户指南》中的使用IAM用户凭据进行身份验证。AWS Command Line Interface • 有关 AWS SDKs和工具，请参阅《工具参考指南》AWS SDKs和《工具参考指南》中的使用长期凭证进行身份验证。 • 有关信息 AWS APIs，请参阅《IAM用户指南》中的管理IAM用户访问密钥。

使用控制台

您可以使用适用于 Timestream Live Analytics 的 AWS 管理控制台来创建、编辑、删除、描述和列出数据库和表。您也可以使用控制台运行查询。

主题

- 教程
- 创建 数据库
- 创建表
- 运行查询
- 创建计划查询

- 删除计划查询
- 删除表
- 删除数据库
- 编辑表格
- 编辑数据库

教程

本教程向您展示如何创建填充了示例数据集的数据库以及如何运行示例查询。本教程中使用的示例数据集经常出现在物联网和 DevOps 场景中。物联网数据集包含时间序列数据，例如卡车的速度、位置和负载，用于简化车队管理并识别优化机会。该 DevOps 数据集包含网络和内存利用率等EC2CPU实例指标，用于提高应用程序性能和可用性。以下是本节中描述的说明的视频教程

按照以下步骤创建填充了示例数据集的数据库，并使用 AWS 控制台运行示例查询。

1. 打开控制AWS 台。
2. 在导航窗格中，选择 “数据库”
3. 单击 “创建数据库”。
4. 在创建数据库页面上，输入以下内容：
   - 选择配置-选择示例数据库。
   - 名称-输入您选择的数据库名称。
   - 选择示例数据集 —选择物联网和 DevOps.
   - 单击 “创建数据库”，创建一个包含两个表的数据库：IoT，并 DevOps 填充了示例数据。
5. 在导航窗格中，选择查询编辑器
6. 从顶部菜单中选择 “示例查询”。
7. 点击其中一个示例查询。这将带您回到查询编辑器，编辑器中填充了示例查询。
8. 单击 “运行” 运行查询并查看查询结果。

创建 数据库

按照以下步骤使用 AWS 控制台创建数据库。

1. 打开控制AWS 台。

2. 在导航窗格中，选择 “数据库”
3. 单击 “创建数据库”。
4. 在创建数据库页面上，输入以下内容。
   - 选择配置-选择标准数据库。
   - 名称-输入您选择的数据库名称。
   - 加密-选择KMS密钥或使用默认选项，如果您的账户中尚不存在密钥，Timestream Live Analytics 将在您的账户中创建KMS密钥。
5. 单击 “创建数据库” 来创建数据库。

创建表

按照以下步骤使用 AWS 控制台创建表。

1. 打开控制AWS 台。
2. 在导航窗格中，选择 “表”
3. 单击 “创建表”。
4. 在创建表格页面上，输入以下内容。
   - 数据库名称-选择在中创建的数据库的名称。创建 数据库
   - 表名-输入您选择的表名。
   - 内存存储保留期-指定要在内存存储中保留数据的时间。内存存储处理传入的数据，包括延迟到达的数据（时间戳早于当前时间的数据），并针对快速 point-in-time查询进行了优化。
   - 磁存储保留期-指定要在磁性存储中保留数据多长时间。磁性存储器专为长期存储而设计，并针对快速分析查询进行了优化。
5. 单击 “创建表”。

运行查询

按照以下步骤使用 AWS 控制台运行查询。

1. 打开控制AWS 台。
2. 在导航窗格中，选择查询编辑器
3. 在左窗格中，选择在中创建的数据库创建 数据库。
4. 在左窗格中，选择在中创建的数据库创建表。

5. 在查询编辑器中，您可以运行查询。要查看表中最新的 10 行，请运行：

```
SELECT * FROM <database_name>.<table_name> ORDER BY time DESC LIMIT 10
```

6. （可选）启用 “启用见解” 以获取有关查询效率的见解。

创建计划查询

按照以下步骤使用 AWS 控制台创建计划查询。

1. 打开控制AWS 台。
2. 在导航窗格中，选择计划查询。
3. 单击 “创建计划查询”。
4. 在 “查询名称” 和 “目标表” 部分，输入以下内容。
   - 名称-输入查询名称。
   - 数据库名称-选择在中创建的数据库的名称。创建 数据库
   - 表名-选择在中创建的表的名称。创建表
5. 在查询语句部分，输入有效的查询语句。然后单击 “验证查询”。
6. 在目标表模型中，为所有未定义的属性定义模型。您可以使用可视化生成器或JSON。
7. 在 “运行计划” 部分中，选择固定速率或 Chron 表达式。对于 chron 表达式，有关计划表达式的更多详细信息，请参阅计划查询的计划表达式。
8. 在SNS主题部分中，输入将用于通知SNS的主题。
9. 在错误日志报告部分，输入将用于报告错误的 S3 位置。

   请选择 Encryption key type（加密密钥类型）。
10. 在AWS KMS密钥的安全设置部分中，选择密 AWS KMS钥的类型。

    输入 Timestream LiveAnalytics 将用于运行计划查询的IAM角色。有关该角色所需权限和信任关系的详细信息，请参阅计划查询的IAM策略示例。
11. 单击 “创建计划查询”。

删除计划查询

按照以下步骤使用 AWS 控制台删除或禁用计划查询。

1. 打开控制AWS 台。
2. 在导航窗格中，选择计划查询
3. 选择在中创建的计划查询创建计划查询。
4. 选择操作。
5. 选择 “禁用” 或 “删除”。
6. 如果您选择了 “删除”，请确认操作并选择 “删除”。

删除表

按照以下步骤使用 AWS 控制台删除数据库。

1. 打开控制AWS 台。
2. 在导航窗格中，选择 “表”
3. 选择您在中创建的表创建表。
4. 单击 Delete（删除）。
5. 在确认框中键入 delete。

删除数据库

按照以下步骤使用 AWS 控制台删除数据库：

1. 打开控制AWS 台。
2. 在导航窗格中，选择 “数据库”
3. 选择您在创建数据库中创建的数据库。
4. 单击 Delete（删除）。
5. 在确认框中键入 delete。

编辑表格

按照以下步骤使用 AWS 控制台编辑表格。

1. 打开控制AWS 台。
2. 在导航窗格中，选择 “表”

3. 选择您在中创建的表创建表。
4. 单击 “编辑”
5. 编辑表格详细信息并保存。

- 内存存储保留期-指定要在内存存储中保留数据的时间。内存存储处理传入的数据，包括延迟到达的数据（时间戳早于当前时间的数据），并针对快速 point-in-time查询进行了优化。
- 磁存储保留期-指定要在磁性存储中保留数据多长时间。磁性存储器专为长期存储而设计，并针对快速分析查询进行了优化。

编辑数据库

按照以下步骤使用 AWS 控制台编辑数据库。

1. 打开控制AWS 台。
2. 在导航窗格中，选择 “数据库”
3. 选择您在创建数据库中创建的数据库。
4. 单击 “编辑”
5. 编辑数据库详细信息并保存。

访问 Amazon Timestream 以使用 LiveAnalytics AWS CLI

您可以使用 AWS Command Line Interface (AWS CLI) 从命令行控制多项 AWS 服务，并通过脚本自动执行这些服务。您可以将 AWS CLI 用于即席操作。您还可以使用它在实用程序脚本中嵌入用于 LiveAnalytics 操作的 Amazon Timestream。

必须先设置编程访问权限，然后才能将 Timestream AWS CLI 与 Timestream 配合使用。LiveAnalytics有关更多信息，请参阅 授予编程式访问权限。

有关中可用于 Timestream LiveAnalytics 查询API的所有命令的完整列表 AWS CLI，请参阅《AWS CLI 命令参考》。

有关中可用于 Timestream for W LiveAnalytics rite 的所有命令API的完整列表 AWS CLI，请参阅AWS CLI 命令参考。

主题

- 下载和配置 AWS CLI

下载和配置 AWS CLI

它们可以在 Windows、macOS 或 Linux 上 AWS CLI 运行。要下载、安装和配置它，请执行以下步骤：

1. AWS CLI 在 http://aws.amazon.com/cli 下载。
2. 按照《AWS Command Line Interface 用户指南》 AWS CLI中有关安装 AWS CLI和配置的说明进行操作。

使用 with Tim AWS CLI estream LiveAnalytics

命令行格式由用于 LiveAnalytics 操作名称的 Amazon Timestream 和该操作的参数组成。除此之外，还 AWS CLI 支持参数值的简写语法。JSON

help用于在 Timestream 中列出所有可用的命令。 LiveAnalytics例如：

```
aws timestream-write help
```

```
aws timestream-query help
```

您还可以使用 help 来描述特定命令并了解有关其用法的详细信息：

```
aws timestream-write create-database help
```

例如，要创建数据库，请执行以下操作：

```
aws timestream-write create-database --database-name myFirstDatabase
```

要创建启用磁性存储写入功能的表，请执行以下操作：

```
aws timestream-write create-table \
--database-name metricsdb \
--table-name metrics \
--magnetic-store-write-properties "{\"EnableMagneticStoreWrites\": true}"
```

要使用单一测量值记录写入数据，请执行以下操作：

```
aws timestream-write write-records \
--database-name metricsdb \
--table-name metrics \
--common-attributes "{\"Dimensions\":[{\"Name\":\"asset_id\", \"Value\":\"100\"}],
 \"Time\":\"1631051324000\",\"TimeUnit\":\"MILLISECONDS\"}" \
--records "[{\"MeasureName\":\"temperature\", \"MeasureValueType\":\"DOUBLE\",
\"MeasureValue\":\"30\"},{\"MeasureName\":\"windspeed\", \"MeasureValueType\":\"DOUBLE
\",\"MeasureValue\":\"7\"},{\"MeasureName\":\"humidity\", \"MeasureValueType\":\"DOUBLE
\",\"MeasureValue\":\"15\"},{\"MeasureName\":\"brightness\", \"MeasureValueType\":
\"DOUBLE\",\"MeasureValue\":\"17\"}]"
```

要使用多度量记录写入数据，请执行以下操作：

```
# wide model helper method to create Multi-measure records
function ingest_multi_measure_records {
  epoch=`date +%s`
  epoch+=$i

  # multi-measure records
  aws timestream-write write-records \
  --database-name $src_db_wide \
  --table-name $src_tbl_wide \
  --common-attributes "{\"Dimensions\":[{\"Name\":\"device_id\", \
              \"Value\":\"12345678\"},\
            {\"Name\":\"device_type\", \"Value\":\"iPhone\"}, \
            {\"Name\":\"os_version\", \"Value\":\"14.8\"}, \
            {\"Name\":\"region\", \"Value\":\"us-east-1\"} ], \
            \"Time\":\"$epoch\",\"TimeUnit\":\"MILLISECONDS\"}" \
--records "[{\"MeasureName\":\"video_metrics\", \"MeasureValueType\":\"MULTI\", \
  \"MeasureValues\": \
  [{\"Name\":\"video_startup_time\",\"Value\":\"0\",\"Type\":\"BIGINT\"}, \
  {\"Name\":\"rebuffering_ratio\",\"Value\":\"0.5\",\"Type\":\"DOUBLE\"}, \
  {\"Name\":\"video_playback_failures\",\"Value\":\"0\",\"Type\":\"BIGINT\"}, \
  {\"Name\":\"average_frame_rate\",\"Value\":\"0.5\",\"Type\":\"DOUBLE\"}]}]" \
--endpoint-url $ingest_endpoint \
  --region  $region
}

# create 5 records
for i in {100..105};
  do ingest_multi_measure_records $i;
```

```
done
```

如何查询表：

```
aws timestream-query query \
--query-string "SELECT time, device_id, device_type, os_version,
region, video_startup_time, rebuffering_ratio, video_playback_failures, \
average_frame_rate \
FROM metricsdb.metrics \
where time >= ago (15m)"
```

要创建计划查询，请执行以下操作：

```
aws timestream-query create-scheduled-query \
  --name scheduled_query_name \
  --query-string "select bin(time, 1m) as time, \
          avg(measure_value::double) as avg_cpu, min(measure_value::double) as min_cpu,
 region \
          from $src_db.$src_tbl where measure_name = 'cpu' \
          and time BETWEEN @scheduled_runtime - (interval '5' minute)  AND
 @scheduled_runtime \
          group by region, bin(time, 1m)" \
  --schedule-configuration "{\"ScheduleExpression\":\"$cron_exp\"}" \
  --notification-configuration "{\"SnsConfiguration\":{\"TopicArn\":\"$sns_topic_arn
\"}}" \
  --scheduled-query-execution-role-arn "arn:aws:iam::452360119086:role/
TimestreamSQExecutionRole" \
  --target-configuration "{\"TimestreamConfiguration\":{\
          \"DatabaseName\": \"$dest_db\",\
          \"TableName\": \"$dest_tbl\",\
          \"TimeColumn\":\"time\",\
          \"DimensionMappings\":[{\
            \"Name\": \"region\", \"DimensionValueType\": \"VARCHAR\"
          }],\
          \"MultiMeasureMappings\":{\
            \"TargetMultiMeasureName\": \"mma_name\",
            \"MultiMeasureAttributeMappings\":[{\
              \"SourceColumn\": \"avg_cpu\", \"MeasureValueType\": \"DOUBLE\",
 \"TargetMultiMeasureAttributeName\": \"target_avg_cpu\"
            },\
            { \
              \"SourceColumn\": \"min_cpu\", \"MeasureValueType\": \"DOUBLE\",
 \"TargetMultiMeasureAttributeName\": \"target_min_cpu\"
```

```
          }] \
        }\
        }}" \
--error-report-configuration "{\"S3Configuration\": {\
      \"BucketName\": \"$s3_err_bucket\",\
      \"ObjectKeyPrefix\": \"scherrors\",\
      \"EncryptionOption\": \"SSE_S3\"\
      }\
    }"
```

使用 API

除此之外 SDKs，Amazon Timestream 还 LiveAnalytics 提供通过终端节点发现模式的直接RESTAPI访问。下文描述了端点发现模式及其用例。

端点发现模式

由于 Timestream Live Analytics 旨在透明地与服务的架构（包括服务端点的管理和映射）配合使用，因此建议您SDKs对大多数应用程序使用。SDKs但是，在某些情况下，必须使用 Timestream 进行 LiveAnalytics RESTAPI端点发现模式：

- 您正在使用带有 T imestream 的VPC端点 (AWS PrivateLink) LiveAnalytics
- 您的应用程序使用的编程语言尚不SDK支持
- 您需要更好地控制客户端实现

本节包括有关端点发现模式的工作原理、如何实现端点发现模式以及使用说明的信息。在下面选择一个主题以了解更多信息。

主题

- 端点发现模式的工作原理
- 实现端点发现模式

端点发现模式的工作原理

Timestream 使用蜂窝架构构建，可确保更好的扩展和流量隔离特性。由于每个客户账户都映射到某个区域中的特定单元，因此您的应用程序必须使用您的账户已映射到的正确的单元格特定终端节点。使用时SDKs，系统会为您透明地处理此映射，您无需管理单元特定的端点。但是，直接访问时RESTAPI，您需要自己管理和映射正确的端点。这个过程，即端点发现模式，如下所述：

1. 端点发现模式从调用DescribeEndpoints操作开始（如DescribeEndpoints部分所述）。
2. 应在返回的 time-to-live (TTL) 值 (the) 指定的时间内缓存和重复使用端点。CachePeriodInMinutes然后，API可以在持续时间内调用 Timestream Live Analytics。TTL
3. TTL过期后， DescribeEndpoints 应重新调用以刷新终端节点（换句话说，从步骤 1 重新开始）。

> **Note**
>
> API参考资料中描述了该DescribeEndpoints操作的语法、参数和其他用法信息。请注意，该DescribeEndpoints操作可通过两者使用SDKs，并且每种操作都相同。

有关端点发现模式的实现，请参阅实现端点发现模式。

实现端点发现模式

要实现端点发现模式，请选择一个API（写入或查询），创建DescribeEndpoints请求，然后在返回TTL值的持续时间内使用返回的端点。实施过程如下所述。

> **Note**
>
> 请确保您熟悉使用说明。

实施程序

1. 使用DescribeEndpoints请求获取API您要对其进行调用（写入或查询）的终端节点。
   a. 使用下面描述DescribeEndpoints的两个端点之一创建与感兴趣的（写入或查询）相对应的请求。API该请求没有输入参数。请务必阅读以下注意事项。

      写下SDK：

      ```
      ingest.timestream.<region>.amazonaws.com
      ```

      查询SDK：

      ```
      query.timestream.<region>.amazonaws.com
      ```

us-east-1以下是区域CLI调用的示例。

```
REGION_ENDPOINT="https://query.timestream.us-east-1.amazonaws.com"
REGION=us-east-1
aws timestream-write describe-endpoints \
--endpoint-url $REGION_ENDPOINT \
--region $REGION
```

> **Note**
>
> HTTP“主机” 标头还必须包含API端点。如果未填充标头，则请求将失败。这是所有HTTP /1.1 请求的标准要求。如果您使用支持 1.1 或更高版本的HTTP库，则该库应自动为您填充标题。HTTP

> **Note**
>
> 替补 *<region>* 带有发出请求的区域的区域标识符，例如 us-east-1

b. 解析响应以提取端点和缓存TTL值。响应是一个或多个Endpoint对象的数组。每个Endpoint对象都包含一个端点地址 (Address) 和该端点TTL的 (CachePeriodInMinutes)。

2. 将终端节点缓存到指定值TTL。
3. TTL过期后，从实现的第 1 步重新开始检索新的端点。

端点发现模式的使用说明

- 该DescribeEndpoints操作是 Timestream Live Analytics 区域端点识别的唯一操作。
- 该响应包含用于API调用 Timestream Live Analytics 的端点列表。
- 成功响应后，列表中应至少有一个端点。如果列表中有多个端点，则其中任何一个都同样可用于API呼叫，并且呼叫者可以随机选择要使用的端点。
- 除了终端节点DNS的地址外，列表中的每个端点还将指定允许使用以分钟为单位的终端节点的生存时间 (TTL)。

- 端点应在返回TTL值指定的时间内缓存并重复使用（以分钟为单位）。TTL过期后，DescribeEndpoints应重新调用以刷新要使用的终端节点，因为终端节点在过期后将不再起作用。TTL

使用 AWS SDKs

您可以使用访问亚马逊 Timestream。 AWS SDKsTimestream SDKs 每种语言支持两种语言；即 “写入” SDK 和 “查询SDK”。写入SDK用于执行CRUD操作并将您的时间序列数据插入时间流。该查询SDK用于查询存储在 Timestream 中的现有时间序列数据。

完成自己选择的必要先决条件后，就可以开始使用代码示例。SDK

主题

- Java
- Java v2
- Go
- Python
- Node.js
- .NET

Java

要开始使用 Java 1.0 SDK 和 Amazon Timestream，请完成下述先决条件。

完成 Java 的必要先决条件后SDK，就可以开始使用了代码示例。

先决条件

在开始使用 Java 之前，必须执行以下操作：

1. 按照中的 AWS 设置说明进行操作访问 Timestream LiveAnalytics。
2. 通过下载并安装以下内容来设置一个 Java 开发环境：
   - Java SE 开发套件 8（例如 Amazon Corretto 8）。
   - JavaIDE（例如 Eclipse 或 Intelli J）。

     有关更多信息，请参阅《入门》 AWS SDK for Java
3. 配置您的 AWS 证书和区域以进行开发：

- 设置您的 AWS 安全证书，以便与一起使用 AWS SDK for Java。
- 设置您的 AWS 区域以确定 LiveAnalytics 终端节点的默认时间流。

使用 Apache Maven

你可以使用 Apache Maven 来配置和构建 AWS SDK for Java 项目。

> **Note**
>
> 要使用 Apache Maven，请确保你的 Java SDK 和运行时间为 1.8 或更高版本。

您可以按照与 A pache Maven SDK 一起使用中所述将依赖项配置 AWS SDK为 Mave n 依赖关系。

您可以使用以下命令运行 compile 并运行源代码：

```
mvn clean compile
mvn exec:java -Dexec.mainClass=<your source code Main class>
```

> **Note**
>
> `<your source code Main class>`是 Java 源代码主类的路径。

设置您的 AWS 凭证

AWS SDK for Java要求您在运行时向应用程序提供 AWS 凭据。本指南中的代码示例假设您使用的是 AWS 证书文件，如AWS SDK for Java 开发人员指南中的设置 AWS 证书和开发区域中所述。

以下是名为的 AWS 凭据文件的示例`~/.aws/credentials`，其中波浪号字符 (~) 表示您的主目录。

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java v2

要开始使用 Java 2.0 SDK 和 Amazon Timestream，请完成下述先决条件。

完成 Java 2.0 的必要先决条件后SDK，就可以开始使用了代码示例。

先决条件

在开始使用 Java 之前，必须执行以下操作：

1. 按照中的 AWS 设置说明进行操作访问 Timestream LiveAnalytics。
2. 您可以按照与 A pache Maven SDK 一起使用中所述将依赖项配置 AWS SDK为 Mave n 依赖关系。
3. 通过下载并安装以下内容来设置一个 Java 开发环境：

   - Java SE 开发套件 8（例如 Amazon Corretto 8）。
   - JavaIDE（例如 Eclipse 或 Intelli J）。

     有关更多信息，请参阅《入门》 AWS SDK for Java

使用 Apache Maven

你可以使用 Apache Maven 来配置和构建 AWS SDK for Java 项目。

> **Note**
>
> 要使用 Apache Maven，请确保你的 Java SDK 和运行时间为 1.8 或更高版本。

您可以按照与 A pache Maven SDK 一起使用中所述将依赖项配置 AWS SDK为 Mave n 依赖关系。此处描述了 pom.xml 文件所需的更改。

您可以使用以下命令运行 compile 并运行源代码：

```
mvn clean compile
mvn exec:java -Dexec.mainClass=<your source code Main class>
```

> **Note**
>
> `<your source code Main class>`是 Java 源代码主类的路径。

Go

要开始使用 Go SDK 和 Amazon Timestream，请完成以下前提条件。

完成 Go 的必要先决条件后SDK，就可以开始使用代码示例。

先决条件

1. 下载 GO SDK 1.14。
2. 配置 GO SDK。
3. 构建您的客户。

Python

要开始使用 Python SDK 和 Amazon Timestream，请完成以下前提条件。

完成 Python 的必要先决条件后SDK，就可以开始使用了代码示例。

先决条件

要使用 Python，请按照此处的说明安装和配置 Boto3。

Node.js

要开始使用 Node.js SDK 和 Amazon Timestream，请完成下述先决条件。

完成 Node.js 的必要先决条件后SDK，就可以开始使用代码示例。

先决条件

在开始使用 Node.js 之前，您必须执行以下操作：

1. 安装 Node.js。
2. 安装 fo AWS SDK r JavaScript。

.NET

要开始使用. NETSDK和 Amazon Timestream，完成前提条件，如下所述。

完成必需的先决条件后. NETSDK，你可以开始使用代码示例。

先决条件

在你开始之前。 NET，安装所需的 NuGet 软件包并通过运行以下命令确保 AWSSDK .Core 版本为 3.3.107 或更高版本：

```
dotnet add package AWSSDK.Core
dotnet add package AWSSDK.TimestreamWrite
dotnet add package AWSSDK.TimestreamQuery
```

开始使用

本部分包括帮助您开始使用 Amazon Timestream Live Analytics 的教程，以及设置功能齐全的示例应用程序的说明。您可以通过选择以下链接之一开始使用本教程或示例应用程序。

主题

- 教程
- 示例应用程序

教程

本教程向您展示如何创建填充了示例数据集的数据库以及如何运行示例查询。本教程中使用的示例数据集经常出现在物联网和 DevOps 场景中。物联网数据集包含时间序列数据，例如卡车的速度、位置和载荷，用于简化车队管理并识别优化机会。 DevOps 数据集包含网络和内存利用率等EC2CPU实例指标，用于提高应用程序性能和可用性。以下是本节中描述的说明的视频教程。

按照以下步骤创建填充了示例数据集的数据库，并使用 AWS 控制台运行示例查询：

使用 控制台

按照以下步骤创建填充了示例数据集的数据库，并使用 AWS 控制台运行示例查询：

1. 打开控制AWS 台。
2. 在导航窗格中，选择数据库
3. 单击 “创建数据库”。
4. 在创建数据库页面上，输入以下内容：
    - 选择配置-选择示例数据库。
    - 名称-输入您选择的数据库名称。

> **Note**
>
> 创建包含示例数据集的数据库后，要使用控制台中提供的示例查询，您可以调整查询中引用的数据库名称，使其与您在此处输入的数据库名称相匹配。样本数据集和时间序列记录类型的每种组合都有示例查询。

- 选择样本数据集 —选择物联网和 DevOps.
- 选择时间序列记录的类型-选择多度量记录。
- 单击 “创建数据库”，创建一个包含两个填充有示例数据的表的数据库。具有多度量记录的样本数据集的表名为`DevOpsMulti`和。`IoTMulti`具有单度量记录的样本数据集的表名为`DevOps`和。`IoT`

5. 在导航窗格中，选择查询编辑器
6. 从顶部菜单中选择 “示例查询”。
7. 单击您在创建示例数据库时选择的数据集的其中一个示例查询。这将带您回到查询编辑器，编辑器中填充了示例查询。
8. 调整示例查询的数据库名称。
9. 单击 “运行” 运行查询并查看查询结果。

使用 SDKs

Timestream Live Analytics 提供了一个功能齐全的示例应用程序，向您展示如何创建数据库和表、在表格中填充约12.6万行的样本数据以及如何运行示例查询。该示例应用程序可用GitHub于 Java、Python、Node.js、Go 和。 NET。

1. 按照中的说明克隆 GitHub 存储库 Timestream Live Analytics 示例应用程序。 GitHub
2. 按照中所述的说明进行配置 AWS SDK以连接至亚马逊 Timestream Live Analytics。使用 AWS SDKs
3. 按照以下说明编译并运行示例应用程序：

   - Java 示例应用程序的说明。
   - Java v2 示例应用程序的说明。
   - Go 示例应用程序的说明。
   - Python 示例应用程序的说明。
   - Node.js 示例应用程序的说明。

- 的说明. NET示例应用程序。

示例应用程序

Timestream 附带了一个功能齐全的示例应用程序，该应用程序展示了如何创建数据库和表、在表中填充大约 12.6K 行的示例数据以及如何运行示例查询。请按照以下步骤开始使用任何支持的语言版本的示例应用程序：

Java

1. 按照中的说明克隆 LiveAnalytics 示例应用程序的 GitHub 存储库 Timestream。GitHub
2. 按照入门中所述的说明将配置为 LiveAnalytics 连接到 Timestream。AWS SDK Java
3. 按照此处描述的说明运行 Java 示例应用程序

Java v2

1. 按照中的说明克隆 LiveAnalytics 示例应用程序的 GitHub 存储库 Timestream。GitHub
2. 按照入门中所述的说明进行 LiveAnalytics 配置 AWS SDK以连接到 Amazon Timestream。Java v2
3. 按照此处描述的说明运行 Java 2.0 示例应用程序

Go

1. 按照中的说明克隆 LiveAnalytics 示例应用程序的 GitHub 存储库 Timestream。GitHub
2. 按照入门中所述的说明进行 LiveAnalytics 配置 AWS SDK以连接到 Amazon Timestream。Go
3. 按照此处描述的说明运行 Go 示例应用程序

Python

1. 按照中的说明克隆 LiveAnalytics 示例应用程序的 GitHub 存储库 Timestream。GitHub
2. 按照中所述的说明进行配置 AWS SDK以连接到 Amazon Timestream。 LiveAnalytics Python
3. 按照此处描述的说明运行 Python 示例应用程序

Node.js

1. 按照中的说明克隆 LiveAnalytics 示例应用程序的 GitHub 存储库 Timestream。GitHub
2. 按照入门中所述的说明进行 LiveAnalytics 配置 AWS SDK以连接到 Amazon Timestream。Node.js
3. 按照此处描述的说明运行 Node.js 示例应用程序

.NET

1. 按照中的说明克隆 LiveAnalytics 示例应用程序的 GitHub 存储库 Timestream。GitHub
2. 按照入门中所述的说明进行 LiveAnalytics 配置 AWS SDK以连接到 Amazon Timestream。.NET
3. 运行. NET按照此处描述的说明进行操作的示例应用程序

代码示例

您可以使用访问亚马逊 Timestream。 AWS SDKsTimestream SDKs 每种语言支持两种语言；即 “写入” SDK 和 “查询SDK”。写入SDK用于执行CRUD操作并将您的时间序列数据插入时间流。该查询SDK用于查询存储在 Timestream 中的现有时间序列数据。从下面的列表中选择一个主题，了解更多详细信息，包括每个支持的主题的代码示例SDKs。

主题

- 写SDK客户端
- 查询SDK客户端
- 创建数据库
- 描述数据库
- 更新数据库
- 删除数据库
- 列出数据库
- 创建表
- 描述表
- 更新表
- 删除表

- 列出表
- 写入数据（插入和向上移动）
- 运行查询
- 运行UNLOAD查询
- 取消查询
- 创建批量加载任务
- 描述批量加载任务
- 列出批量加载任务
- 恢复批量加载任务
- 创建计划查询
- 列出计划查询
- 描述计划查询
- 执行预设查询
- 更新计划查询
- 删除计划查询

写SDK客户端

您可以使用以下代码片段为 Write 创建 Timestream 客户端。SDK写入SDK用于执行CRUD操作并将您的时间序列数据插入时间流。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
    private static AmazonTimestreamWrite buildWriteClient() {
        final ClientConfiguration clientConfiguration = new ClientConfiguration()
                .withMaxConnections(5000)
                .withRequestTimeout(20 * 1000)
                .withMaxErrorRetry(10);
```

```
        return AmazonTimestreamWriteClientBuilder
                .standard()
                .withRegion("us-east-1")
                .withClientConfiguration(clientConfiguration)
                .build();
    }
```

Java v2

```
    private static TimestreamWriteClient buildWriteClient() {
        ApacheHttpClient.Builder httpClientBuilder =
                ApacheHttpClient.builder();
        httpClientBuilder.maxConnections(5000);

        RetryPolicy.Builder retryPolicy =
                RetryPolicy.builder();
        retryPolicy.numRetries(10);

        ClientOverrideConfiguration.Builder overrideConfig =
                ClientOverrideConfiguration.builder();
        overrideConfig.apiCallAttemptTimeout(Duration.ofSeconds(20));
        overrideConfig.retryPolicy(retryPolicy.build());

        return TimestreamWriteClient.builder()
                .httpClientBuilder(httpClientBuilder)
                .overrideConfiguration(overrideConfig.build())
                .region(Region.US_EAST_1)
                .build();
    }
```

Go

```
tr := &http.Transport{
        ResponseHeaderTimeout: 20 * time.Second,
        // Using DefaultTransport values for other parameters: https://golang.org/
pkg/net/http/#RoundTripper
        Proxy: http.ProxyFromEnvironment,
        DialContext: (&net.Dialer{
            KeepAlive: 30 * time.Second,
            DualStack: true,
            Timeout:   30 * time.Second,
        }).DialContext,
```

```
        MaxIdleConns:          100,
        IdleConnTimeout:       90 * time.Second,
        TLSHandshakeTimeout:   10 * time.Second,
        ExpectContinueTimeout: 1 * time.Second,
    }

    // So client makes HTTP/2 requests
    http2.ConfigureTransport(tr)

    sess, err := session.NewSession(&aws.Config{ Region: aws.String("us-east-1"),
 MaxRetries: aws.Int(10), HTTPClient: &http.Client{ Transport: tr }})
    writeSvc := timestreamwrite.New(sess)
```

Python

```
write_client = session.client('timestream-write', config=Config(read_timeout=20,
 max_pool_connections = 5000, retries={'max_attempts': 10}))
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

此处显示了其他命令导入。创建客户端不需要CreateDatabaseCommand导入。

```
import { TimestreamWriteClient, CreateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
```

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
var https = require('https');
var agent = new https.Agent({
    maxSockets: 5000
});
writeClient = new AWS.TimestreamWrite({
        maxRetries: 10,
        httpOptions: {
            timeout: 20000,
            agent: agent
        }
```

```
    });
```

.NET

```
var writeClientConfig = new AmazonTimestreamWriteConfig
{
    RegionEndpoint = RegionEndpoint.USEast1,
    Timeout = TimeSpan.FromSeconds(20),
    MaxErrorRetry = 10
};

var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
```

我们建议您使用以下配置。

- 将SDK重试次数设置为。10
- 使用 SDK DEFAULT_BACKOFF_STRATEGY。
- 设置RequestTimeout为20秒。
- 将最大连接数设置为5000或更高。

查询SDK客户端

您可以使用以下代码片段为查询创建 Timestream 客户端。SDK该查询SDK用于查询存储在 Timestream 中的现有时间序列数据。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
    private static AmazonTimestreamQuery buildQueryClient() {
        AmazonTimestreamQuery client =
AmazonTimestreamQueryClient.builder().withRegion("us-east-1").build();
        return client;
    }
```

Java v2

```
    private static TimestreamQueryClient buildQueryClient() {
        return TimestreamQueryClient.builder()
                .region(Region.US_EAST_1)
                .build();
    }
```

Go

```
sess, err := session.NewSession(&aws.Config{Region: aws.String("us-east-1")})
```

Python

```
query_client = session.client('timestream-query')
```

Node.js

以下代码段用 AWS SDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅适用于 JavaScript v3 的 Timestream 查询客户端-。AWS SDK

此处显示了其他命令导入。创建客户端不需要QueryCommand导入。

```
import { TimestreamQueryClient, QueryCommand } from "@aws-sdk/client-timestream-
query";
const queryClient = new TimestreamQueryClient({ region: "us-east-1" });
```

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
queryClient = new AWS.TimestreamQuery();
```

.NET

```
var queryClientConfig = new AmazonTimestreamQueryConfig
{
    RegionEndpoint = RegionEndpoint.USEast1
};

var queryClient = new AmazonTimestreamQueryClient(queryClientConfig);
```

创建数据库

您可以使用以下代码片段来创建数据库。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
  public void createDatabase() {
       System.out.println("Creating database");
       CreateDatabaseRequest request = new CreateDatabaseRequest();
       request.setDatabaseName(DATABASE_NAME);
       try {
           amazonTimestreamWrite.createDatabase(request);
           System.out.println("Database [" + DATABASE_NAME + "] created
successfully");
       } catch (ConflictException e) {
           System.out.println("Database [" + DATABASE_NAME + "] exists. Skipping
database creation");
       }
   }
```

Java v2

```
   public void createDatabase() {
       System.out.println("Creating database");
       CreateDatabaseRequest request =
CreateDatabaseRequest.builder().databaseName(DATABASE_NAME).build();
       try {
           timestreamWriteClient.createDatabase(request);
           System.out.println("Database [" + DATABASE_NAME + "] created
successfully");
       } catch (ConflictException e) {
           System.out.println("Database [" + DATABASE_NAME + "] exists. Skipping
database creation");
       }
   }
```

Go

```
// Create database.
    createDatabaseInput := &timestreamwrite.CreateDatabaseInput{
        DatabaseName: aws.String(*databaseName),
    }

    _, err = writeSvc.CreateDatabase(createDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Database successfully created")
    }

    fmt.Println("Describing the database, hit enter to continue")
```

Python

```
    def create_database(self):
        print("Creating Database")
        try:
            self.client.create_database(DatabaseName=Constant.DATABASE_NAME)
            print("Database [%s] created successfully." % Constant.DATABASE_NAME)
        except self.client.exceptions.ConflictException:
            print("Database [%s] exists. Skipping database creation" %
 Constant.DATABASE_NAME)
        except Exception as err:
            print("Create database failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 CreateDatabaseCommand和CreateDatabase。

```
import { TimestreamWriteClient, CreateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
```

```
    DatabaseName: "testDbFromNode"
};

const command = new CreateDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} created successfully`);
} catch (error) {
    if (error.code === 'ConflictException') {
        console.log(`Database ${params.DatabaseName} already exists. Skipping
 creation.`);
    } else {
        console.log("Error creating database", error);
    }
}
```

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function createDatabase() {
    console.log("Creating Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.createDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} created
 successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Database ${params.DatabaseName} already exists.
 Skipping creation.`);
            } else {
                console.log("Error creating database", err);
            }
        }
    );
}
```

.NET

```
        public async Task CreateDatabase()
        {
            Console.WriteLine("Creating Database");

            try
            {
                var createDatabaseRequest = new CreateDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
                CreateDatabaseResponse response = await
writeClient.CreateDatabaseAsync(createDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Database already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create database failed:" + e.ToString());
            }

        }
```

描述数据库

您可以使用以下代码片段来获取有关新创建的数据库的属性的信息。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
    public void describeDatabase() {
        System.out.println("Describing database");
```

```
        final DescribeDatabaseRequest describeDatabaseRequest = new
 DescribeDatabaseRequest();
        describeDatabaseRequest.setDatabaseName(DATABASE_NAME);
        try {
            DescribeDatabaseResult result =
 amazonTimestreamWrite.describeDatabase(describeDatabaseRequest);
            final Database databaseRecord = result.getDatabase();
            final String databaseId = databaseRecord.getArn();
            System.out.println("Database " + DATABASE_NAME + " has id " +
 databaseId);
        } catch (final Exception e) {
            System.out.println("Database doesn't exist = " + e);
            throw e;
        }
    }
```

Java v2

```
    public void describeDatabase() {
        System.out.println("Describing database");
        final DescribeDatabaseRequest describeDatabaseRequest =
 DescribeDatabaseRequest.builder()
                .databaseName(DATABASE_NAME).build();
        try {
            DescribeDatabaseResponse response =
 timestreamWriteClient.describeDatabase(describeDatabaseRequest);
            final Database databaseRecord = response.database();
            final String databaseId = databaseRecord.arn();
            System.out.println("Database " + DATABASE_NAME + " has id " +
 databaseId);
        } catch (final Exception e) {
            System.out.println("Database doesn't exist = " + e);
            throw e;
        }
    }
```

Go

```
describeDatabaseOutput, err := writeSvc.DescribeDatabase(describeDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
```

```
    } else {
        fmt.Println("Describe database is successful, below is the output:")
        fmt.Println(describeDatabaseOutput)
    }
```

Python

```
    def describe_database(self):
        print("Describing database")
        try:
            result =
 self.client.describe_database(DatabaseName=Constant.DATABASE_NAME)
            print("Database [%s] has id [%s]" % (Constant.DATABASE_NAME,
 result['Database']['Arn']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Database doesn't exist")
        except Exception as err:
            print("Describe database failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 DescribeDatabaseCommand和DescribeDatabase。

```
import { TimestreamWriteClient, DescribeDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode"
};

const command = new DescribeDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} has id
 ${data.Database.Arn}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Database doesn't exist.");
```

```
    } else {
        console.log("Describe database failed.", error);
        throw error;
    }
}
```

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function describeDatabase () {
    console.log("Describing Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.describeDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} has id
 ${data.Database.Arn}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Database doesn't exist.");
            } else {
                console.log("Describe database failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DescribeDatabase()
        {
            Console.WriteLine("Describing Database");

            try
            {
                var describeDatabaseRequest = new DescribeDatabaseRequest
                {
```

```
                    DatabaseName = Constants.DATABASE_NAME
                };
                DescribeDatabaseResponse response = await
 writeClient.DescribeDatabaseAsync(describeDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} has id:
{response.Database.Arn}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Database does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe database failed:" + e.ToString());
            }

        }
```

更新数据库

您可以使用以下代码片段来更新数据库。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
    public void updateDatabase(String kmsId) {
        System.out.println("Updating kmsId to " + kmsId);
        UpdateDatabaseRequest request = new UpdateDatabaseRequest();
        request.setDatabaseName(DATABASE_NAME);
        request.setKmsKeyId(kmsId);
        try {
            UpdateDatabaseResult result =
 amazonTimestreamWrite.updateDatabase(request);
            System.out.println("Update Database complete");
        } catch (final ValidationException e) {
            System.out.println("Update database failed:");
```

```
            e.printStackTrace();
        } catch (final ResourceNotFoundException e) {
            System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
e);
        } catch (final Exception e) {
            System.out.println("Could not update Database " + DATABASE_NAME + " = "
+ e);
            throw e;
        }
    }
```

Java v2

```
    public void updateDatabase(String kmsKeyId) {

        if (kmsKeyId == null) {
            System.out.println("Skipping UpdateDatabase because KmsKeyId was not
given");
            return;
        }

        System.out.println("Updating database");

        UpdateDatabaseRequest request = UpdateDatabaseRequest.builder()
                .databaseName(DATABASE_NAME)
                .kmsKeyId(kmsKeyId)
                .build();
        try {
            timestreamWriteClient.updateDatabase(request);
            System.out.println("Database [" + DATABASE_NAME + "] updated
successfully with kmsKeyId " + kmsKeyId);
        } catch (ResourceNotFoundException e) {
            System.out.println("Database [" + DATABASE_NAME + "] does not exist.
Skipping UpdateDatabase");
        } catch (Exception e) {
            System.out.println("UpdateDatabase failed: " + e);
        }
    }
```

Go

```
// Update Database.
        updateDatabaseInput := &timestreamwrite.UpdateDatabaseInput {
```

```
            DatabaseName: aws.String(*databaseName),
            KmsKeyId: aws.String(*kmsKeyId),
        }

        updateDatabaseOutput, err := writeSvc.UpdateDatabase(updateDatabaseInput)

        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update database is successful, below is the output:")
            fmt.Println(updateDatabaseOutput)
        }
```

Python

```
   def update_database(self, kms_id):
        print("Updating database")
        try:
            result =
self.client.update_database(DatabaseName=Constant.DATABASE_NAME, KmsKeyId=kms_id)
            print("Database [%s] was updated to use kms [%s] successfully" %
(Constant.DATABASE_NAME,

result['Database']['KmsKeyId']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Database doesn't exist")
        except Exception as err:
            print("Update database failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 UpdateDatabaseCommand和UpdateDatabase。

```
import { TimestreamWriteClient, UpdateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
let updatedKmsKeyId = "<updatedKmsKeyId>";

const params = {
```

```
    DatabaseName: "testDbFromNode",
    KmsKeyId: updatedKmsKeyId
};

const command = new UpdateDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} updated kmsKeyId to
 ${updatedKmsKeyId}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Database doesn't exist.");
    } else {
        console.log("Update database failed.", error);
    }
}
```

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function updateDatabase(updatedKmsKeyId) {

    if (updatedKmsKeyId === undefined) {
        console.log("Skipping UpdateDatabase; KmsKeyId was not given");
        return;
    }
    console.log("Updating Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        KmsKeyId: updatedKmsKeyId
    }

    const promise = writeClient.updateDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} updated kmsKeyId to
 ${updatedKmsKeyId}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Database doesn't exist.");
```

```
            } else {
                console.log("Update database failed.", err);
            }
        }
    );
}
```

.NET

```
        public async Task UpdateDatabase(String updatedKmsKeyId)
        {
            Console.WriteLine("Updating Database");

            try
            {
                var updateDatabaseRequest = new UpdateDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    KmsKeyId = updatedKmsKeyId
                };
                UpdateDatabaseResponse response = await
 writeClient.UpdateDatabaseAsync(updateDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} updated with
 KmsKeyId {updatedKmsKeyId}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Database does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Update database failed: " + e.ToString());
            }

        }

        private void PrintDatabases(List<Database> databases)
        {
            foreach (Database database in databases)
                Console.WriteLine($"Database:{database.DatabaseName}");
        }
```

删除数据库

您可以使用以下代码段来删除数据库。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
   public void deleteDatabase() {
       System.out.println("Deleting database");
       final DeleteDatabaseRequest deleteDatabaseRequest = new
DeleteDatabaseRequest();
       deleteDatabaseRequest.setDatabaseName(DATABASE_NAME);
       try {
           DeleteDatabaseResult result =
                   amazonTimestreamWrite.deleteDatabase(deleteDatabaseRequest);
           System.out.println("Delete database status: " +
result.getSdkHttpMetadata().getHttpStatusCode());
       } catch (final ResourceNotFoundException e) {
           System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
e);
           throw e;
       } catch (final Exception e) {
           System.out.println("Could not delete Database " + DATABASE_NAME + " = "
+ e);
           throw e;
       }
   }
```

Java v2

```
   public void deleteDatabase() {
       System.out.println("Deleting database");
       final DeleteDatabaseRequest deleteDatabaseRequest = new
DeleteDatabaseRequest();
       deleteDatabaseRequest.setDatabaseName(DATABASE_NAME);
       try {
```

```
            DeleteDatabaseResult result =
                    amazonTimestreamWrite.deleteDatabase(deleteDatabaseRequest);
            System.out.println("Delete database status: " +
 result.getSdkHttpMetadata().getHttpStatusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
 e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete Database " + DATABASE_NAME + " = "
 + e);
            throw e;
        }
    }
```

Go

```
deleteDatabaseInput := &timestreamwrite.DeleteDatabaseInput{
        DatabaseName:   aws.String(*databaseName),
    }

    _, err = writeSvc.DeleteDatabase(deleteDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Database deleted:", *databaseName)
    }
```

Python

```
    def delete_database(self):
        print("Deleting Database")
        try:
            result =
 self.client.delete_database(DatabaseName=Constant.DATABASE_NAME)
            print("Delete database status [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
        except self.client.exceptions.ResourceNotFoundException:
            print("database [%s] doesn't exist" % Constant.DATABASE_NAME)
        except Exception as err:
            print("Delete database failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 DeleteDatabaseCommand和DeleteDatabase。

```
import { TimestreamWriteClient, DeleteDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode"
};

const command = new DeleteDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Deleted database");
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log(`Database ${params.DatabaseName} doesn't exists.`);
    } else {
        console.log("Delete database failed.", error);
        throw error;
    }
}
```

以下代码段使用 for JavaScript V2 样式。 AWS SDK它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function deleteDatabase() {
    console.log("Deleting Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.deleteDatabase(params).promise();

    await promise.then(
        function (data) {
            console.log("Deleted database");
```

```
         },
        function(err) {
            if (err.code === 'ResourceNotFoundException') {
                console.log(`Database ${params.DatabaseName} doesn't exists.`);
            } else {
                console.log("Delete database failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DeleteDatabase()
        {
            Console.WriteLine("Deleting database");
            try
            {
                var deleteDatabaseRequest = new DeleteDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
                DeleteDatabaseResponse response = await
 writeClient.DeleteDatabaseAsync(deleteDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} delete
 request status:{response.HttpStatusCode}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine($"Database {Constants.DATABASE_NAME} does not
 exists");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception while deleting database:" +
 e.ToString());
            }
        }
```

列出数据库

您可以使用以下代码片段来列出您的数据库。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
    public void listDatabases() {
        System.out.println("Listing databases");
        ListDatabasesRequest request = new ListDatabasesRequest();
        ListDatabasesResult result = amazonTimestreamWrite.listDatabases(request);
        final List<Database> databases = result.getDatabases();
        printDatabases(databases);

        String nextToken = result.getNextToken();
        while (nextToken != null && !nextToken.isEmpty()) {
            request.setNextToken(nextToken);
            ListDatabasesResult nextResult =
amazonTimestreamWrite.listDatabases(request);
            final List<Database> nextDatabases = nextResult.getDatabases();
            printDatabases(nextDatabases);
            nextToken = nextResult.getNextToken();
        }
    }

    private void printDatabases(List<Database> databases) {
        for (Database db : databases) {
            System.out.println(db.getDatabaseName());
        }
    }
```

Java v2

```
    public void listDatabases() {
        System.out.println("Listing databases");
        ListDatabasesRequest request =
ListDatabasesRequest.builder().maxResults(2).build();
```

```
        ListDatabasesIterable listDatabasesIterable =
 timestreamWriteClient.listDatabasesPaginator(request);
        for(ListDatabasesResponse listDatabasesResponse : listDatabasesIterable) {
            final List<Database> databases = listDatabasesResponse.databases();
            databases.forEach(database ->
 System.out.println(database.databaseName()));
        }
    }
```

Go

```
// List databases.
    listDatabasesMaxResult := int64(15)

    listDatabasesInput := &timestreamwrite.ListDatabasesInput{
        MaxResults: &listDatabasesMaxResult,
    }

    listDatabasesOutput, err := writeSvc.ListDatabases(listDatabasesInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("List databases is successful, below is the output:")
        fmt.Println(listDatabasesOutput)
    }
```

Python

```
    def list_databases(self):
        print("Listing databases")
        try:
            result = self.client.list_databases(MaxResults=5)
            self._print_databases(result['Databases'])
            next_token = result.get('NextToken', None)
            while next_token:
                result = self.client.list_databases(NextToken=next_token,
 MaxResults=5)
                self._print_databases(result['Databases'])
                next_token = result.get('NextToken', None)
        except Exception as err:
            print("List databases failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 ListDatabasesCommand和ListDatabases。

```
import { TimestreamWriteClient, ListDatabasesCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    MaxResults: 15
};

const command = new ListDatabasesCommand(params);

getDatabasesList(null);

async function getDatabasesList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.Databases.forEach(function (database) {
            console.log(database.DatabaseName);
        });

        if (data.NextToken) {
            return getDatabasesList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing databases", error);
    }
}
```

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function listDatabases() {
```

```
    console.log("Listing databases:");
    const databases = await getDatabasesList(null);
    databases.forEach(function(database){
        console.log(database.DatabaseName);
    });
}

function getDatabasesList(nextToken, databases = []) {
    var params = {
        MaxResults: 15
    };

    if(nextToken) {
        params.NextToken = nextToken;
    }

    return writeClient.listDatabases(params).promise()
        .then(
            (data) => {
                databases.push.apply(databases, data.Databases);
                if (data.NextToken) {
                    return getDatabasesList(data.NextToken, databases);
                } else {
                    return databases;
                }
            },
            (err) => {
                console.log("Error while listing databases", err);
            });
}
```

.NET

```
        public async Task ListDatabases()
        {
            Console.WriteLine("Listing Databases");

            try
            {
                var listDatabasesRequest = new ListDatabasesRequest
                {
                    MaxResults = 5
                };
```

```
                ListDatabasesResponse response = await
writeClient.ListDatabasesAsync(listDatabasesRequest);
                PrintDatabases(response.Databases);
                var nextToken = response.NextToken;
                while (nextToken != null)
                {
                    listDatabasesRequest.NextToken = nextToken;
                    response = await
writeClient.ListDatabasesAsync(listDatabasesRequest);
                    PrintDatabases(response.Databases);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List database failed:" + e.ToString());
            }

        }
```

创建表

主题

- 内存存储写入
- 磁性存储器写入

内存存储写入

您可以使用以下代码片段来创建禁用磁存储写入功能的表，因此您只能将数据写入内存存储保留窗口。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
    public void createTable() {
```

```
        System.out.println("Creating table");
        CreateTableRequest createTableRequest = new CreateTableRequest();
        createTableRequest.setDatabaseName(DATABASE_NAME);
        createTableRequest.setTableName(TABLE_NAME);
        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
        createTableRequest.setRetentionProperties(retentionProperties);

        try {
            amazonTimestreamWrite.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Java v2

```
    public void createTable() {
        System.out.println("Creating table");

        final RetentionProperties retentionProperties =
RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS).build();
        final CreateTableRequest createTableRequest = CreateTableRequest.builder()

.databaseName(DATABASE_NAME).tableName(TABLE_NAME).retentionProperties(retentionProperties)

        try {
            timestreamWriteClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Go

```
// Create table.
```

```
    createTableInput := &timestreamwrite.CreateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    _, err = writeSvc.CreateTable(createTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Create table is successful")
    }
```

Python

```
    def create_table(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        try:
            self.client.create_table(DatabaseName=Constant.DATABASE_NAME,
TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties)
            print("Table [%s] successfully created." % Constant.TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                Constant.TABLE_NAME, Constant.DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 CreateTableCommand和CreateTable。

```
import { TimestreamWriteClient, CreateTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
```

```
const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode",
    RetentionProperties: {
        MemoryStoreRetentionPeriodInHours: 24,
        MagneticStoreRetentionPeriodInDays: 365
    }
};

const command = new CreateTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Table ${data.Table.TableName} created successfully`);
} catch (error) {
    if (error.code === 'ConflictException') {
        console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
    } else {
        console.log("Error creating table. ", error);
        throw error;
    }
}
```

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function createTable() {
    console.log("Creating Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        }
    };

    const promise = writeClient.createTable(params).promise();

    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} created successfully`);
```

```
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
            } else {
                console.log("Error creating table. ", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task CreateTable()
        {
            Console.WriteLine("Creating Table");

            try
            {
                var createTableRequest = new CreateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    }
                };
                CreateTableResponse response = await
 writeClient.CreateTableAsync(createTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Table already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create table failed:" + e.ToString());
            }
```

```
        }
```

磁性存储器写入

您可以使用以下代码片段创建启用磁性存储写入功能的表。通过磁存储写入，您可以将数据写入内存存储保留窗口和磁存储保留窗口。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
   public void createTable(String databaseName, String tableName) {
       System.out.println("Creating table");
       CreateTableRequest createTableRequest = new CreateTableRequest();
       createTableRequest.setDatabaseName(databaseName);
       createTableRequest.setTableName(tableName);
       final RetentionProperties retentionProperties = new RetentionProperties()
               .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
               .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
       createTableRequest.setRetentionProperties(retentionProperties);
       // Enable MagneticStoreWrite
       final MagneticStoreWriteProperties magneticStoreWriteProperties = new
MagneticStoreWriteProperties()
               .withEnableMagneticStoreWrites(true);

createTableRequest.setMagneticStoreWriteProperties(magneticStoreWriteProperties);
       try {
           amazonTimestreamWrite.createTable(createTableRequest);
           System.out.println("Table [" + tableName + "] successfully created.");
       } catch (ConflictException e) {
           System.out.println("Table [" + tableName + "] exists on database [" +
databaseName + "] . Skipping table creation");
           //We do not throw exception here, we use the existing table instead
       }
   }
```

Java v2

```
    public void createTable(String databaseName, String tableName) {
        System.out.println("Creating table");

        // Enable MagneticStoreWrite
        final MagneticStoreWriteProperties magneticStoreWriteProperties =
                MagneticStoreWriteProperties.builder()
                        .enableMagneticStoreWrites(true)
                        .build();

        CreateTableRequest createTableRequest =
                CreateTableRequest.builder()
                        .databaseName(databaseName)
                        .tableName(tableName)
                        .retentionProperties(RetentionProperties.builder()
                                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS)
                                .build())
                        .magneticStoreWriteProperties(magneticStoreWriteProperties)
                        .build();
        try {
            timestreamWriteClient.createTable(createTableRequest);
            System.out.println("Table [" + tableName + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + tableName + "] exists in database [" +
 databaseName + "] . Skipping table creation");
        }
    }
```

Go

```
// Create table.
    createTableInput := &timestreamwrite.CreateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    // Enable MagneticStoreWrite
        MagneticStoreWriteProperties: &timestreamwrite.MagneticStoreWriteProperties{
            EnableMagneticStoreWrites: aws.Bool(true),
             },
      }
    _, err = writeSvc.CreateTable(createTableInput)
```

Python

```
    def create_table(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        magnetic_store_write_properties = {
            'EnableMagneticStoreWrites': True
        }
        try:
            self.client.create_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties,

 MagneticStoreWriteProperties=magnetic_store_write_properties)
            print("Table [%s] successfully created." % Constant.TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                Constant.TABLE_NAME, Constant.DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

Node.js

```
async function createTable() {
    console.log("Creating Table");

    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        },
        MagneticStoreWriteProperties: {
            EnableMagneticStoreWrites: true
        }
    };

    const promise = writeClient.createTable(params).promise();
```

```
    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} created successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
            } else {
                console.log("Error creating table. ", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task CreateTable()
        {
            Console.WriteLine("Creating Table");

            try
            {
                var createTableRequest = new CreateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    },
                    // Enable MagneticStoreWrite
                    MagneticStoreWriteProperties = new MagneticStoreWriteProperties
                    {
                        EnableMagneticStoreWrites = true,
                    }
                };
                CreateTableResponse response = await
 writeClient.CreateTableAsync(createTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} created");
            }
```

```
            catch (ConflictException)
            {
                Console.WriteLine("Table already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create table failed:" + e.ToString());
            }

        }
```

描述表

您可以使用以下代码片段来获取有关表属性的信息。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
   public void describeTable() {
       System.out.println("Describing table");
       final DescribeTableRequest describeTableRequest = new
DescribeTableRequest();
       describeTableRequest.setDatabaseName(DATABASE_NAME);
       describeTableRequest.setTableName(TABLE_NAME);
       try {
           DescribeTableResult result =
amazonTimestreamWrite.describeTable(describeTableRequest);
           String tableId = result.getTable().getArn();
           System.out.println("Table " + TABLE_NAME + " has id " + tableId);
       } catch (final Exception e) {
           System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
           throw e;
       }
   }
```

Java v2

```
    public void describeTable() {
        System.out.println("Describing table");
        final DescribeTableRequest describeTableRequest =
 DescribeTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DescribeTableResponse response =
 timestreamWriteClient.describeTable(describeTableRequest);
            String tableId = response.table().arn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
        } catch (final Exception e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
    }
```

Go

```
// Describe table.
    describeTableInput := &timestreamwrite.DescribeTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    describeTableOutput, err := writeSvc.DescribeTable(describeTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Describe table is successful, below is the output:")
        fmt.Println(describeTableOutput)
    }
```

Python

```
    def describe_table(self):
        print("Describing table")
        try:
            result = self.client.describe_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME)
```

```
            print("Table [%s] has id [%s]" % (Constant.TABLE_NAME, result['Table']
['Arn']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Table doesn't exist")
        except Exception as err:
            print("Describe table failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 DescribeTableCommand和DescribeTable。

```
import { TimestreamWriteClient, DescribeTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode"
};

const command = new DescribeTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Table ${data.Table.TableName} has id ${data.Table.Arn}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Table or Database doesn't exist.");
    } else {
        console.log("Describe table failed.", error);
        throw error;
    }
}
```

以下代码段使用 for JavaScript V2 样式。 AWS SDK它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function describeTable() {
    console.log("Describing Table");
    const params = {
```

```
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME
    };

    const promise = writeClient.describeTable(params).promise();

    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} has id ${data.Table.Arn}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Table or Database doesn't exists.");
            } else {
                console.log("Describe table failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DescribeTable()
        {
            Console.WriteLine("Describing Table");

            try
            {
                var describeTableRequest = new DescribeTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME
                };
                DescribeTableResponse response = await
 writeClient.DescribeTableAsync(describeTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} has id:
{response.Table.Arn}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Table does not exist.");
            }
```

```
            catch (Exception e)
            {
                Console.WriteLine("Describe table failed:" + e.ToString());
            }

        }
```

更新表

您可以使用以下代码片段来更新表。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
    public void updateTable() {
        System.out.println("Updating table");
        UpdateTableRequest updateTableRequest = new UpdateTableRequest();
        updateTableRequest.setDatabaseName(DATABASE_NAME);
        updateTableRequest.setTableName(TABLE_NAME);

        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);

        updateTableRequest.setRetentionProperties(retentionProperties);

        amazonTimestreamWrite.updateTable(updateTableRequest);
        System.out.println("Table updated");
    }
```

Java v2

```
    public void updateTable() {
        System.out.println("Updating table");
```

```
        final RetentionProperties retentionProperties =
 RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS).build();
        final UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()

 .databaseName(DATABASE_NAME).tableName(TABLE_NAME).retentionProperties(retentionProperties)

        timestreamWriteClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
    }
```

Go

```
// Update table.
    magneticStoreRetentionPeriodInDays := int64(7 * 365)
    memoryStoreRetentionPeriodInHours := int64(24)

    updateTableInput := &timestreamwrite.UpdateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
        RetentionProperties: &timestreamwrite.RetentionProperties{
            MagneticStoreRetentionPeriodInDays: &magneticStoreRetentionPeriodInDays,
            MemoryStoreRetentionPeriodInHours:  &memoryStoreRetentionPeriodInHours,
        },
    }
    updateTableOutput, err := writeSvc.UpdateTable(updateTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Update table is successful, below is the output:")
        fmt.Println(updateTableOutput)
    }
```

Python

```
    def update_table(self):
        print("Updating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
```

```
        }
        try:
            self.client.update_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties)
            print("Table updated.")
        except Exception as err:
            print("Update table failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 UpdateTableCommand和UpdateTable。

```
import { TimestreamWriteClient, UpdateTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode",
    RetentionProperties: {
        MemoryStoreRetentionPeriodInHours: 24,
        MagneticStoreRetentionPeriodInDays: 180
    }
};

const command = new UpdateTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Table updated")
} catch (error) {
    console.log("Error updating table. ", error);
}
```

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function updateTable() {
    console.log("Updating Table");
```

```
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        }
    };

    const promise = writeClient.updateTable(params).promise();

    await promise.then(
        (data) => {
            console.log("Table updated")
        },
        (err) => {
            console.log("Error updating table. ", err);
            throw err;
        }
    );
}
```

.NET

```
        public async Task UpdateTable()
        {
            Console.WriteLine("Updating Table");

            try
            {
                var updateTableRequest = new UpdateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    }
                };
                UpdateTableResponse response = await
 writeClient.UpdateTableAsync(updateTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} updated");
```

```
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Table does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Update table failed:" + e.ToString());
            }

        }
```

删除表

您可以使用以下代码片段来删除表。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
   public void deleteTable() {
       System.out.println("Deleting table");
       final DeleteTableRequest deleteTableRequest = new DeleteTableRequest();
       deleteTableRequest.setDatabaseName(DATABASE_NAME);
       deleteTableRequest.setTableName(TABLE_NAME);
       try {
           DeleteTableResult result =
                   amazonTimestreamWrite.deleteTable(deleteTableRequest);
           System.out.println("Delete table status: " +
result.getSdkHttpMetadata().getHttpStatusCode());
       } catch (final ResourceNotFoundException e) {
           System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
           throw e;
       } catch (final Exception e) {
           System.out.println("Could not delete table " + TABLE_NAME + " = " + e);
           throw e;
       }
```

```
    }
```

Java v2

```
    public void deleteTable() {
        System.out.println("Deleting table");
        final DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DeleteTableResponse response =
                    timestreamWriteClient.deleteTable(deleteTableRequest);
            System.out.println("Delete table status: " +
 response.sdkHttpResponse().statusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete table " + TABLE_NAME + " = " + e);
            throw e;
        }
    }
```

Go

```
deleteTableInput := &timestreamwrite.DeleteTableInput{
        DatabaseName:   aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    _, err = writeSvc.DeleteTable(deleteTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Table deleted", *tableName)
    }
```

Python

```
    def delete_table(self):
        print("Deleting Table")
        try:
```

```
            result = self.client.delete_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME)
            print("Delete table status [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
        except self.client.exceptions.ResourceNotFoundException:
            print("Table [%s] doesn't exist" % Constant.TABLE_NAME)
        except Exception as err:
            print("Delete table failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 DeleteTableCommand和DeleteTable。

```
import { TimestreamWriteClient, DeleteTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode"
};

const command = new DeleteTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Deleted table");
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log(`Table ${params.TableName} or Database ${params.DatabaseName}
 doesn't exist.`);
    } else {
        console.log("Delete table failed.", error);
        throw error;
    }
}
```

以下代码段使用 for JavaScript V2 样式。 AWS SDK它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function deleteTable() {
    console.log("Deleting Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME
    };

    const promise = writeClient.deleteTable(params).promise();

    await promise.then(
        function (data) {
            console.log("Deleted table");
        },
        function(err) {
            if (err.code === 'ResourceNotFoundException') {
                console.log(`Table ${params.TableName} or Database
 ${params.DatabaseName} doesn't exists.`);
            } else {
                console.log("Delete table failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DeleteTable()
        {
            Console.WriteLine("Deleting table");
            try
            {
                var deleteTableRequest = new DeleteTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME
                };
                DeleteTableResponse response = await
 writeClient.DeleteTableAsync(deleteTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} delete request
 status: {response.HttpStatusCode}");
            }
            catch (ResourceNotFoundException)
```

```
            {
                Console.WriteLine($"Table {Constants.TABLE_NAME} does not exists");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception while deleting table:" + e.ToString());
            }
        }
```

列出表

您可以使用以下代码片段来列出表。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
    public void listTables() {
        System.out.println("Listing tables");
        ListTablesRequest request = new ListTablesRequest();
        request.setDatabaseName(DATABASE_NAME);
        ListTablesResult result = amazonTimestreamWrite.listTables(request);
        printTables(result.getTables());

        String nextToken = result.getNextToken();
        while (nextToken != null && !nextToken.isEmpty()) {
            request.setNextToken(nextToken);
            ListTablesResult nextResult = amazonTimestreamWrite.listTables(request);

            printTables(nextResult.getTables());
            nextToken = nextResult.getNextToken();
        }
    }

     private void printTables(List<Table> tables) {
        for (Table table : tables) {
            System.out.println(table.getTableName());
```

```
        }
    }
```

Java v2

```
    public void listTables() {
        System.out.println("Listing tables");
        ListTablesRequest request =
 ListTablesRequest.builder().databaseName(DATABASE_NAME).maxResults(2).build();
        ListTablesIterable listTablesIterable =
 timestreamWriteClient.listTablesPaginator(request);
        for(ListTablesResponse listTablesResponse : listTablesIterable) {
            final List<Table> tables = listTablesResponse.tables();
            tables.forEach(table -> System.out.println(table.tableName()));
        }
    }
```

Go

```
listTablesMaxResult := int64(15)

    listTablesInput := &timestreamwrite.ListTablesInput{
        DatabaseName: aws.String(*databaseName),
        MaxResults:   &listTablesMaxResult,
    }
    listTablesOutput, err := writeSvc.ListTables(listTablesInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("List tables is successful, below is the output:")
        fmt.Println(listTablesOutput)
    }
```

Python

```
    def list_tables(self):
        print("Listing tables")
        try:
            result = self.client.list_tables(DatabaseName=Constant.DATABASE_NAME,
 MaxResults=5)
```

```
            self.__print_tables(result['Tables'])
            next_token = result.get('NextToken', None)
            while next_token:
                result =
 self.client.list_tables(DatabaseName=Constant.DATABASE_NAME,
                                                    NextToken=next_token, MaxResults=5)
                self.__print_tables(result['Tables'])
                next_token = result.get('NextToken', None)
        except Exception as err:
            print("List tables failed:", err)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

另请参阅类 ListTablesCommand和ListTables。

```
import { TimestreamWriteClient, ListTablesCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    MaxResults: 15
};

const command = new ListTablesCommand(params);

getTablesList(null);

async function getTablesList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.Tables.forEach(function (table) {
            console.log(table.TableName);
        });

        if (data.NextToken) {
```

```
            return getTablesList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing tables", error);
    }
}
```

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function listTables() {
    console.log("Listing tables:");
    const tables = await getTablesList(null);
    tables.forEach(function(table){
        console.log(table.TableName);
    });
}

function getTablesList(nextToken, tables = []) {
    var params = {
        DatabaseName: constants.DATABASE_NAME,
        MaxResults: 15
    };

    if(nextToken) {
        params.NextToken = nextToken;
    }

    return writeClient.listTables(params).promise()
        .then(
            (data) => {
                tables.push.apply(tables, data.Tables);
                if (data.NextToken) {
                    return getTablesList(data.NextToken, tables);
                } else {
                    return tables;
                }
            },
            (err) => {
                console.log("Error while listing databases", err);
            });
}
```

.NET

```
        public async Task ListTables()
        {
            Console.WriteLine("Listing Tables");

            try
            {
                var listTablesRequest = new ListTablesRequest
                {
                    MaxResults = 5,
                    DatabaseName = Constants.DATABASE_NAME
                };
                ListTablesResponse response = await
writeClient.ListTablesAsync(listTablesRequest);
                PrintTables(response.Tables);
                string nextToken = response.NextToken;
                while (nextToken != null)
                {
                    listTablesRequest.NextToken = nextToken;
                    response = await writeClient.ListTablesAsync(listTablesRequest);
                    PrintTables(response.Tables);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List table failed:" + e.ToString());
            }

        }

        private void PrintTables(List<Table> tables)
        {
            foreach (Table table in tables)
                Console.WriteLine($"Table: {table.TableName}");
        }
```

写入数据（插入和向上移动）

主题

- 写入成批的记录
- 批量写入具有共同属性的记录
- 颠覆记录
- 多度量属性示例
- 处理写入失败

写入成批的记录

您可以使用以下代码片段将数据写入 Amazon Timestream 表。批量写入数据有助于优化写入成本。请参阅计算写入次数了解更多信息。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
  public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record cpuUtilization = new Record()
        .withDimensions(dimensions)
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5")
```

```
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));
    Record memoryUtilization = new Record()
        .withDimensions(dimensions)
        .withMeasureName("memory_utilization")
        .withMeasureValue("40")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ":
"
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
 public void writeRecords() {
   System.out.println("Writing records");
   // Specify repeated values for all records
   List<Record> records = new ArrayList<>();
   final long time = System.currentTimeMillis();
```

```
    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record cpuUtilization = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .measureName("cpu_utilization")
        .measureValue("13.5")
        .time(String.valueOf(time)).build();

    Record memoryUtilization = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .measureName("memory_utilization")
        .measureValue("40")
        .time(String.valueOf(time)).build();

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME).tableName(TABLE_NAME).records(records).build();

    try {
      WriteRecordsResponse writeRecordsResponse =
 timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
 writeRecordsResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
            + rejectedRecord.reason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
```

```
      System.out.println("Error: " + e);
    }
  }
```

Go

```
now := time.Now()
currentTimeInSeconds := now.Unix()
writeRecordsInput := &timestreamwrite.WriteRecordsInput{
  DatabaseName: aws.String(*databaseName),
  TableName:  aws.String(*tableName),
  Records: []*timestreamwrite.Record{
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
        &timestreamwrite.Dimension{
          Name:  aws.String("region"),
          Value: aws.String("us-east-1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("az"),
          Value: aws.String("az1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("hostname"),
          Value: aws.String("host1"),
        },
      },
      MeasureName:    aws.String("cpu_utilization"),
      MeasureValue:   aws.String("13.5"),
      MeasureValueType: aws.String("DOUBLE"),
      Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
    },
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
        &timestreamwrite.Dimension{
          Name:  aws.String("region"),
          Value: aws.String("us-east-1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("az"),
          Value: aws.String("az1"),
        },
```

```
        &timestreamwrite.Dimension{
          Name:  aws.String("hostname"),
          Value: aws.String("host1"),
        },
      },
      MeasureName:    aws.String("memory_utilization"),
      MeasureValue:   aws.String("40"),
      MeasureValueType: aws.String("DOUBLE"),
      Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
    },
  },
}

_, err = writeSvc.WriteRecords(writeRecordsInput)

if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
} else {
  fmt.Println("Write records is successful")
}
```

Python

```
  def write_records(self):
    print("Writing records")
    current_time = self._current_milli_time()

    dimensions = [
      {'Name': 'region', 'Value': 'us-east-1'},
      {'Name': 'az', 'Value': 'az1'},
      {'Name': 'hostname', 'Value': 'host1'}
    ]

    cpu_utilization = {
      'Dimensions': dimensions,
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5',
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }
```

```
    memory_utilization = {
      'Dimensions': dimensions,
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40',
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }

    records = [cpu_utilization, memory_utilization]

    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes={})
      print("WriteRecords Status: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _print_rejected_records_exceptions(err):
    print("RejectedRecords: ", err)
    for rr in err.response["RejectedRecords"]:
      print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
      if "ExistingVersion" in rr:
        print("Rejected record existing version: ", rr["ExistingVersion"])

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function writeRecords() {
  console.log("Writing records");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
```

```
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const cpuUtilization = {
    'Dimensions': dimensions,
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5',
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const memoryUtilization = {
    'Dimensions': dimensions,
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40',
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records
  };

  const request = writeClient.writeRecords(params);

  await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
JSON.parse(request.response.httpResponse.body.toString());
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
```

```
  );
}
```

.NET

```
   public async Task WriteRecords()
   {
     Console.WriteLine("Writing records");

     DateTimeOffset now = DateTimeOffset.UtcNow;
     string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

     List<Dimension> dimensions = new List<Dimension>{
       new Dimension { Name = "region", Value = "us-east-1" },
       new Dimension { Name = "az", Value = "az1" },
       new Dimension { Name = "hostname", Value = "host1" }
     };

     var cpuUtilization = new Record
     {
       Dimensions = dimensions,
       MeasureName = "cpu_utilization",
       MeasureValue = "13.6",
       MeasureValueType = MeasureValueType.DOUBLE,
       Time = currentTimeString
     };

     var memoryUtilization = new Record
     {
       Dimensions = dimensions,
       MeasureName = "memory_utilization",
       MeasureValue = "40",
       MeasureValueType = MeasureValueType.DOUBLE,
       Time = currentTimeString
     };


     List<Record> records = new List<Record> {
       cpuUtilization,
       memoryUtilization
     };

     try
```

```
    {
      var writeRecordsRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = records
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      Console.WriteLine("RejectedRecordsException:" + e.ToString());
      foreach (RejectedRecord rr in e.RejectedRecords) {
        Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
      }
      Console.WriteLine("Other records were written successfully. ");
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }
  }
```

批量写入具有共同属性的记录

如果您的时间序列数据具有在许多数据点中通用的度量和/或维度，则还可以使用以下的优化版本将数据插入 writeRecords API到 Timestream 中。 LiveAnalytics在批处理中使用常用属性可以进一步优化写入成本，如中所述。计算写入次数

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
public void writeRecordsWithCommonAttributes() {
  System.out.println("Writing records with extracting common attributes");
```

```
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    Record cpuUtilization = new Record()
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5");
    Record memoryUtilization = new Record()
        .withMeasureName("memory_utilization")
        .withMeasureValue("40");

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsRequest.setRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
 amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("writeRecordsWithCommonAttributes Status: " +
 writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
```

```
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ":
 "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
  public void writeRecordsWithCommonAttributes() {
    System.out.println("Writing records with extracting common attributes");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .time(String.valueOf(time)).build();

    Record cpuUtilization = Record.builder()
        .measureName("cpu_utilization")
        .measureValue("13.5").build();
    Record memoryUtilization = Record.builder()
        .measureName("memory_utilization")
        .measureValue("40").build();

    records.add(cpuUtilization);
    records.add(memoryUtilization);
```

```
   WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
       .databaseName(DATABASE_NAME)
       .tableName(TABLE_NAME)
       .commonAttributes(commonAttributes)
       .records(records).build();

   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("writeRecordsWithCommonAttributes Status: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("RejectedRecords: " + e);
     for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
       System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
           + rejectedRecord.reason());
     }
     System.out.println("Other records were written successfully. ");
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }
 }
```

Go

```
now = time.Now()
currentTimeInSeconds = now.Unix()
writeRecordsCommonAttributesInput := &timestreamwrite.WriteRecordsInput{
 DatabaseName: aws.String(*databaseName),
 TableName:  aws.String(*tableName),
 CommonAttributes: &timestreamwrite.Record{
  Dimensions: []*timestreamwrite.Dimension{
   &timestreamwrite.Dimension{
    Name:  aws.String("region"),
    Value: aws.String("us-east-1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("az"),
    Value: aws.String("az1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("hostname"),
```

```
    Value: aws.String("host1"),
   },
  },
  MeasureValueType: aws.String("DOUBLE"),
  Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
  TimeUnit:     aws.String("SECONDS"),
 },
 Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
   MeasureName:  aws.String("cpu_utilization"),
   MeasureValue: aws.String("13.5"),
  },
  &timestreamwrite.Record{
   MeasureName:  aws.String("memory_utilization"),
   MeasureValue: aws.String("40"),
  },
 },
}

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Ingest records is successful")
}
```

Python

```
  def write_records_with_common_attributes(self):
    print("Writing records extracting common attributes")
    current_time = self._current_milli_time()

    dimensions = [
      {'Name': 'region', 'Value': 'us-east-1'},
      {'Name': 'az', 'Value': 'az1'},
      {'Name': 'hostname', 'Value': 'host1'}
    ]

    common_attributes = {
      'Dimensions': dimensions,
      'MeasureValueType': 'DOUBLE',
```

```
      'Time': current_time
    }

    cpu_utilization = {
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5'
    }

    memory_utilization = {
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40'
    }

    records = [cpu_utilization, memory_utilization]

    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _print_rejected_records_exceptions(err):
    print("RejectedRecords: ", err)
    for rr in err.response["RejectedRecords"]:
      print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
      if "ExistingVersion" in rr:
        print("Rejected record existing version: ", rr["ExistingVersion"])

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function writeRecordsWithCommonAttributes() {
  console.log("Writing records with common attributes");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const commonAttributes = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const cpuUtilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5'
  };

  const memoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40'
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records,
    CommonAttributes: commonAttributes
  };

  const request = writeClient.writeRecords(params);

  await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
```

```
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
 JSON.parse(request.response.httpResponse.body.toString());
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
}
```

.NET

```
  public async Task WriteRecordsWithCommonAttributes()
  {
    Console.WriteLine("Writing records with common attributes");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

    List<Dimension> dimensions = new List<Dimension>{
      new Dimension { Name = "region", Value = "us-east-1" },
      new Dimension { Name = "az", Value = "az1" },
      new Dimension { Name = "hostname", Value = "host1" }
    };

    var commonAttributes = new Record
    {
      Dimensions = dimensions,
      MeasureValueType = MeasureValueType.DOUBLE,
      Time = currentTimeString
    };

    var cpuUtilization = new Record
    {
      MeasureName = "cpu_utilization",
      MeasureValue = "13.6"
    };

    var memoryUtilization = new Record
    {
      MeasureName = "memory_utilization",
      MeasureValue = "40"
    };
```

```
   List<Record> records = new List<Record>();
   records.Add(cpuUtilization);
   records.Add(memoryUtilization);

   try
   {
     var writeRecordsRequest = new WriteRecordsRequest
     {
       DatabaseName = Constants.DATABASE_NAME,
       TableName = Constants.TABLE_NAME,
       Records = records,
       CommonAttributes = commonAttributes
     };
     WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
     Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
   }
   catch (RejectedRecordsException e) {
     Console.WriteLine("RejectedRecordsException:" + e.ToString());
     foreach (RejectedRecord rr in e.RejectedRecords) {
       Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
     }
     Console.WriteLine("Other records were written successfully. ");
   }
   catch (Exception e)
   {
     Console.WriteLine("Write records failure:" + e.ToString());
   }
 }
```

颠覆记录

虽然 Amazon Timestream 中的默认写入遵循第一个写入者获胜的语义，即数据仅作为附加存储并拒绝重复记录，但有些应用程序要求能够使用最后一个写入者获胜语义将数据写入 Amazon Timestream，即版本最高的记录存储在系统中。还有一些应用程序需要能够更新现有记录。为了解决这些情况，Amazon Timestream 提供了更新数据的功能。Upsert 是一种在记录不存在时将记录插入系统的操作，或者在记录存在时更新记录的操作。

您可以通过在发送WriteRecords请求时包含记录Version中的定义来更新记录。亚马逊 Timestream 将存储记录最高的记录。Version下面的代码示例显示了如何更新插入数据：

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
  public void writeRecordsWithUpsert() {
    System.out.println("Writing records with upsert");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time))
        .withVersion(version);

    Record cpuUtilization = new Record()
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5");
    Record memoryUtilization = new Record()
        .withMeasureName("memory_utilization")
```

```
        .withMeasureValue("40");

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsRequest.setRecords(records);

    // write records for first time
    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status for first time: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status for retry: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // upsert with lower version, this would fail because a higher version is
required to update the measure value.
    version -= 1;
    commonAttributes.setVersion(version);

    cpuUtilization.setMeasureValue("14.5");
    memoryUtilization.setMeasureValue("50");
```

```
   List<Record> upsertedRecords = new ArrayList<>();
   upsertedRecords.add(cpuUtilization);
   upsertedRecords.add(memoryUtilization);

   WriteRecordsRequest writeRecordsUpsertRequest = new WriteRecordsRequest()
       .withDatabaseName(DATABASE_NAME)
       .withTableName(TABLE_NAME)
       .withCommonAttributes(commonAttributes);
   writeRecordsUpsertRequest.setRecords(upsertedRecords);

   try {
     WriteRecordsResult writeRecordsUpsertResult =
amazonTimestreamWrite.writeRecords(writeRecordsUpsertRequest);
     System.out.println("WriteRecords Status for upsert with lower version: " +
writeRecordsUpsertResult.getSdkHttpMetadata().getHttpStatusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("WriteRecords Status for upsert with lower version: ");
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // upsert with higher version as new data in generated
   version = System.currentTimeMillis();
   commonAttributes.setVersion(version);

   writeRecordsUpsertRequest = new WriteRecordsRequest()
       .withDatabaseName(DATABASE_NAME)
       .withTableName(TABLE_NAME)
       .withCommonAttributes(commonAttributes);
   writeRecordsUpsertRequest.setRecords(upsertedRecords);

   try {
     WriteRecordsResult writeRecordsUpsertResult =
amazonTimestreamWrite.writeRecords(writeRecordsUpsertRequest);
     System.out.println("WriteRecords Status for upsert with higher version: " +
writeRecordsUpsertResult.getSdkHttpMetadata().getHttpStatusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }
 }
```

Java v2

```
  public void writeRecordsWithUpsert() {
    System.out.println("Writing records with upsert");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .time(String.valueOf(time))
        .version(version)
        .build();

    Record cpuUtilization = Record.builder()
        .measureName("cpu_utilization")
        .measureValue("13.5").build();
    Record memoryUtilization = Record.builder()
        .measureName("memory_utilization")
        .measureValue("40").build();

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();
```

```
   // write records for first time
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for first time: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for retry: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // upsert with lower version, this would fail because a higher version is
required to update the measure value.
   version -= 1;
   commonAttributes = Record.builder()
       .dimensions(dimensions)
       .measureValueType(MeasureValueType.DOUBLE)
       .time(String.valueOf(time))
       .version(version)
       .build();

   cpuUtilization = Record.builder()
       .measureName("cpu_utilization")
       .measureValue("14.5").build();
   memoryUtilization = Record.builder()
       .measureName("memory_utilization")
       .measureValue("50").build();

   List<Record> upsertedRecords = new ArrayList<>();
```

```
   upsertedRecords.add(cpuUtilization);
   upsertedRecords.add(memoryUtilization);

   WriteRecordsRequest writeRecordsUpsertRequest = WriteRecordsRequest.builder()
       .databaseName(DATABASE_NAME)
       .tableName(TABLE_NAME)
       .commonAttributes(commonAttributes)
       .records(upsertedRecords).build();

   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsUpsertRequest);
     System.out.println("WriteRecords Status for upsert with lower version: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("WriteRecords Status for upsert with lower version: ");
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // upsert with higher version as new data in generated
   version = System.currentTimeMillis();
   commonAttributes = Record.builder()
       .dimensions(dimensions)
       .measureValueType(MeasureValueType.DOUBLE)
       .time(String.valueOf(time))
       .version(version)
       .build();

   writeRecordsUpsertRequest = WriteRecordsRequest.builder()
       .databaseName(DATABASE_NAME)
       .tableName(TABLE_NAME)
       .commonAttributes(commonAttributes)
       .records(upsertedRecords).build();

   try {
     WriteRecordsResponse writeRecordsUpsertResponse =
timestreamWriteClient.writeRecords(writeRecordsUpsertRequest);
     System.out.println("WriteRecords Status for upsert with higher version: " +
writeRecordsUpsertResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
```

```
      System.out.println("Error: " + e);
    }
  }
```

Go

```
// Below code will ingest and upsert cpu_utilization and memory_utilization metric
 for a host on
// region=us-east-1, az=az1, and hostname=host1
fmt.Println("Ingesting records and set version as currentTimeInMills, hit enter to
 continue")
reader.ReadString('\n')

// Get current time in seconds.
now = time.Now()
currentTimeInSeconds = now.Unix()
// To achieve upsert (last writer wins) semantic, one example is to use current time
 as the version if you are writing directly from the data source
version := time.Now().Round(time.Millisecond).UnixNano() / 1e6   // set version as
 currentTimeInMills

writeRecordsCommonAttributesUpsertInput := &timestreamwrite.WriteRecordsInput{
 DatabaseName: aws.String(*databaseName),
 TableName:  aws.String(*tableName),
 CommonAttributes: &timestreamwrite.Record{
  Dimensions: []*timestreamwrite.Dimension{
   &timestreamwrite.Dimension{
    Name:  aws.String("region"),
    Value: aws.String("us-east-1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("az"),
    Value: aws.String("az1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("hostname"),
    Value: aws.String("host1"),
   },
  },
  MeasureValueType: aws.String("DOUBLE"),
  Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
  TimeUnit:  aws.String("SECONDS"),
  Version:      &version,
```

```
 },
 Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
   MeasureName:  aws.String("cpu_utilization"),
   MeasureValue: aws.String("13.5"),
  },
  &timestreamwrite.Record{
   MeasureName:  aws.String("memory_utilization"),
   MeasureValue: aws.String("40"),
  },
 },
}

// write records for first time
_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Frist-time write records is successful")
}

fmt.Println("Retry same writeRecordsRequest with same records and versions. Because
 writeRecords API is idempotent, this will success. hit enter to continue")
reader.ReadString('\n')

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Retry write records for same request is successful")
}

fmt.Println("Upsert with lower version, this would fail because a higher version is
 required to update the measure value. hit enter to continue")
reader.ReadString('\n')
version -= 1
writeRecordsCommonAttributesUpsertInput.CommonAttributes.Version = &version

updated_cpu_utilization := &timestreamwrite.Record{
 MeasureName:    aws.String("cpu_utilization"),
```

```
 MeasureValue:   aws.String("14.5"),
}
updated_memory_utilization := &timestreamwrite.Record{
 MeasureName:    aws.String("memory_utilization"),
 MeasureValue:   aws.String("50"),
}


writeRecordsCommonAttributesUpsertInput.Records = []*timestreamwrite.Record{
 updated_cpu_utilization,
 updated_memory_utilization,
}

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Write records with lower version is successful")
}

fmt.Println("Upsert with higher version as new data in generated, this would
 success. hit enter to continue")
reader.ReadString('\n')

version = time.Now().Round(time.Millisecond).UnixNano() / 1e6  // set version as
 currentTimeInMills
writeRecordsCommonAttributesUpsertInput.CommonAttributes.Version = &version

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Write records with higher version is successful")
}
```

Python

```
  def write_records_with_upsert(self):
    print("Writing records with upsert")
```

```
    current_time = self._current_milli_time()
    # To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    version = int(self._current_milli_time())

    dimensions = [
          {'Name': 'region', 'Value': 'us-east-1'},
          {'Name': 'az', 'Value': 'az1'},
          {'Name': 'hostname', 'Value': 'host1'}
        ]

    common_attributes = {
      'Dimensions': dimensions,
      'MeasureValueType': 'DOUBLE',
      'Time': current_time,
      'Version': version
    }

    cpu_utilization = {
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5'
    }

    memory_utilization = {
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40'
    }

    records = [cpu_utilization, memory_utilization]

    # write records for first time
    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                          Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status for first time: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

    # Successfully retry same writeRecordsRequest with same records and versions,
 because writeRecords API is idempotent.
```

```
    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                          Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status for retry: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

    # upsert with lower version, this would fail because a higher version is
 required to update the measure value.
    version -= 1
    common_attributes["Version"] = version

    cpu_utilization["MeasureValue"] = '14.5'
    memory_utilization["MeasureValue"] = '50'

    upsertedRecords = [cpu_utilization, memory_utilization]

    try:
      upsertedResult =
 self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                            Records=upsertedRecords,
 CommonAttributes=common_attributes)
      print("WriteRecords Status for upsert with lower version: [%s]" %
 upsertedResult['ResponseMetadata']['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)


    # upsert with higher version as new data is generated
    version = int(self._current_milli_time())
    common_attributes["Version"] = version

    try:
      upsertedResult =
 self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
```

```
                              Records=upsertedRecords,
 CommonAttributes=common_attributes)
      print("WriteRecords Upsert Status: [%s]" % upsertedResult['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function writeRecordsWithUpsert() {
  console.log("Writing records with upsert");
  const currentTime = Date.now().toString(); // Unix time in milliseconds
  // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
  let version = Date.now();

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const commonAttributes = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString(),
    'Version': version
  };

  const cpuUtilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5'
  };
```

```
  const memoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40'
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records,
    CommonAttributes: commonAttributes
  };

  const request = writeClient.writeRecords(params);

  // write records for first time
  await request.promise().then(
    (data) => {
      console.log("Write records successful for first time.");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(request);
      }
    }
  );

  // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
  await request.promise().then(
    (data) => {
      console.log("Write records successful for retry.");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(request);
      }
    }
  );
```

```
 // upsert with lower version, this would fail because a higher version is required
to update the measure value.
 version--;

 const commonAttributesWithLowerVersion = {
   'Dimensions': dimensions,
   'MeasureValueType': 'DOUBLE',
   'Time': currentTime.toString(),
   'Version': version
 };

 const updatedCpuUtilization = {
   'MeasureName': 'cpu_utilization',
   'MeasureValue': '14.5'
 };

 const updatedMemoryUtilization = {
   'MeasureName': 'memory_utilization',
   'MeasureValue': '50'
 };

 const upsertedRecords = [updatedCpuUtilization, updatedMemoryUtilization];

 const upsertedParamsWithLowerVersion = {
   DatabaseName: constants.DATABASE_NAME,
   TableName: constants.TABLE_NAME,
   Records: upsertedRecords,
   CommonAttributes: commonAttributesWithLowerVersion
 };

 const upsertRequestWithLowerVersion =
writeClient.writeRecords(upsertedParamsWithLowerVersion);

 await upsertRequestWithLowerVersion.promise().then(
   (data) => {
     console.log("Write records for upsert with lower version successful");
   },
   (err) => {
     console.log("Error writing records:", err);
     if (err.code === 'RejectedRecordsException') {
       printRejectedRecordsException(upsertRequestWithLowerVersion);
     }
   }
 );
```

```
  // upsert with higher version as new data in generated
  version = Date.now();

  const commonAttributesWithHigherVersion = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString(),
    'Version': version
  };

  const upsertedParamsWithHigherVerion = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: upsertedRecords,
    CommonAttributes: commonAttributesWithHigherVersion
  };

  const upsertRequestWithHigherVersion =
 writeClient.writeRecords(upsertedParamsWithHigherVerion);

  await upsertRequestWithHigherVersion.promise().then(
    (data) => {
      console.log("Write records upsert successful with higher version");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(upsertedParamsWithHigherVerion);
      }
    }
  );

}
```

.NET

```
  public async Task WriteRecordsWithUpsert()
  {
    Console.WriteLine("Writing records with upsert");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();
```

```
   // To achieve upsert (last writer wins) semantic, one example is to use current
time as the version if you are writing directly from the data source
   long version = now.ToUnixTimeMilliseconds();

   List<Dimension> dimensions = new List<Dimension>{
     new Dimension { Name = "region", Value = "us-east-1" },
     new Dimension { Name = "az", Value = "az1" },
     new Dimension { Name = "hostname", Value = "host1" }
   };

   var commonAttributes = new Record
   {
     Dimensions = dimensions,
     MeasureValueType = MeasureValueType.DOUBLE,
     Time = currentTimeString,
     Version = version
   };

   var cpuUtilization = new Record
   {
     MeasureName = "cpu_utilization",
     MeasureValue = "13.6"
   };

   var memoryUtilization = new Record
   {
     MeasureName = "memory_utilization",
     MeasureValue = "40"
   };


   List<Record> records = new List<Record>();
   records.Add(cpuUtilization);
   records.Add(memoryUtilization);

   // write records for first time
   try
   {
     var writeRecordsRequest = new WriteRecordsRequest
     {
       DatabaseName = Constants.DATABASE_NAME,
       TableName = Constants.TABLE_NAME,
       Records = records,
       CommonAttributes = commonAttributes
```

```
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"WriteRecords Status for first time:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }

    // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
    try
    {
      var writeRecordsRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = records,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"WriteRecords Status for retry:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }

    // upsert with lower version, this would fail because a higher version is
required to update the measure value.
    version--;
    Type recordType = typeof(Record);
    recordType.GetProperty("Version").SetValue(commonAttributes, version);
    recordType.GetProperty("MeasureValue").SetValue(cpuUtilization, "14.6");
```

```
    recordType.GetProperty("MeasureValue").SetValue(memoryUtilization, "50");

    List<Record> upsertedRecords = new List<Record> {
      cpuUtilization,
      memoryUtilization
    };

    try
    {
      var writeRecordsUpsertRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = upsertedRecords,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse upsertResponse = await
writeClient.WriteRecordsAsync(writeRecordsUpsertRequest);
      Console.WriteLine($"WriteRecords Status for upsert with lower version:
{upsertResponse.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }

    // upsert with higher version as new data in generated
    now = DateTimeOffset.UtcNow;
    version = now.ToUnixTimeMilliseconds();
    recordType.GetProperty("Version").SetValue(commonAttributes, version);

    try
    {
      var writeRecordsUpsertRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = upsertedRecords,
        CommonAttributes = commonAttributes
      };
```

```
     WriteRecordsResponse upsertResponse = await
writeClient.WriteRecordsAsync(writeRecordsUpsertRequest);
     Console.WriteLine($"WriteRecords Status for upsert with higher version:
{upsertResponse.HttpStatusCode.ToString()}");
   }
   catch (RejectedRecordsException e) {
     PrintRejectedRecordsException(e);
   }
   catch (Exception e)
   {
     Console.WriteLine("Write records failure:" + e.ToString());
   }
 }
```

多度量属性示例

此示例说明如何编写多度量属性。当您正在跟踪的设备或应用程序在同一时间戳发出多个指标或事件时，多指标@@ 属性非常有用。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
package com.amazonaws.services.timestream;

import static com.amazonaws.services.timestream.Main.DATABASE_NAME;
import static com.amazonaws.services.timestream.Main.REGION;
import static com.amazonaws.services.timestream.Main.TABLE_NAME;

import java.util.ArrayList;
import java.util.List;

import com.amazonaws.services.timestreamwrite.AmazonTimestreamWrite;
import com.amazonaws.services.timestreamwrite.model.Dimension;
import com.amazonaws.services.timestreamwrite.model.MeasureValue;
import com.amazonaws.services.timestreamwrite.model.MeasureValueType;
import com.amazonaws.services.timestreamwrite.model.Record;
```

```
import com.amazonaws.services.timestreamwrite.model.RejectedRecordsException;
import com.amazonaws.services.timestreamwrite.model.WriteRecordsRequest;
import com.amazonaws.services.timestreamwrite.model.WriteRecordsResult;


public class multimeasureAttributeExample {
  AmazonTimestreamWrite timestreamWriteClient;

  public multimeasureAttributeExample(AmazonTimestreamWrite client) {
    this.timestreamWriteClient = client;
  }

  public void writeRecordsMultiMeasureValueSingleRecord() {
    System.out.println("Writing records with multi value attributes");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue(REGION);
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withTime(String.valueOf(time))
        .withVersion(version);

    MeasureValue cpuUtilization = new MeasureValue()
        .withName("cpu_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("13.5");
    MeasureValue memoryUtilization = new MeasureValue()
        .withName("memory_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("40");
    Record computationalResources = new Record()
        .withMeasureName("cpu_memory")
```

```
        .withMeasureValues(cpuUtilization, memoryUtilization)
        .withMeasureValueType(MeasureValueType.MULTI);

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes)
        .withRecords(records);

    // write records for first time
    try {
      WriteRecordsResult writeRecordResult =
timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordResult
              .getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  public void writeRecordsMultiMeasureValueMultipleRecords() {
    System.out.println(
        "Writing records with multi value attributes mixture type");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue(REGION);
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
```

```
        .withDimensions(dimensions)
        .withTime(String.valueOf(time))
        .withVersion(version);

    MeasureValue cpuUtilization = new MeasureValue()
        .withName("cpu_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("13");
    MeasureValue memoryUtilization =new MeasureValue()
        .withName("memory_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("40");
    MeasureValue activeCores = new MeasureValue()
        .withName("active_cores")
        .withType(MeasureValueType.BIGINT)
        .withValue("4");


    Record computationalResources = new Record()
        .withMeasureName("computational_utilization")
        .withMeasureValues(cpuUtilization, memoryUtilization, activeCores)
        .withMeasureValueType(MeasureValueType.MULTI);

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes)
        .withRecords(records);

    // write records for first time
    try {
      WriteRecordsResult writeRecordResult =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println(
         "WriteRecords Status for multi value attributes: " + writeRecordResult
             .getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

```
  private void printRejectedRecordsException(RejectedRecordsException e) {
    System.out.println("RejectedRecords: " + e);
    e.getRejectedRecords().forEach(System.out::println);
  }
}
```

Java v2

```
package com.amazonaws.services.timestream;

import java.util.ArrayList;
import java.util.List;

import software.amazon.awssdk.services.timestreamwrite.TimestreamWriteClient;
import software.amazon.awssdk.services.timestreamwrite.model.Dimension;
import software.amazon.awssdk.services.timestreamwrite.model.MeasureValue;
import software.amazon.awssdk.services.timestreamwrite.model.MeasureValueType;
import software.amazon.awssdk.services.timestreamwrite.model.Record;
import
 software.amazon.awssdk.services.timestreamwrite.model.RejectedRecordsException;
import software.amazon.awssdk.services.timestreamwrite.model.WriteRecordsRequest;
import software.amazon.awssdk.services.timestreamwrite.model.WriteRecordsResponse;

import static com.amazonaws.services.timestream.Main.DATABASE_NAME;
import static com.amazonaws.services.timestream.Main.TABLE_NAME;


public class multimeasureAttributeExample {

  TimestreamWriteClient timestreamWriteClient;

  public multimeasureAttributeExample(TimestreamWriteClient client) {
    this.timestreamWriteClient = client;
  }

  public void writeRecordsMultiMeasureValueSingleRecord() {
    System.out.println("Writing records with multi value attributes");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();
```

```
    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region =
        Dimension.builder().name("region").value("us-east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
        Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .time(String.valueOf(time))
        .version(version)
        .build();

    MeasureValue cpuUtilization = MeasureValue.builder()
        .name("cpu_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("13.5").build();
    MeasureValue memoryUtilization = MeasureValue.builder()
        .name("memory_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("40").build();
    Record computationalResources = Record
        .builder()
        .measureName("cpu_memory")
        .measureValues(cpuUtilization, memoryUtilization)
        .measureValueType(MeasureValueType.MULTI)
        .build();

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();

    // write records for first time
    try {
      WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
```

```
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordsResponse
              .sdkHttpResponse()
              .statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  public void writeRecordsMultiMeasureValueMultipleRecords() {
    System.out.println(
        "Writing records with multi value attributes mixture type");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region =
        Dimension.builder().name("region").value("us-east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
        Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .time(String.valueOf(time))
        .version(version)
        .build();

    MeasureValue cpuUtilization = MeasureValue.builder()
        .name("cpu_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("13.5").build();
    MeasureValue memoryUtilization = MeasureValue.builder()
        .name("memory_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("40").build();
```

```
    MeasureValue activeCores = MeasureValue.builder()
        .name("active_cores")
        .type(MeasureValueType.BIGINT)
        .value("4").build();


    Record computationalResources = Record
        .builder()
        .measureName("computational_utilization")
        .measureValues(cpuUtilization, memoryUtilization, activeCores)
        .measureValueType(MeasureValueType.MULTI)
        .build();

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();

    // write records for first time
    try {
      WriteRecordsResponse writeRecordsResponse =
 timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordsResponse
              .sdkHttpResponse()
              .statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  private void printRejectedRecordsException(RejectedRecordsException e) {
    System.out.println("RejectedRecords: " + e);
    e.rejectedRecords().forEach(System.out::println);
  }
}
```

Go

```
now := time.Now()
currentTimeInSeconds := now.Unix()
writeRecordsInput := &timestreamwrite.WriteRecordsInput{
  DatabaseName: aws.String(*databaseName),
  TableName:  aws.String(*tableName),
  Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
    Dimensions: []*timestreamwrite.Dimension{
    &timestreamwrite.Dimension{
      Name:  aws.String("region"),
      Value: aws.String("us-east-1"),
    },
    &timestreamwrite.Dimension{
      Name:  aws.String("az"),
      Value: aws.String("az1"),
    },
    &timestreamwrite.Dimension{
      Name:  aws.String("hostname"),
      Value: aws.String("host1"),
    },
    },
    MeasureName:  aws.String("metrics"),
    MeasureValueType: aws.String("MULTI"),
    Time:     aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
    TimeUnit:  aws.String("SECONDS"),
    MeasureValues: []*timestreamwrite.MeasureValue{
    &timestreamwrite.MeasureValue{
      Name:  aws.String("cpu_utilization"),
      Value: aws.String("13.5"),
      Type:  aws.String("DOUBLE"),
    },
    &timestreamwrite.MeasureValue{
      Name:  aws.String("memory_utilization"),
      Value: aws.String("40"),
      Type:  aws.String("DOUBLE"),
    },
    },
  },
  },
}

_, err = writeSvc.WriteRecords(writeRecordsInput)
```

```
  if err != nil {
    fmt.Println("Error:")
    fmt.Println(err)
  } else {
    fmt.Println("Write records is successful")
  }
```

Python

```
import time
import boto3
import psutil
import os

from botocore.config import Config

DATABASE_NAME = os.environ['DATABASE_NAME']
TABLE_NAME = os.environ['TABLE_NAME']

COUNTRY = "UK"
CITY = "London"
HOSTNAME = "MyHostname" # You can make it dynamic using socket.gethostname()

INTERVAL = 1 # Seconds

def prepare_common_attributes():
  common_attributes = {
    'Dimensions': [
      {'Name': 'country', 'Value': COUNTRY},
      {'Name': 'city', 'Value': CITY},
      {'Name': 'hostname', 'Value': HOSTNAME}
    ],
    'MeasureName': 'utilization',
    'MeasureValueType': 'MULTI'
  }
  return common_attributes


def prepare_record(current_time):
  record = {
    'Time': str(current_time),
    'MeasureValues': []
```

```
  }
  return record


def prepare_measure(measure_name, measure_value):
  measure = {
    'Name': measure_name,
    'Value': str(measure_value),
    'Type': 'DOUBLE'
  }
  return measure


def write_records(records, common_attributes):
  try:
    result = write_client.write_records(DatabaseName=DATABASE_NAME,
                                        TableName=TABLE_NAME,
                                        CommonAttributes=common_attributes,
                                        Records=records)
    status = result['ResponseMetadata']['HTTPStatusCode']
    print("Processed %d records. WriteRecords HTTPStatusCode: %s" %
        (len(records), status))
  except Exception as err:
    print("Error:", err)


if __name__ == '__main__':

  print("writing data to database {} table {}".format(
    DATABASE_NAME, TABLE_NAME))

  session = boto3.Session()
  write_client = session.client('timestream-write', config=Config(
    read_timeout=20, max_pool_connections=5000, retries={'max_attempts': 10}))
  query_client = session.client('timestream-query') # Not used

  common_attributes = prepare_common_attributes()

  records = []

  while True:

    current_time = int(time.time() * 1000)
    cpu_utilization = psutil.cpu_percent()
```

```
    memory_utilization = psutil.virtual_memory().percent
    swap_utilization = psutil.swap_memory().percent
    disk_utilization = psutil.disk_usage('/').percent

    record = prepare_record(current_time)
    record['MeasureValues'].append(prepare_measure('cpu', cpu_utilization))
    record['MeasureValues'].append(prepare_measure('memory', memory_utilization))
    record['MeasureValues'].append(prepare_measure('swap', swap_utilization))
    record['MeasureValues'].append(prepare_measure('disk', disk_utilization))

    records.append(record)

    print("records {} - cpu {} - memory {} - swap {} - disk {}".format(
      len(records), cpu_utilization, memory_utilization,
      swap_utilization, disk_utilization))

    if len(records) == 100:
      write_records(records, common_attributes)
      records = []

    time.sleep(INTERVAL)
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function writeRecords() {
  console.log("Writing records");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
  {'Name': 'region', 'Value': 'us-east-1'},
  {'Name': 'az', 'Value': 'az1'},
  {'Name': 'hostname', 'Value': 'host1'}
  ];

  const record = {
  'Dimensions': dimensions,
  'MeasureName': 'metrics',
  'MeasureValues': [
    {
      'Name': 'cpu_utilization',
```

```
        'Value': '40',
        'Type': 'DOUBLE',
      },
      {
        'Name': 'memory_utilization',
        'Value': '13.5',
        'Type': 'DOUBLE',
      },
      ],
      'MeasureValueType': 'MULTI',
      'Time': currentTime.toString()
    }

    const records = [record];

    const params = {
    DatabaseName: 'DatabaseName',
    TableName: 'TableName',
    Records: records
    };

    const response = await writeClient.writeRecords(params);

    console.log(response);
  }
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
  static class MultiMeasureValueConstants
  {
    public const string MultiMeasureValueSampleDb = "multiMeasureValueSampleDb";
    public const string MultiMeasureValueSampleTable =
 "multiMeasureValueSampleTable";
  }
```

```
public class MultiValueAttributesExample
{
  private readonly AmazonTimestreamWriteClient writeClient;

  public MultiValueAttributesExample(AmazonTimestreamWriteClient writeClient)
  {
    this.writeClient = writeClient;
  }

  public async Task WriteRecordsMultiMeasureValueSingleRecord()
  {
    Console.WriteLine("Writing records with multi value attributes");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

    List<Dimension> dimensions = new List<Dimension>{
      new Dimension { Name = "region", Value = "us-east-1" },
      new Dimension { Name = "az", Value = "az1" },
      new Dimension { Name = "hostname", Value = "host1" }
    };

    var commonAttributes = new Record
    {
      Dimensions = dimensions,
      Time = currentTimeString
    };

    var cpuUtilization = new MeasureValue
    {
      Name = "cpu_utilization",
      Value = "13.6",
      Type = "DOUBLE"
    };

    var memoryUtilization = new MeasureValue
    {
      Name = "memory_utilization",
      Value = "40",
      Type = "DOUBLE"
    };

    var computationalRecord = new Record
```

```
      {
        MeasureName = "cpu_memory",
        MeasureValues = new List<MeasureValue> {cpuUtilization, memoryUtilization},
        MeasureValueType = "MULTI"
      };


      List<Record> records = new List<Record>();
      records.Add(computationalRecord);

      try
      {
        var writeRecordsRequest = new WriteRecordsRequest
        {
          DatabaseName = MultiMeasureValueConstants.MultiMeasureValueSampleDb,
          TableName = MultiMeasureValueConstants.MultiMeasureValueSampleTable,
          Records = records,
          CommonAttributes = commonAttributes
        };
        WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
        Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
      }
      catch (Exception e)
      {
        Console.WriteLine("Write records failure:" + e.ToString());
      }
    }

    public async Task WriteRecordsMultiMeasureValueMultipleRecords()
    {
      Console.WriteLine("Writing records with multi value attributes mixture type");

      DateTimeOffset now = DateTimeOffset.UtcNow;
      string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

      List<Dimension> dimensions = new List<Dimension>{
        new Dimension { Name = "region", Value = "us-east-1" },
        new Dimension { Name = "az", Value = "az1" },
        new Dimension { Name = "hostname", Value = "host1" }
      };

      var commonAttributes = new Record
```

```
      {
        Dimensions = dimensions,
        Time = currentTimeString
      };

      var cpuUtilization = new MeasureValue
      {
        Name = "cpu_utilization",
        Value = "13.6",
        Type = "DOUBLE"
      };

      var memoryUtilization = new MeasureValue
      {
        Name = "memory_utilization",
        Value = "40",
        Type = "DOUBLE"
      };

      var activeCores = new MeasureValue
      {
        Name = "active_cores",
        Value = "4",
        Type = "BIGINT"
      };

      var computationalRecord = new Record
      {
        MeasureName = "computational_utilization",
        MeasureValues = new List<MeasureValue> {cpuUtilization, memoryUtilization,
activeCores},
        MeasureValueType = "MULTI"
      };

      var aliveRecord = new Record
      {
        MeasureName = "is_healthy",
        MeasureValue = "true",
        MeasureValueType = "BOOLEAN"
      };

      List<Record> records = new List<Record>();
      records.Add(computationalRecord);
      records.Add(aliveRecord);
```

```
      try
      {
        var writeRecordsRequest = new WriteRecordsRequest
        {
          DatabaseName = MultiMeasureValueConstants.MultiMeasureValueSampleDb,
          TableName = MultiMeasureValueConstants.MultiMeasureValueSampleTable,
          Records = records,
          CommonAttributes = commonAttributes
        };
        WriteRecordsResponse response = await
 writeClient.WriteRecordsAsync(writeRecordsRequest);
        Console.WriteLine($"Write records status code:
 {response.HttpStatusCode.ToString()}");
      }
      catch (Exception e)
      {
        Console.WriteLine("Write records failure:" + e.ToString());
      }
    }
  }
}
```

处理写入失败

由于以下一个或多个原因，在 Amazon Timestream 中写入操作可能会失败：

- 有些记录的时间戳超出了内存存储的保留期限。
- 有些记录包含超过时间流定义限制的维度和/或度量。
- 亚马逊 Timestream 已检测到重复的记录。如果存在多个具有相同维度、时间戳和度量名称的记录，但有：
  - 测量值不同。
  - 请求中不存在版本，或者新记录中版本的值等于或小于现有值。如果 Amazon Timestream 出于此原因拒绝数据，则中的`ExistingVersion`字段`RejectedRecords`将包含存储在亚马逊 Timestream 中的记录的当前版本。要强制更新，您可以重新发送请求，并将该记录的版本设置为大于。`ExistingVersion`

有关错误和被拒绝记录的更多信息，请参阅错误和RejectedRecord。

如果您的应用程序RejectedRecordsException在尝试将记录写入 Timestream 时收到，则可以解析被拒绝的记录以了解有关写入失败的更多信息，如下所示。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
 try {
   WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
   System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
 } catch (RejectedRecordsException e) {
   System.out.println("RejectedRecords: " + e);
   for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
     System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
         + rejectedRecord.getReason());
   }
   System.out.println("Other records were written successfully. ");
 } catch (Exception e) {
   System.out.println("Error: " + e);
 }
```

Java v2

```
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("writeRecordsWithCommonAttributes Status: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("RejectedRecords: " + e);
     for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
       System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
           + rejectedRecord.reason());
     }
     System.out.println("Other records were written successfully. ");
   } catch (Exception e) {
```

```
      System.out.println("Error: " + e);
    }
```

Go

```
_, err = writeSvc.WriteRecords(writeRecordsInput)

if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
} else {
  fmt.Println("Write records is successful")
}
```

Python

```
try:
  result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME, Records=records, CommonAttributes=common_attributes)
  print("WriteRecords Status: [%s]" % result['ResponseMetadata']['HTTPStatusCode'])
except self.client.exceptions.RejectedRecordsException as err:
  print("RejectedRecords: ", err)
  for rr in err.response["RejectedRecords"]:
    print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
  print("Other records were written successfully. ")
except Exception as err:
  print("Error:", err)
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
 JSON.parse(request.response.httpResponse.body.toString());
```

```
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
```

.NET

```
 try
 {
   var writeRecordsRequest = new WriteRecordsRequest
   {
     DatabaseName = Constants.DATABASE_NAME,
     TableName = Constants.TABLE_NAME,
     Records = records,
     CommonAttributes = commonAttributes
   };
   WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
   Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
 }
 catch (RejectedRecordsException e) {
   Console.WriteLine("RejectedRecordsException:" + e.ToString());
   foreach (RejectedRecord rr in e.RejectedRecords) {
     Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
   }
   Console.WriteLine("Other records were written successfully. ");
 }
 catch (Exception e)
 {
   Console.WriteLine("Write records failure:" + e.ToString());
 }
```

运行查询

主题

- 对结果进行分页
- 解析结果集
- 访问查询状态

对结果进行分页

运行查询时，Timestream 会以分页方式返回结果集，以优化应用程序的响应能力。下面的代码片段显示了如何对结果集进行分页。必须循环浏览所有结果集页面，直到遇到空值。分页令牌在 Timestream 发放 3 小时后过期。 LiveAnalytics

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
   private void runQuery(String queryString) {
       try {
           QueryRequest queryRequest = new QueryRequest();
           queryRequest.setQueryString(queryString);
           QueryResult queryResult = queryClient.query(queryRequest);
           while (true) {
               parseQueryResult(queryResult);
               if (queryResult.getNextToken() == null) {
                   break;
               }
               queryRequest.setNextToken(queryResult.getNextToken());
               queryResult = queryClient.query(queryRequest);
           }
       } catch (Exception e) {
           // Some queries might fail with 500 if the result of a sequence function
has more than 10000 entries
           e.printStackTrace();
       }
   }
```

Java v2

```
   private void runQuery(String queryString) {
       try {
           QueryRequest queryRequest =
QueryRequest.builder().queryString(queryString).build();
           final QueryIterable queryResponseIterator =
timestreamQueryClient.queryPaginator(queryRequest);
```

```
            for(QueryResponse queryResponse : queryResponseIterator) {
                parseQueryResult(queryResponse);
            }
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
 has more than 10000 entries
            e.printStackTrace();
        }
    }
```

Go

```
func runQuery(queryPtr *string, querySvc *timestreamquery.TimestreamQuery, f
 *os.File) {
    queryInput := &timestreamquery.QueryInput{
        QueryString: aws.String(*queryPtr),
    }
    fmt.Println("QueryInput:")
    fmt.Println(queryInput)
    // execute the query
    err := querySvc.QueryPages(queryInput,
        func(page *timestreamquery.QueryOutput, lastPage bool) bool {
            // process query response
            queryStatus := page.QueryStatus
            fmt.Println("Current query status:", queryStatus)
            // query response metadata
            // includes column names and types
            metadata := page.ColumnInfo
            // fmt.Println("Metadata:")
            fmt.Println(metadata)
            header := ""
            for i := 0; i < len(metadata); i++ {
                header += *metadata[i].Name
                if i != len(metadata)-1 {
                    header += ", "
                }
            }
            write(f, header)

            // query response data
            fmt.Println("Data:")
            // process rows
            rows := page.Rows
```

```
            for i := 0; i < len(rows); i++ {
                data := rows[i].Data
                value := processRowType(data, metadata)
                fmt.Println(value)
                write(f, value)
            }
            fmt.Println("Number of rows:", len(page.Rows))
            return true
        })
    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    }
}
```

Python

```
    def run_query(self, query_string):
        try:
            page_iterator = self.paginator.paginate(QueryString=query_string)
            for page in page_iterator:
                self._parse_query_result(page)
        except Exception as err:
            print("Exception while running query:", err)
```

Node.js

以下代码段使用 for JavaScript V2 样式。 AWS SDK它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function getAllRows(query, nextToken) {
    const params = {
        QueryString: query
    };

    if (nextToken) {
        params.NextToken = nextToken;
    }

    await queryClient.query(params).promise()
        .then(
            (response) => {
                parseQueryResult(response);
```

```
                if (response.NextToken) {
                    getAllRows(query, response.NextToken);
                }
            },
            (err) => {
                console.error("Error while querying:", err);
            });
}
```

.NET

```
        private async Task RunQueryAsync(string queryString)
        {
            try
            {
                QueryRequest queryRequest = new QueryRequest();
                queryRequest.QueryString = queryString;
                QueryResponse queryResponse = await
 queryClient.QueryAsync(queryRequest);
                while (true)
                {
                    ParseQueryResult(queryResponse);
                    if (queryResponse.NextToken == null)
                    {
                        break;
                    }
                    queryRequest.NextToken = queryResponse.NextToken;
                    queryResponse = await queryClient.QueryAsync(queryRequest);
                }
            } catch(Exception e)
            {
                // Some queries might fail with 500 if the result of a sequence
 function has more than 10000 entries
                Console.WriteLine(e.ToString());
            }
        }
```

解析结果集

您可以使用以下代码片段从结果集中提取数据。查询完成后，查询结果最长可在 24 小时内访问。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
   private static final DateTimeFormatter TIMESTAMP_FORMATTER =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.SSSSSSSSS");
   private static final DateTimeFormatter DATE_FORMATTER =
DateTimeFormatter.ofPattern("yyyy-MM-dd");
   private static final DateTimeFormatter TIME_FORMATTER =
DateTimeFormatter.ofPattern("HH:mm:ss.SSSSSSSSS");

   private static final long ONE_GB_IN_BYTES = 1073741824L;

   private void parseQueryResult(QueryResult response) {
       final QueryStatus currentStatusOfQuery = queryResult.getQueryStatus();

       System.out.println("Query progress so far: " +
currentStatusOfQuery.getProgressPercentage() + "%");

       double bytesScannedSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesScanned() / ONE_GB_IN_BYTES);
       System.out.println("Bytes scanned so far: " + bytesScannedSoFar + " GB");

       double bytesMeteredSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesMetered() / ONE_GB_IN_BYTES);
       System.out.println("Bytes metered so far: " + bytesMeteredSoFar + " GB");

       List<ColumnInfo> columnInfo = response.getColumnInfo();
       List<Row> rows = response.getRows();

       System.out.println("Metadata: " + columnInfo);
       System.out.println("Data: ");

       // iterate every row
       for (Row row : rows) {
           System.out.println(parseRow(columnInfo, row));
       }
   }
```

```
    private String parseRow(List<ColumnInfo> columnInfo, Row row) {
        List<Datum> data = row.getData();
        List<String> rowOutput = new ArrayList<>();
        // iterate every column per row
        for (int j = 0; j < data.size(); j++) {
            ColumnInfo info = columnInfo.get(j);
            Datum datum = data.get(j);
            rowOutput.add(parseDatum(info, datum));
        }
        return String.format("{%s}",
rowOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseDatum(ColumnInfo info, Datum datum) {
        if (datum.isNullValue() != null && datum.isNullValue()) {
            return info.getName() + "=" + "NULL";
        }
        Type columnType = info.getType();
        // If the column is of TimeSeries Type
        if (columnType.getTimeSeriesMeasureValueColumnInfo() != null) {
            return parseTimeSeries(info, datum);
        }
        // If the column is of Array Type
        else if (columnType.getArrayColumnInfo() != null) {
            List<Datum> arrayValues = datum.getArrayValue();
            return info.getName() + "=" +
parseArray(info.getType().getArrayColumnInfo(), arrayValues);
        }
        // If the column is of Row Type
        else if (columnType.getRowColumnInfo() != null) {
            List<ColumnInfo> rowColumnInfo = info.getType().getRowColumnInfo();
            Row rowValues = datum.getRowValue();
            return parseRow(rowColumnInfo, rowValues);
        }
        // If the column is of Scalar Type
        else {
            return parseScalarType(info, datum);
        }
    }

    private String parseTimeSeries(ColumnInfo info, Datum datum) {
        List<String> timeSeriesOutput = new ArrayList<>();
        for (TimeSeriesDataPoint dataPoint : datum.getTimeSeriesValue()) {
            timeSeriesOutput.add("{time=" + dataPoint.getTime() + ", value=" +
```

```
                    parseDatum(info.getType().getTimeSeriesMeasureValueColumnInfo(),
dataPoint.getValue()) + "}");
        }
        return String.format("[%s]",
timeSeriesOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseScalarType(ColumnInfo info, Datum datum) {
        switch (ScalarType.fromValue(info.getType().getScalarType())) {
            case VARCHAR:
                return parseColumnName(info) + datum.getScalarValue();
            case BIGINT:
                Long longValue = Long.valueOf(datum.getScalarValue());
                return parseColumnName(info) + longValue;
            case INTEGER:
                Integer intValue = Integer.valueOf(datum.getScalarValue());
                return parseColumnName(info) + intValue;
            case BOOLEAN:
                Boolean booleanValue = Boolean.valueOf(datum.getScalarValue());
                return parseColumnName(info) + booleanValue;
            case DOUBLE:
                Double doubleValue = Double.valueOf(datum.getScalarValue());
                return parseColumnName(info) + doubleValue;
            case TIMESTAMP:
                return parseColumnName(info) +
LocalDateTime.parse(datum.getScalarValue(), TIMESTAMP_FORMATTER);
            case DATE:
                return parseColumnName(info) +
LocalDate.parse(datum.getScalarValue(), DATE_FORMATTER);
            case TIME:
                return parseColumnName(info) +
LocalTime.parse(datum.getScalarValue(), TIME_FORMATTER);
            case INTERVAL_DAY_TO_SECOND:
            case INTERVAL_YEAR_TO_MONTH:
                return parseColumnName(info) + datum.getScalarValue();
            case UNKNOWN:
                return parseColumnName(info) + datum.getScalarValue();
            default:
                throw new IllegalArgumentException("Given type is not valid: " +
info.getType().getScalarType());
        }
    }

    private String parseColumnName(ColumnInfo info) {
```

```
        return info.getName() == null ? "" : info.getName() + "=";
    }

    private String parseArray(ColumnInfo arrayColumnInfo, List<Datum> arrayValues) {
        List<String> arrayOutput = new ArrayList<>();
        for (Datum datum : arrayValues) {
            arrayOutput.add(parseDatum(arrayColumnInfo, datum));
        }
        return String.format("[%s]",
arrayOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }
```

Java v2

```
    private static final long ONE_GB_IN_BYTES = 1073741824L;

    private void parseQueryResult(QueryResponse response) {
        final QueryStatus currentStatusOfQuery = response.queryStatus();

        System.out.println("Query progress so far: " +
currentStatusOfQuery.progressPercentage() + "%");

        double bytesScannedSoFar = ((double)
currentStatusOfQuery.cumulativeBytesScanned() / ONE_GB_IN_BYTES);
        System.out.println("Bytes scanned so far: " + bytesScannedSoFar + " GB");

        double bytesMeteredSoFar = ((double)
currentStatusOfQuery.cumulativeBytesMetered() / ONE_GB_IN_BYTES);
        System.out.println("Bytes metered so far: " + bytesMeteredSoFar + " GB");

        List<ColumnInfo> columnInfo = response.columnInfo();
        List<Row> rows = response.rows();

        System.out.println("Metadata: " + columnInfo);
        System.out.println("Data: ");

        // iterate every row
        for (Row row : rows) {
            System.out.println(parseRow(columnInfo, row));
        }
    }

    private String parseRow(List<ColumnInfo> columnInfo, Row row) {
```

```
        List<Datum> data = row.data();
        List<String> rowOutput = new ArrayList<>();
        // iterate every column per row
        for (int j = 0; j < data.size(); j++) {
            ColumnInfo info = columnInfo.get(j);
            Datum datum = data.get(j);
            rowOutput.add(parseDatum(info, datum));
        }
        return String.format("{%s}",
rowOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseDatum(ColumnInfo info, Datum datum) {
        if (datum.nullValue() != null && datum.nullValue()) {
            return info.name() + "=" + "NULL";
        }
        Type columnType = info.type();
        // If the column is of TimeSeries Type
        if (columnType.timeSeriesMeasureValueColumnInfo() != null) {
            return parseTimeSeries(info, datum);
        }
        // If the column is of Array Type
        else if (columnType.arrayColumnInfo() != null) {
            List<Datum> arrayValues = datum.arrayValue();
            return info.name() + "=" + parseArray(info.type().arrayColumnInfo(),
arrayValues);
        }
        // If the column is of Row Type
        else if (columnType.rowColumnInfo() != null &&
columnType.rowColumnInfo().size() > 0) {
            List<ColumnInfo> rowColumnInfo = info.type().rowColumnInfo();
            Row rowValues = datum.rowValue();
            return parseRow(rowColumnInfo, rowValues);
        }
        // If the column is of Scalar Type
        else {
            return parseScalarType(info, datum);
        }
    }

    private String parseTimeSeries(ColumnInfo info, Datum datum) {
        List<String> timeSeriesOutput = new ArrayList<>();
        for (TimeSeriesDataPoint dataPoint : datum.timeSeriesValue()) {
            timeSeriesOutput.add("{time=" + dataPoint.time() + ", value=" +
```

```
                    parseDatum(info.type().timeSeriesMeasureValueColumnInfo(),
 dataPoint.value()) + "}");
        }
        return String.format("[%s]",
 timeSeriesOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseScalarType(ColumnInfo info, Datum datum) {
        return parseColumnName(info) + datum.scalarValue();
    }

    private String parseColumnName(ColumnInfo info) {
        return info.name() == null ? "" : info.name() + "=";
    }

    private String parseArray(ColumnInfo arrayColumnInfo, List<Datum> arrayValues) {
        List<String> arrayOutput = new ArrayList<>();
        for (Datum datum : arrayValues) {
            arrayOutput.add(parseDatum(arrayColumnInfo, datum));
        }
        return String.format("[%s]",
 arrayOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }
```

Go

```
func processScalarType(data *timestreamquery.Datum) string {
    return *data.ScalarValue
}

func processTimeSeriesType(data []*timestreamquery.TimeSeriesDataPoint, columnInfo
 *timestreamquery.ColumnInfo) string {
    value := ""
    for k := 0; k < len(data); k++ {
        time := data[k].Time
        value += *time + ":"
        if columnInfo.Type.ScalarType != nil {
            value += processScalarType(data[k].Value)
        } else if columnInfo.Type.ArrayColumnInfo != nil {
            value += processArrayType(data[k].Value.ArrayValue,
 columnInfo.Type.ArrayColumnInfo)
        } else if columnInfo.Type.RowColumnInfo != nil {
```

```
            value += processRowType(data[k].Value.RowValue.Data,
 columnInfo.Type.RowColumnInfo)
        } else {
            fail("Bad data type")
        }
        if k != len(data)-1 {
            value += ", "
        }
    }
    return value
}

func processArrayType(datumList []*timestreamquery.Datum, columnInfo
 *timestreamquery.ColumnInfo) string {
    value := ""
    for k := 0; k < len(datumList); k++ {
        if columnInfo.Type.ScalarType != nil {
            value += processScalarType(datumList[k])
        } else if columnInfo.Type.TimeSeriesMeasureValueColumnInfo != nil {
            value += processTimeSeriesType(datumList[k].TimeSeriesValue,
 columnInfo.Type.TimeSeriesMeasureValueColumnInfo)
        } else if columnInfo.Type.ArrayColumnInfo != nil {
            value += "["
            value += processArrayType(datumList[k].ArrayValue,
 columnInfo.Type.ArrayColumnInfo)
            value += "]"
        } else if columnInfo.Type.RowColumnInfo != nil {
            value += "["
            value += processRowType(datumList[k].RowValue.Data,
 columnInfo.Type.RowColumnInfo)
            value += "]"
        } else {
            fail("Bad column type")
        }

        if k != len(datumList)-1 {
            value += ", "
        }
    }
    return value
}

func processRowType(data []*timestreamquery.Datum, metadata
 []*timestreamquery.ColumnInfo) string {
```

```
    value := ""
    for j := 0; j < len(data); j++ {
        if metadata[j].Type.ScalarType != nil {
            // process simple data types
            value += processScalarType(data[j])
        } else if metadata[j].Type.TimeSeriesMeasureValueColumnInfo != nil {
            // fmt.Println("Timeseries measure value column info")
            // fmt.Println(metadata[j].Type.TimeSeriesMeasureValueColumnInfo.Type)
            datapointList := data[j].TimeSeriesValue
            value += "["
            value += processTimeSeriesType(datapointList,
 metadata[j].Type.TimeSeriesMeasureValueColumnInfo)
            value += "]"
        } else if metadata[j].Type.ArrayColumnInfo != nil {
            columnInfo := metadata[j].Type.ArrayColumnInfo
            // fmt.Println("Array column info")
            // fmt.Println(columnInfo)
            datumList := data[j].ArrayValue
            value += "["
            value += processArrayType(datumList, columnInfo)
            value += "]"
        } else if metadata[j].Type.RowColumnInfo != nil {
            columnInfo := metadata[j].Type.RowColumnInfo
            datumList := data[j].RowValue.Data
            value += "["
            value += processRowType(datumList, columnInfo)
            value += "]"
        } else {
            panic("Bad column type")
        }
        // comma seperated column values
        if j != len(data)-1 {
            value += ", "
        }
    }
    return value
}
```

Python

```
    def _parse_query_result(self, query_result):
        query_status = query_result["QueryStatus"]
```

```
        progress_percentage = query_status["ProgressPercentage"]
        print(f"Query progress so far: {progress_percentage}%")

        bytes_scanned = float(query_status["CumulativeBytesScanned"]) /
 ONE_GB_IN_BYTES
        print(f"Data scanned so far: {bytes_scanned} GB")

        bytes_metered = float(query_status["CumulativeBytesMetered"]) /
 ONE_GB_IN_BYTES
        print(f"Data metered so far: {bytes_metered} GB")

        column_info = query_result['ColumnInfo']

        print("Metadata: %s" % column_info)
        print("Data: ")
        for row in query_result['Rows']:
            print(self._parse_row(column_info, row))

    def _parse_row(self, column_info, row):
        data = row['Data']
        row_output = []
        for j in range(len(data)):
            info = column_info[j]
            datum = data[j]
            row_output.append(self._parse_datum(info, datum))

        return "{%s}" % str(row_output)

    def _parse_datum(self, info, datum):
        if datum.get('NullValue', False):
            return "%s=NULL" % info['Name'],

        column_type = info['Type']

        # If the column is of TimeSeries Type
        if 'TimeSeriesMeasureValueColumnInfo' in column_type:
            return self._parse_time_series(info, datum)

        # If the column is of Array Type
        elif 'ArrayColumnInfo' in column_type:
            array_values = datum['ArrayValue']
            return "%s=%s" % (info['Name'], self._parse_array(info['Type']
['ArrayColumnInfo'], array_values))
```

```
        # If the column is of Row Type
        elif 'RowColumnInfo' in column_type:
            row_column_info = info['Type']['RowColumnInfo']
            row_values = datum['RowValue']
            return self._parse_row(row_column_info, row_values)

        # If the column is of Scalar Type
        else:
            return self._parse_column_name(info) + datum['ScalarValue']

    def _parse_time_series(self, info, datum):
        time_series_output = []
        for data_point in datum['TimeSeriesValue']:
            time_series_output.append("{time=%s, value=%s}"
                                      % (data_point['Time'],
                                         self._parse_datum(info['Type']
['TimeSeriesMeasureValueColumnInfo'],
                                                           data_point['Value'])))
        return "[%s]" % str(time_series_output)

    def _parse_array(self, array_column_info, array_values):
        array_output = []
        for datum in array_values:
            array_output.append(self._parse_datum(array_column_info, datum))

        return "[%s]" % str(array_output)

    @staticmethod
    def _parse_column_name(info):
        if 'Name' in info:
            return info['Name'] + "="
        else:
            return ""
```

Node.js

以下代码段使用 for JavaScript V2 样式。 AWS SDK它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
function parseQueryResult(response) {
    const queryStatus = response.QueryStatus;
    console.log("Current query status: " + JSON.stringify(queryStatus));
```

```
    const columnInfo = response.ColumnInfo;
    const rows = response.Rows;

    console.log("Metadata: " + JSON.stringify(columnInfo));
    console.log("Data: ");

    rows.forEach(function (row) {
        console.log(parseRow(columnInfo, row));
    });
}

function parseRow(columnInfo, row) {
    const data = row.Data;
    const rowOutput = [];

    var i;
    for ( i = 0; i < data.length; i++ ) {
        info = columnInfo[i];
        datum = data[i];
        rowOutput.push(parseDatum(info, datum));
    }

    return `{${rowOutput.join(", ")}}`
}

function parseDatum(info, datum) {
    if (datum.NullValue != null && datum.NullValue === true) {
        return `${info.Name}=NULL`;
    }

    const columnType = info.Type;

    // If the column is of TimeSeries Type
    if (columnType.TimeSeriesMeasureValueColumnInfo != null) {
        return parseTimeSeries(info, datum);
    }
    // If the column is of Array Type
    else if (columnType.ArrayColumnInfo != null) {
        const arrayValues = datum.ArrayValue;
        return `${info.Name}=${parseArray(info.Type.ArrayColumnInfo, arrayValues)}`;
    }
    // If the column is of Row Type
    else if (columnType.RowColumnInfo != null) {
        const rowColumnInfo = info.Type.RowColumnInfo;
```

```
        const rowValues = datum.RowValue;
        return parseRow(rowColumnInfo, rowValues);
    }
    // If the column is of Scalar Type
    else {
        return parseScalarType(info, datum);
    }
}

function parseTimeSeries(info, datum) {
    const timeSeriesOutput = [];
    datum.TimeSeriesValue.forEach(function (dataPoint) {
        timeSeriesOutput.push(`{time=${dataPoint.Time}, value=
${parseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, dataPoint.Value)}}`)
    });

    return `[${timeSeriesOutput.join(", ")}]`
}

function parseScalarType(info, datum) {
    return parseColumnName(info) + datum.ScalarValue;
}

function parseColumnName(info) {
    return info.Name == null ? "" : `${info.Name}=`;
}

function parseArray(arrayColumnInfo, arrayValues) {
    const arrayOutput = [];
    arrayValues.forEach(function (datum) {
        arrayOutput.push(parseDatum(arrayColumnInfo, datum));
    });
    return `[${arrayOutput.join(", ")}]`
}
```

.NET

```
        private void ParseQueryResult(QueryResponse response)
        {
            List<ColumnInfo> columnInfo = response.ColumnInfo;
            var options = new JsonSerializerOptions
            {
                IgnoreNullValues = true
```

```
            };
            List<String> columnInfoStrings = columnInfo.ConvertAll(x =>
JsonSerializer.Serialize(x, options));
            List<Row> rows = response.Rows;

            QueryStatus queryStatus = response.QueryStatus;
            Console.WriteLine("Current Query status:" +
JsonSerializer.Serialize(queryStatus, options));

            Console.WriteLine("Metadata:" + string.Join(",", columnInfoStrings));
            Console.WriteLine("Data:");

            foreach (Row row in rows)
            {
                Console.WriteLine(ParseRow(columnInfo, row));
            }
        }

        private string ParseRow(List<ColumnInfo> columnInfo, Row row)
        {
            List<Datum> data = row.Data;
            List<string> rowOutput = new List<string>();
            for (int j = 0; j < data.Count; j++)
            {
                ColumnInfo info = columnInfo[j];
                Datum datum = data[j];
                rowOutput.Add(ParseDatum(info, datum));
            }
            return $"{{{string.Join(",", rowOutput)}}}";
        }

        private string ParseDatum(ColumnInfo info, Datum datum)
        {
            if (datum.NullValue)
            {
                return $"{info.Name}=NULL";
            }

            Amazon.TimestreamQuery.Model.Type columnType = info.Type;
            if (columnType.TimeSeriesMeasureValueColumnInfo != null)
            {
                return ParseTimeSeries(info, datum);
            }
            else if (columnType.ArrayColumnInfo != null)
```

```
            {
                List<Datum> arrayValues = datum.ArrayValue;
                return $"{info.Name}={ParseArray(info.Type.ArrayColumnInfo,
arrayValues)}";
            }
            else if (columnType.RowColumnInfo != null &&
columnType.RowColumnInfo.Count > 0)
            {
                List<ColumnInfo> rowColumnInfo = info.Type.RowColumnInfo;
                Row rowValue = datum.RowValue;
                return ParseRow(rowColumnInfo, rowValue);
            }
            else
            {
                return ParseScalarType(info, datum);
            }
        }

        private string ParseTimeSeries(ColumnInfo info, Datum datum)
        {
            var timeseriesString = datum.TimeSeriesValue
                .Select(value => $"{{time={value.Time},
value={ParseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, value.Value)}}}")
                .Aggregate((current, next) => current + "," + next);

            return $"[{timeseriesString}]";
        }

        private string ParseScalarType(ColumnInfo info, Datum datum)
        {
            return ParseColumnName(info) + datum.ScalarValue;
        }

        private string ParseColumnName(ColumnInfo info)
        {
            return info.Name == null ? "" : (info.Name + "=");
        }

        private string ParseArray(ColumnInfo arrayColumnInfo, List<Datum>
arrayValues)
        {
            return $"[{arrayValues.Select(value => ParseDatum(arrayColumnInfo,
value)).Aggregate((current, next) => current + "," + next)}]";
```

```
        }
```

访问查询状态

您可以通过访问查询状态QueryResponse，其中包含有关查询进度、查询扫描的字节和查询计量的字节的信息。bytesMetered和bytesScanned值是累积的，在分页查询结果时会持续更新。您可以使用此信息来了解单个查询所扫描的字节，也可以使用它来做出某些决策。例如，假设查询价格为每扫描 GB 0.01 美元，则可能需要取消每次查询超过 25 美元或 X GB 的查询。下面的代码片段显示了如何做到这一点。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
   private static final long ONE_GB_IN_BYTES = 1073741824L;
   private static final double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
price of query is $0.01 per GB

   public void cancelQueryBasedOnQueryStatus() {
       System.out.println("Starting query: " + SELECT_ALL_QUERY);
       QueryRequest queryRequest = new QueryRequest();
       queryRequest.setQueryString(SELECT_ALL_QUERY);
       QueryResult queryResult = queryClient.query(queryRequest);

       while (true) {
           final QueryStatus currentStatusOfQuery = queryResult.getQueryStatus();
           System.out.println("Query progress so far: " +
currentStatusOfQuery.getProgressPercentage() + "%");
           double bytesMeteredSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesMetered() / ONE_GB_IN_BYTES);
           System.out.println("Bytes metered so far: " + bytesMeteredSoFar + "
GB");
           // Cancel query if its costing more than 1 cent
           if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01) {
               cancelQuery(queryResult);
               break;
           }
```

```
            if (queryResult.getNextToken() == null) {
                break;
            }
            queryRequest.setNextToken(queryResult.getNextToken());
            queryResult = queryClient.query(queryRequest);
        }
    }
```

Java v2

```
    private static final long ONE_GB_IN_BYTES = 1073741824L;
    private static final double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
price of query is $0.01 per GB

    public void cancelQueryBasedOnQueryStatus() {
        System.out.println("Starting query: " + SELECT_ALL_QUERY);
        QueryRequest queryRequest =
QueryRequest.builder().queryString(SELECT_ALL_QUERY).build();

        final QueryIterable queryResponseIterator =
timestreamQueryClient.queryPaginator(queryRequest);
        for(QueryResponse queryResponse : queryResponseIterator) {
            final QueryStatus currentStatusOfQuery = queryResponse.queryStatus();
            System.out.println("Query progress so far: " +
currentStatusOfQuery.progressPercentage() + "%");
            double bytesMeteredSoFar = ((double)
currentStatusOfQuery.cumulativeBytesMetered() / ONE_GB_IN_BYTES);
            System.out.println("Bytes metered so far: " + bytesMeteredSoFar + "GB");
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01) {
                cancelQuery(queryResponse);
                break;
            }
        }
    }
```

Go

```
const OneGbInBytes = 1073741824
// Assuming the price of query is $0.01 per GB
const QueryCostPerGbInDollars = 0.01
```

```
func cancelQueryBasedOnQueryStatus(queryPtr *string, querySvc
 *timestreamquery.TimestreamQuery, f *os.File) {
    queryInput := &timestreamquery.QueryInput{
        QueryString: aws.String(*queryPtr),
    }
    fmt.Println("QueryInput:")
    fmt.Println(queryInput)
    // execute the query
    err := querySvc.QueryPages(queryInput,
        func(page *timestreamquery.QueryOutput, lastPage bool) bool {
            // process query response
            queryStatus := page.QueryStatus
            fmt.Println("Current query status:", queryStatus)
            bytes_metered := float64(*queryStatus.CumulativeBytesMetered) /
 float64(ONE_GB_IN_BYTES)
            if bytes_metered * QUERY_COST_PER_GB_IN_DOLLARS > 0.01 {
                cancelQuery(page, querySvc)
                return true
            }
            // query response metadata
            // includes column names and types
            metadata := page.ColumnInfo
            // fmt.Println("Metadata:")
            fmt.Println(metadata)
            header := ""
            for i := 0; i < len(metadata); i++ {
                header += *metadata[i].Name
                if i != len(metadata)-1 {
                    header += ", "
                }
            }
            write(f, header)

            // query response data
            fmt.Println("Data:")
            // process rows
            rows := page.Rows
            for i := 0; i < len(rows); i++ {
                data := rows[i].Data
                value := processRowType(data, metadata)
                fmt.Println(value)
                write(f, value)
            }
            fmt.Println("Number of rows:", len(page.Rows))
```

```
            return true
        })
    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    }
}
```

Python

```
ONE_GB_IN_BYTES = 1073741824
# Assuming the price of query is $0.01 per GB
QUERY_COST_PER_GB_IN_DOLLARS = 0.01

    def cancel_query_based_on_query_status(self):
        try:
            print("Starting query: " + self.SELECT_ALL)
            page_iterator = self.paginator.paginate(QueryString=self.SELECT_ALL)
            for page in page_iterator:
                query_status = page["QueryStatus"]
                progress_percentage = query_status["ProgressPercentage"]
                print("Query progress so far: " + str(progress_percentage) + "%")
                bytes_metered = query_status["CumulativeBytesMetered"] /
 self.ONE_GB_IN_BYTES
                print("Bytes Metered so far: " + str(bytes_metered) + " GB")
                if bytes_metered * self.QUERY_COST_PER_GB_IN_DOLLARS > 0.01:
                    self.cancel_query_for(page)
                    break
        except Exception as err:
            print("Exception while running query:", err)
            traceback.print_exc(file=sys.stderr)
```

Node.js

以下代码段使用 for JavaScript V2 样式。 AWS SDK它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
function parseQueryResult(response) {
    const queryStatus = response.QueryStatus;
    console.log("Current query status: " + JSON.stringify(queryStatus));

    const columnInfo = response.ColumnInfo;
    const rows = response.Rows;
```

```
    console.log("Metadata: " + JSON.stringify(columnInfo));
    console.log("Data: ");

    rows.forEach(function (row) {
        console.log(parseRow(columnInfo, row));
    });
}

function parseRow(columnInfo, row) {
    const data = row.Data;
    const rowOutput = [];

    var i;
    for ( i = 0; i < data.length; i++ ) {
        info = columnInfo[i];
        datum = data[i];
        rowOutput.push(parseDatum(info, datum));
    }

    return `{${rowOutput.join(", ")}}`
}

function parseDatum(info, datum) {
    if (datum.NullValue != null && datum.NullValue === true) {
        return `${info.Name}=NULL`;
    }

    const columnType = info.Type;

    // If the column is of TimeSeries Type
    if (columnType.TimeSeriesMeasureValueColumnInfo != null) {
        return parseTimeSeries(info, datum);
    }
    // If the column is of Array Type
    else if (columnType.ArrayColumnInfo != null) {
        const arrayValues = datum.ArrayValue;
        return `${info.Name}=${parseArray(info.Type.ArrayColumnInfo, arrayValues)}`;
    }
    // If the column is of Row Type
    else if (columnType.RowColumnInfo != null) {
        const rowColumnInfo = info.Type.RowColumnInfo;
        const rowValues = datum.RowValue;
        return parseRow(rowColumnInfo, rowValues);
```

```
    }
    // If the column is of Scalar Type
    else {
        return parseScalarType(info, datum);
    }
}

function parseTimeSeries(info, datum) {
    const timeSeriesOutput = [];
    datum.TimeSeriesValue.forEach(function (dataPoint) {
        timeSeriesOutput.push(`{time=${dataPoint.Time}, value=
${parseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, dataPoint.Value)}}`)
    });

    return `[${timeSeriesOutput.join(", ")}]`
}

function parseScalarType(info, datum) {
    return parseColumnName(info) + datum.ScalarValue;
}

function parseColumnName(info) {
    return info.Name == null ? "" : `${info.Name}=`;
}

function parseArray(arrayColumnInfo, arrayValues) {
    const arrayOutput = [];
    arrayValues.forEach(function (datum) {
        arrayOutput.push(parseDatum(arrayColumnInfo, datum));
    });
    return `[${arrayOutput.join(", ")}]`
}
```

.NET

```
private static readonly long ONE_GB_IN_BYTES = 1073741824L;
private static readonly double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
 price of query is $0.01 per GB

private async Task CancelQueryBasedOnQueryStatus(string queryString)
{
    try
    {
```

```
        QueryRequest queryRequest = new QueryRequest();
        queryRequest.QueryString = queryString;
        QueryResponse queryResponse = await queryClient.QueryAsync(queryRequest);
        while (true)
        {
            QueryStatus queryStatus = queryResponse.QueryStatus;
            double bytesMeteredSoFar = ((double)
 queryStatus.CumulativeBytesMetered / ONE_GB_IN_BYTES);
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01)
            {
                await CancelQuery(queryResponse);
                break;
            }

            ParseQueryResult(queryResponse);
            if (queryResponse.NextToken == null)
            {
                break;
            }
            queryRequest.NextToken = queryResponse.NextToken;
            queryResponse = await queryClient.QueryAsync(queryRequest);
       }
    } catch(Exception e)
    {
        // Some queries might fail with 500 if the result of a sequence function has
 more than 10000 entries
        Console.WriteLine(e.ToString());
    }
}
```

有关如何取消查询的更多详细信息，请参阅取消查询。

运行UNLOAD查询

以下代码示例调用UNLOAD查询。有关 UNLOAD 的信息，请参阅UNLOAD用于将查询结果从 Timestream 导出到 S3 LiveAnalytics。有关UNLOAD查询的示例，请参阅来UNLOAD自 Timestream 的示例用例 LiveAnalytics。

主题

- 生成并运行UNLOAD查询

- 解析UNLOAD响应并获取行数、清单链接和元数据链接
- 读取和解析清单内容
- 读取和解析元数据内容

生成并运行UNLOAD查询

Java

```
// When you have a SELECT like below

String QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
        + DATABASE_NAME + "." + UNLOAD_TABLE_NAME
        + " WHERE time BETWEEN ago(2d) AND now()";

// You can construct UNLOAD query as follows
UnloadQuery unloadQuery = UnloadQuery.builder()
        .selectQuery(QUERY_1)
        .bucketName("timestream-sample-<region>-<accountId>")
        .resultsPrefix("without_partition")
        .format(CSV)
        .compression(UnloadQuery.Compression.GZIP)
        .build();
QueryResult unloadResult = runQuery(unloadQuery.getUnloadQuery());

// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
    private QueryResult runQuery(String queryString) {
        QueryResult queryResult = null;
        try {
            QueryRequest queryRequest = new QueryRequest();
            queryRequest.setQueryString(queryString);
            queryResult = queryClient.query(queryRequest);
            while (true) {
                parseQueryResult(queryResult);
                if (queryResult.getNextToken() == null) {
                    break;
                }
                queryRequest.setNextToken(queryResult.getNextToken());
                queryResult = queryClient.query(queryRequest);
            }
```

```
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
 has more than 10000 entries
            e.printStackTrace();
        }
        return queryResult;
    }

// Utility that helps to construct UNLOAD query

@Builder
static class UnloadQuery {
    private String selectQuery;
    private String bucketName;
    private String resultsPrefix;
    private Format format;
    private Compression compression;
    private EncryptionType encryptionType;
    private List<String> partitionColumns;
    private String kmsKey;
    private Character csvFieldDelimiter;
    private Character csvEscapeCharacter;

    public String getUnloadQuery() {
        String destination = constructDestination();
        String withClause = constructOptionalParameters();
        return String.format("UNLOAD (%s) TO '%s' %s", selectQuery, destination,
 withClause);
    }

    private String constructDestination() {
        return "s3://" + this.bucketName + "/" + this.resultsPrefix + "/";
    }

    private String constructOptionalParameters() {
        boolean isOptionalParametersPresent = Objects.nonNull(format)
                || Objects.nonNull(compression)
                || Objects.nonNull(encryptionType)
                || Objects.nonNull(partitionColumns)
                || Objects.nonNull(kmsKey)
                || Objects.nonNull(csvFieldDelimiter)
                || Objects.nonNull(csvEscapeCharacter);

        String withClause = "";
```

```
        if (isOptionalParametersPresent) {
            StringJoiner optionalParameters = new StringJoiner(",");
            if (Objects.nonNull(format)) {
                optionalParameters.add("format = '" + format + "'");
            }
            if (Objects.nonNull(compression)) {
                optionalParameters.add("compression = '" + compression + "'");
            }
            if (Objects.nonNull(encryptionType)) {
                optionalParameters.add("encryption = '" + encryptionType + "'");
            }
            if (Objects.nonNull(kmsKey)) {
                optionalParameters.add("kms_key = '" + kmsKey + "'");
            }
            if (Objects.nonNull(csvFieldDelimiter)) {
                optionalParameters.add("field_delimiter = '" + csvFieldDelimiter +
"'");
            }
            if (Objects.nonNull(csvEscapeCharacter)) {
                optionalParameters.add("escaped_by = '" + csvEscapeCharacter + "'");
            }
            if (Objects.nonNull(partitionColumns) && !partitionColumns.isEmpty()) {
                final StringJoiner partitionedByList = new StringJoiner(",");
                partitionColumns.forEach(column -> partitionedByList.add("'" +
column + "'"));
                optionalParameters.add(String.format("partitioned_by = ARRAY[%s]",
partitionedByList));
            }
            withClause = String.format("WITH (%s)", optionalParameters);
        }
        return withClause;
    }

    public enum Format {
        CSV, PARQUET
    }

    public enum Compression {
        GZIP, NONE
    }

    public enum EncryptionType {
        SSE_S3, SSE_KMS
    }
```

```
    @Override
    public String toString() {
        return getUnloadQuery();
    }
}
```

Java v2

```
// When you have a SELECT like below

String QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
        + DATABASE_NAME + "." + UNLOAD_TABLE_NAME
        + " WHERE time BETWEEN ago(2d) AND now()";

//You can construct UNLOAD query as follows
UnloadQuery unloadQuery = UnloadQuery.builder()
        .selectQuery(QUERY_1)
        .bucketName("timestream-sample-<region>-<accountId>")
        .resultsPrefix("without_partition")
        .format(CSV)
        .compression(UnloadQuery.Compression.GZIP)
        .build();

QueryResponse unloadResponse = runQuery(unloadQuery.getUnloadQuery());


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
private QueryResponse runQuery(String queryString) {
   QueryResponse finalResponse = null;
    try {
        QueryRequest queryRequest =
 QueryRequest.builder().queryString(queryString).build();
        final QueryIterable queryResponseIterator =
 timestreamQueryClient.queryPaginator(queryRequest);
        for(QueryResponse queryResponse : queryResponseIterator) {
            parseQueryResult(queryResponse);
           finalResponse = queryResponse;
        }
    } catch (Exception e) {
```

```
        // Some queries might fail with 500 if the result of a sequence function has
 more than 10000 entries
        e.printStackTrace();
    }
   return finalResponse;
}

// Utility that helps to construct UNLOAD query
@Builder
static class UnloadQuery {
    private String selectQuery;
    private String bucketName;
    private String resultsPrefix;
    private Format format;
    private Compression compression;
    private EncryptionType encryptionType;
    private List<String> partitionColumns;
    private String kmsKey;
    private Character csvFieldDelimiter;
    private Character csvEscapeCharacter;

    public String getUnloadQuery() {
        String destination = constructDestination();
        String withClause = constructOptionalParameters();
        return String.format("UNLOAD (%s) TO '%s' %s", selectQuery, destination,
 withClause);
    }

    private String constructDestination() {
        return "s3://" + this.bucketName + "/" + this.resultsPrefix + "/";
    }

    private String constructOptionalParameters() {
        boolean isOptionalParametersPresent = Objects.nonNull(format)
                || Objects.nonNull(compression)
                || Objects.nonNull(encryptionType)
                || Objects.nonNull(partitionColumns)
                || Objects.nonNull(kmsKey)
                || Objects.nonNull(csvFieldDelimiter)
                || Objects.nonNull(csvEscapeCharacter);

        String withClause = "";
        if (isOptionalParametersPresent) {
            StringJoiner optionalParameters = new StringJoiner(",");
```

```
            if (Objects.nonNull(format)) {
                optionalParameters.add("format = '" + format + "'");
            }
            if (Objects.nonNull(compression)) {
                optionalParameters.add("compression = '" + compression + "'");
            }
            if (Objects.nonNull(encryptionType)) {
                optionalParameters.add("encryption = '" + encryptionType + "'");
            }
            if (Objects.nonNull(kmsKey)) {
                optionalParameters.add("kms_key = '" + kmsKey + "'");
            }
            if (Objects.nonNull(csvFieldDelimiter)) {
                optionalParameters.add("field_delimiter = '" + csvFieldDelimiter +
"'");
            }
            if (Objects.nonNull(csvEscapeCharacter)) {
                optionalParameters.add("escaped_by = '" + csvEscapeCharacter + "'");
            }
            if (Objects.nonNull(partitionColumns) && !partitionColumns.isEmpty()) {
                final StringJoiner partitionedByList = new StringJoiner(",");
                partitionColumns.forEach(column -> partitionedByList.add("'" +
column + "'"));
                optionalParameters.add(String.format("partitioned_by = ARRAY[%s]",
partitionedByList));
            }
            withClause = String.format("WITH (%s)", optionalParameters);
        }
        return withClause;
    }

    public enum Format {
        CSV, PARQUET
    }

    public enum Compression {
        GZIP, NONE
    }

    public enum EncryptionType {
        SSE_S3, SSE_KMS
    }

    @Override
```

```
    public String toString() {
        return getUnloadQuery();
    }
}
```

Go

```
// When you have a SELECT like below
var Query = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
+ *databaseName + "." + *tableName + " WHERE time BETWEEN ago(2d) AND now()"

// You can construct UNLOAD query as follows
var unloadQuery = UnloadQuery{
    Query: "SELECT user_id, ip_address, session_id, measure_name, time, query,
 quantity, product_id, channel, event FROM " + *databaseName + "." + *tableName +
    " WHERE time BETWEEN ago(2d) AND now()",
    Partitioned_by: []string{},
    Compression: "GZIP",
    Format: "CSV",
    S3Location: bucketName,
    ResultPrefix: "without_partition",
}


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination

queryInput := &timestreamquery.QueryInput{
    QueryString: build_query(unloadQuery),
}

err := querySvc.QueryPages(queryInput,
    func(page *timestreamquery.QueryOutput, lastPage bool) bool {
        if (lastPage) {
            var response = parseQueryResult(page)
            var unloadFiles = getManifestAndMetadataFiles(s3Svc, response)
            displayColumns(unloadFiles, unloadQuery.Partitioned_by)
            displayResults(s3Svc, unloadFiles)
        }
        return true
    })
```

```
if err != nil {
    fmt.Println("Error:")
    fmt.Println(err)
}

// Utility that helps to construct UNLOAD query
type UnloadQuery struct {
    Query string
    Partitioned_by []string
    Format string
    S3Location string
    ResultPrefix string
    Compression string
}

func build_query(unload_query UnloadQuery) *string {
    var query_results_s3_path = "'s3://" + unload_query.S3Location + "/" +
 unload_query.ResultPrefix + "/'"
    var query = "UNLOAD(" + unload_query.Query + ") TO " + query_results_s3_path + "
 WITH ( "
    if (len(unload_query.Partitioned_by) > 0) {
        query = query + "partitioned_by=ARRAY["
        for i, column := range unload_query.Partitioned_by {
            if i == 0 {
                query = query + "'" + column + "'"
            } else {
                query = query + ",'" + column + "'"
            }
        }
        query = query + "],"
    }
    query = query + " format='" + unload_query.Format + "', "
    query = query + "  compression='" + unload_query.Compression + "')"
    fmt.Println(query)
    return aws.String(query)
}
```

Python

```
# When you have a SELECT like below
QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel FROM "
```

```
        + database_name + "." + table_name + " WHERE time BETWEEN ago(2d) AND now()"
# You can construct UNLOAD query as follows
UNLOAD_QUERY_1 = UnloadQuery(QUERY_1, "timestream-sample-<region>-<accountId>",
 "without_partition", "CSV", "GZIP", "")

# Run UNLOAD query (Similar to how you run SELECT query)
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
def run_query(self, query_string):
    try:
        page_iterator = self.paginator.paginate(QueryString=UNLOAD_QUERY_1)
    except Exception as err:
        print("Exception while running query:", err)

# Utility that helps to construct UNLOAD query
class UnloadQuery:
    def __init__(self, query, s3_bucket_location, results_prefix, format,
 compression , partition_by):
        self.query = query
        self.s3_bucket_location = s3_bucket_location
        self.results_prefix = results_prefix
        self.format = format
        self.compression = compression
        self.partition_by = partition_by

    def build_query(self):
        query_results_s3_path = "'s3://" + self.s3_bucket_location + "/" +
 self.results_prefix + "/'"
        unload_query = "UNLOAD("
        unload_query = unload_query + self.query
        unload_query = unload_query + ") "
        unload_query = unload_query + " TO " + query_results_s3_path
        unload_query = unload_query + " WITH ( "

        if(len(self.partition_by) > 0) :
            unload_query = unload_query + " partitioned_by = ARRAY" +
 str(self.partition_by) + ","

        unload_query = unload_query + " format='" + self.format  + "', "
        unload_query = unload_query + "  compression='" + self.compression + "')"

        return unload_query
```

Node.js

```
// When you have a SELECT like below
QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel FROM "
        + database_name + "." + table_name + " WHERE time BETWEEN ago(2d) AND now()"
// You can construct UNLOAD query as follows
UNLOAD_QUERY_1 = new UnloadQuery(QUERY_1, "timestream-sample-<region>-<accountId>",
 "without_partition", "CSV", "GZIP", "")


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination

async runQuery(query = UNLOAD_QUERY_1, nextToken) {
    const params = new QueryCommand({
        QueryString: query
    });

    if (nextToken) {
        params.NextToken = nextToken;
    }

    await queryClient.send(params).then(
            (response) => {
                if (response.NextToken) {
                    runQuery(queryClient, query, response.NextToken);
                } else {
                    await parseAndDisplayResults(response);
                }
            },
            (err) => {
                console.error("Error while querying:", err);
            });
}


class UnloadQuery {
    constructor(query, s3_bucket_location, results_prefix, format, compression ,
 partition_by) {
        this.query = query;
        this.s3_bucket_location = s3_bucket_location
        this.results_prefix = results_prefix
```

```
        this.format = format
        this.compression = compression
        this.partition_by = partition_by
    }

    buildQuery() {
        const query_results_s3_path = "'s3://" + this.s3_bucket_location + "/" +
 this.results_prefix + "/'"
        let unload_query = "UNLOAD("
        unload_query = unload_query + this.query
        unload_query = unload_query + ") "
        unload_query = unload_query + " TO " + query_results_s3_path
        unload_query = unload_query + " WITH ( "

        if(this.partition_by.length > 0) {
            let partitionBy = ""
            this.partition_by.forEach((str, i) => {
                partitionBy = partitionBy + (i ? ",'" : "'") + str + "'"
            })
            unload_query = unload_query + " partitioned_by = ARRAY[" + partitionBy +
 "],"
        }
        unload_query = unload_query + " format='" + this.format  + "', "
        unload_query = unload_query + "  compression='" + this.compression + "')"

        return unload_query
    }
}
```

解析UNLOAD响应并获取行数、清单链接和元数据链接

Java

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

public UnloadResponse parseResult(QueryResult queryResult) {
    Map<String, String> outputMap = new HashMap<>();
```

```
    for (int i = 0; i < queryResult.getColumnInfo().size(); i++) {
        outputMap.put(queryResult.getColumnInfo().get(i).getName(),
                queryResult.getRows().get(0).getData().get(i).getScalarValue());

    }
    return new UnloadResponse(outputMap);
}

@Getter
class UnloadResponse {
    private final String metadataFile;
    private final String manifestFile;
    private final int rows;

    public UnloadResponse(Map<String, String> unloadResponse) {
        this.metadataFile = unloadResponse.get("metadataFile");
        this.manifestFile = unloadResponse.get("manifestFile");
        this.rows = Integer.parseInt(unloadResponse.get("rows"));
    }
}
```

Java v2

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

public UnloadResponse parseResult(QueryResponse queryResponse) {
    Map<String, String> outputMap = new HashMap<>();
    for (int i = 0; i < queryResponse.columnInfo().size(); i++) {
        outputMap.put(queryResponse.columnInfo().get(i).name(),
                queryResponse.rows().get(0).data().get(i).scalarValue());

    }
    return new UnloadResponse(outputMap);
}

@Getter
class UnloadResponse {
    private final String metadataFile;
```

```
    private final String manifestFile;
    private final int rows;

    public UnloadResponse(Map<String, String> unloadResponse) {
        this.metadataFile = unloadResponse.get("metadataFile");
        this.manifestFile = unloadResponse.get("manifestFile");
        this.rows = Integer.parseInt(unloadResponse.get("rows"));
    }
}
```

Go

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

func parseQueryResult(queryOutput *timestreamquery.QueryOutput) map[string]string {
    var columnInfo = queryOutput.ColumnInfo;
    fmt.Println("ColumnInfo", columnInfo)
    fmt.Println("QueryId", queryOutput.QueryId)
    fmt.Println("QueryStatus", queryOutput.QueryStatus)
    return parseResponse(columnInfo, queryOutput.Rows[0])
}

func parseResponse(columnInfo []*timestreamquery.ColumnInfo, row
 *timestreamquery.Row) map[string]string {
    var datum = row.Data
    response := make(map[string]string)
    for i, column := range columnInfo {
        response[*column.Name] = *datum[i].ScalarValue
    }
    return response
}
```

Python

```
# Parsing UNLOAD query response is similar to how you parse SELECT query response:
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing
```

```
# But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
# (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

for page in page_iterator:
  last_page = page
response = self._parse_unload_query_result(last_page)

def _parse_unload_query_result(self, query_result):
    column_info = query_result['ColumnInfo']

    print("ColumnInfo: %s" % column_info)
    print("QueryId: %s" % query_result['QueryId'])
    print("QueryStatus:%s" % query_result['QueryStatus'])
    return self.parse_unload_response(column_info, query_result['Rows'][0])

def parse_unload_response(self, column_info, row):
    response = {}
    data = row['Data']
    for i, column in enumerate(column_info):
        response[column['Name']] = data[i]['ScalarValue']
   print("Rows: %s" % response['rows'])
   print("Metadata File location: %s" % response['metadataFile'])
   print("Manifest File location: %s" % response['manifestFile'])
   return response
```

Node.js

```
# Parsing UNLOAD query response is similar to how you parse SELECT query response:
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

# But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
# (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

async parseAndDisplayResults(data, query) {
    const columnInfo = data['ColumnInfo'];
    console.log("ColumnInfo:", columnInfo)
    console.log("QueryId: %s", data['QueryId'])
    console.log("QueryStatus:", data['QueryStatus'])
    await this.parseResponse(columnInfo, data['Rows'][0], query)
}

async parseResponse(columnInfo, row, query) {
```

```
    let response = {}
    const data = row['Data']
    columnInfo.forEach((column, i) => {
        response[column['Name']] = data[i]['ScalarValue']
    })

   console.log("Manifest file", response['manifestFile']);
   console.log("Metadata file", response['metadataFile']);

   return response
}
```

读取和解析清单内容

Java

```
// Read and parse manifest content
public UnloadManifest getUnloadManifest(UnloadResponse unloadResponse) throws
 IOException {
   AmazonS3URI s3URI = new AmazonS3URI(unloadResponse.getManifestFile());
    S3Object s3Object = s3Client.getObject(s3URI.getBucket(), s3URI.getKey());
    String manifestFileContent = new
 String(IOUtils.toByteArray(s3Object.getObjectContent()), StandardCharsets.UTF_8);
    return new Gson().fromJson(manifestFileContent, UnloadManifest.class);
}

class UnloadManifest {
    @Getter
    public class FileMetadata {
        long content_length_in_bytes;
        long row_count;
    }

    @Getter
    public class ResultFile {
        String url;
        FileMetadata file_metadata;
    }

    @Getter
    public class QueryMetadata {
        long total_content_length_in_bytes;
```

```
        long total_row_count;
        String result_format;
        String result_version;
    }

    @Getter
    public class Author {
        String name;
        String manifest_file_version;
    }

    @Getter
    private List<ResultFile> result_files;
    @Getter
    private QueryMetadata query_metadata;
    @Getter
    private Author author;
}
```

Java v2

```
// Read and parse manifest content
public UnloadManifest getUnloadManifest(UnloadResponse unloadResponse) throws
 URISyntaxException {
    // Space needs to encoded to use S3 parseUri function
    S3Uri s3Uri =
 s3Utilities.parseUri(URI.create(unloadResponse.getManifestFile().replace(" ",
 "%20")));
    ResponseBytes<GetObjectResponse> objectBytes =
 s3Client.getObjectAsBytes(GetObjectRequest.builder()
            .bucket(s3Uri.bucket().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getManifestFile(), "Invalid S3 URI")))
            .key(s3Uri.key().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getManifestFile(), "Invalid S3 URI")))
            .build());
    String manifestFileContent = new String(objectBytes.asByteArray(),
 StandardCharsets.UTF_8);
    return new Gson().fromJson(manifestFileContent, UnloadManifest.class);
}

class UnloadManifest {
    @Getter
    public class FileMetadata {
```

```
        long content_length_in_bytes;
        long row_count;
    }

    @Getter
    public class ResultFile {
        String url;
        FileMetadata file_metadata;
    }

    @Getter
    public class QueryMetadata {
        long total_content_length_in_bytes;
        long total_row_count;
        String result_format;
        String result_version;
    }

    @Getter
    public class Author {
        String name;
        String manifest_file_version;
    }

    @Getter
    private List<ResultFile> result_files;
    @Getter
    private QueryMetadata query_metadata;
    @Getter
    private Author author;
}
```

Go

```
// Read and parse manifest content

func getManifestFile(s3Svc *s3.S3, response map[string]string) Manifest {
    var manifestBuf = getObject(s3Svc, response["manifestFile"])
    var manifest Manifest
    json.Unmarshal(manifestBuf.Bytes(), &manifest)
    return manifest
}
```

```
func getObject(s3Svc *s3.S3, s3Uri string) *bytes.Buffer {
    u,_ := url.Parse(s3Uri)
    getObjectInput := &s3.GetObjectInput{
        Key:    aws.String(u.Path),
        Bucket: aws.String(u.Host),
    }
    getObjectOutput, err := s3Svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Println("Error: %s\n", err.Error())
    }
    buf := new(bytes.Buffer)
    buf.ReadFrom(getObjectOutput.Body)
    return buf
}

// Unload's Manifest structure

type Manifest struct {
    Author interface{}
    Query_metadata map[string]any
    Result_files  []struct {
        File_metadata interface{}
        Url string
    }
}}
```

Python

```
def __get_manifest_file(self, response):
    manifest = self.get_object(response['manifestFile']).read().decode('utf-8')
    parsed_manifest = json.loads(manifest)
    print("Manifest contents: \n%s" % parsed_manifest)

def get_object(self, uri):
    try:
        bucket, key = uri.replace("s3://", "").split("/", 1)
        s3_client = boto3.client('s3', region_name=<region>)
        response = s3_client.get_object(Bucket=bucket, Key=key)
        return response['Body']
    except Exception as err:
        print("Failed to get the object for URI:", uri)
        raise err
```

Node.js

```
// Read and parse manifest content

async getManifestFile(response) {
    let manifest;
    await this.getS3Object(response['manifestFile']).then(
        (data) => {
            manifest = JSON.parse(data);
        }
    );
    return manifest;
}

async getS3Object(uri) {
    const {bucketName, key} = this.getBucketAndKey(uri);
    const params = new GetObjectCommand({
        Bucket: bucketName,
        Key: key
    })
    const response = await this.s3Client.send(params);
    return await response.Body.transformToString();
}

getBucketAndKey(uri) {
    const [bucketName] = uri.replace("s3://", "").split("/", 1);
    const key = uri.replace("s3://", "").split('/').slice(1).join('/');
    return {bucketName, key};
}
```

读取和解析元数据内容

Java

```
// Read and parse metadata content
public UnloadMetadata getUnloadMetadata(UnloadResponse unloadResponse) throws
 IOException {
   AmazonS3URI s3URI = new AmazonS3URI(unloadResponse.getMetadataFile());
   S3Object s3Object = s3Client.getObject(s3URI.getBucket(), s3URI.getKey());
    String metadataFileContent = new
 String(IOUtils.toByteArray(s3Object.getObjectContent()), StandardCharsets.UTF_8);
    final Gson gson = new GsonBuilder()
```

```
            .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
            .create();
    return gson.fromJson(metadataFileContent, UnloadMetadata.class);
}

class UnloadMetadata {
    @JsonProperty("ColumnInfo")
    List<ColumnInfo> columnInfo;
    @JsonProperty("Author")
    Author author;

    @Data
    public class Author {
        @JsonProperty("Name")
        String name;
        @JsonProperty("MetadataFileVersion")
        String metadataFileVersion;
    }
}
```

Java v2

```
// Read and parse metadata content

public UnloadMetadata getUnloadMetadata(UnloadResponse unloadResponse) throws
 URISyntaxException {
   // Space needs to encoded to use S3 parseUri function
    S3Uri s3Uri =
 s3Utilities.parseUri(URI.create(unloadResponse.getMetadataFile().replace(" ",
 "%20")));
    ResponseBytes<GetObjectResponse> objectBytes =
 s3Client.getObjectAsBytes(GetObjectRequest.builder()
            .bucket(s3Uri.bucket().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getMetadataFile(), "Invalid S3 URI")))
            .key(s3Uri.key().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getMetadataFile(), "Invalid S3 URI")))
            .build());
    String metadataFileContent = new String(objectBytes.asByteArray(),
 StandardCharsets.UTF_8);
    final Gson gson = new GsonBuilder()
            .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
            .create();
    return gson.fromJson(metadataFileContent, UnloadMetadata.class);
```

```
}

class UnloadMetadata {
    @JsonProperty("ColumnInfo")
    List<ColumnInfo> columnInfo;
    @JsonProperty("Author")
    Author author;

    @Data
    public class Author {
        @JsonProperty("Name")
        String name;
        @JsonProperty("MetadataFileVersion")
        String metadataFileVersion;
    }
}
```

Go

```
// Read and parse metadata content

func getMetadataFile(s3Svc *s3.S3, response map[string]string) Metadata {
    var metadataBuf = getObject(s3Svc, response["metadataFile"])
    var metadata Metadata
    json.Unmarshal(metadataBuf.Bytes(), &metadata)
    return metadata
}

func getObject(s3Svc *s3.S3, s3Uri string) *bytes.Buffer {
    u,_ := url.Parse(s3Uri)
    getObjectInput := &s3.GetObjectInput{
        Key:    aws.String(u.Path),
        Bucket: aws.String(u.Host),
    }
    getObjectOutput, err := s3Svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Println("Error: %s\n", err.Error())
    }
    buf := new(bytes.Buffer)
    buf.ReadFrom(getObjectOutput.Body)
    return buf
}
```

```
// Unload's Metadata structure

type Metadata struct {
    Author interface{}
    ColumnInfo []struct {
        Name string
        Type map[string]string
    }
}
```

Python

```
def __get_metadata_file(self, response):
    metadata = self.get_object(response['metadataFile']).read().decode('utf-8')
    parsed_metadata = json.loads(metadata)
   print("Metadata contents: \n%s" % parsed_metadata)

def get_object(self, uri):
    try:
        bucket, key = uri.replace("s3://", "").split("/", 1)
        s3_client = boto3.client('s3', region_name=<region>)
        response = s3_client.get_object(Bucket=bucket, Key=key)
        return response['Body']
    except Exception as err:
        print("Failed to get the object for URI:", uri)
        raise err
```

Node.js

```
// Read and parse metadata content
async getMetadataFile(response) {
    let metadata;
    await this.getS3Object(response['metadataFile']).then(
        (data) => {
            metadata = JSON.parse(data);
        }
    );
    return metadata;
}

async getS3Object(uri) {
```

```
    const {bucketName, key} = this.getBucketAndKey(uri);
    const params = new GetObjectCommand({
        Bucket: bucketName,
        Key: key
    })
    const response = await this.s3Client.send(params);
    return await response.Body.transformToString();
}

getBucketAndKey(uri) {
    const [bucketName] = uri.replace("s3://", "").split("/", 1);
    const key = uri.replace("s3://", "").split('/').slice(1).join('/');
    return {bucketName, key};
}
```

取消查询

您可以使用以下代码片段来取消查询。

> **Note**
>
> 这些代码片段基于上的完整示例应用程序。GitHub有关如何开始使用示例应用程序的更多信息，请参阅示例应用程序。

Java

```
    public void cancelQuery() {
        System.out.println("Starting query: " + SELECT_ALL_QUERY);
        QueryRequest queryRequest = new QueryRequest();
        queryRequest.setQueryString(SELECT_ALL_QUERY);
        QueryResult queryResult = queryClient.query(queryRequest);

        System.out.println("Cancelling the query: " + SELECT_ALL_QUERY);
        final CancelQueryRequest cancelQueryRequest = new CancelQueryRequest();
        cancelQueryRequest.setQueryId(queryResult.getQueryId());
        try {
            queryClient.cancelQuery(cancelQueryRequest);
            System.out.println("Query has been successfully cancelled");
        } catch (Exception e) {
```

```
            System.out.println("Could not cancel the query: " + SELECT_ALL_QUERY + "
 = " + e);
        }
    }
```

Java v2

```
    public void cancelQuery() {
        System.out.println("Starting query: " + SELECT_ALL_QUERY);
        QueryRequest queryRequest =
 QueryRequest.builder().queryString(SELECT_ALL_QUERY).build();
        QueryResponse queryResponse = timestreamQueryClient.query(queryRequest);

        System.out.println("Cancelling the query: " + SELECT_ALL_QUERY);
        final CancelQueryRequest cancelQueryRequest = CancelQueryRequest.builder()
                .queryId(queryResponse.queryId()).build();
        try {
            timestreamQueryClient.cancelQuery(cancelQueryRequest);
            System.out.println("Query has been successfully cancelled");
        } catch (Exception e) {
            System.out.println("Could not cancel the query: " + SELECT_ALL_QUERY + "
 = " + e);
        }
    }
```

Go

```
cancelQueryInput := &timestreamquery.CancelQueryInput{
      QueryId: aws.String(*queryOutput.QueryId),
  }

  fmt.Println("Submitting cancellation for the query")
  fmt.Println(cancelQueryInput)

  // submit the query
  cancelQueryOutput, err := querySvc.CancelQuery(cancelQueryInput)

  if err != nil {
      fmt.Println("Error:")
      fmt.Println(err)
  } else {
    fmt.Println("Query has been cancelled successfully")
    fmt.Println(cancelQueryOutput)
```

```
    }
```

Python

```
    def cancel_query(self):
        print("Starting query: " + self.SELECT_ALL)
        result = self.client.query(QueryString=self.SELECT_ALL)
        print("Cancelling query: " + self.SELECT_ALL)
        try:
            self.client.cancel_query(QueryId=result['QueryId'])
            print("Query has been successfully cancelled")
        except Exception as err:
            print("Cancelling query failed:", err)
```

Node.js

以下代码段使用 for JavaScript V2 样式。 AWS SDK它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function tryCancelQuery() {
    const params = {
        QueryString: SELECT_ALL_QUERY
    };
    console.log(`Running query: ${SELECT_ALL_QUERY}`);

    await queryClient.query(params).promise()
        .then(
            async (response) => {
                await cancelQuery(response.QueryId);
            },
            (err) => {
                console.error("Error while executing select all query:", err);
            });
}

async function cancelQuery(queryId) {
    const cancelParams = {
        QueryId: queryId
    };
    console.log(`Sending cancellation for query: ${SELECT_ALL_QUERY}`);
    await queryClient.cancelQuery(cancelParams).promise()
        .then(
            (response) => {
```

```
                console.log("Query has been cancelled successfully");
            },
            (err) => {
                console.error("Error while cancelling select all:", err);
            });
}
```

.NET

```
        public async Task CancelQuery()
        {
            Console.WriteLine("Starting query: " + SELECT_ALL_QUERY);
            QueryRequest queryRequest = new QueryRequest();
            queryRequest.QueryString = SELECT_ALL_QUERY;
            QueryResponse queryResponse = await
 queryClient.QueryAsync(queryRequest);

            Console.WriteLine("Cancelling query: " + SELECT_ALL_QUERY);
            CancelQueryRequest cancelQueryRequest = new CancelQueryRequest();
            cancelQueryRequest.QueryId = queryResponse.QueryId;

            try
            {
                await queryClient.CancelQueryAsync(cancelQueryRequest);
                Console.WriteLine("Query has been successfully cancelled.");
            } catch(Exception e)
            {
                Console.WriteLine("Could not cancel the query: " + SELECT_ALL_QUERY
 + " = " + e);
            }
        }
```

创建批量加载任务

您可以使用以下代码片段来创建批量加载任务。

Java

```
package com.example.tryit;

import java.util.Arrays;
```

```
import
 software.amazon.awssdk.services.timestreamwrite.model.CreateBatchLoadTaskRequest;
import
 software.amazon.awssdk.services.timestreamwrite.model.CreateBatchLoadTaskResponse;
import software.amazon.awssdk.services.timestreamwrite.model.DataModel;
import software.amazon.awssdk.services.timestreamwrite.model.DataModelConfiguration;
import
 software.amazon.awssdk.services.timestreamwrite.model.DataSourceConfiguration;
import
 software.amazon.awssdk.services.timestreamwrite.model.DataSourceS3Configuration;
import software.amazon.awssdk.services.timestreamwrite.model.DimensionMapping;
import
 software.amazon.awssdk.services.timestreamwrite.model.MultiMeasureAttributeMapping;
import software.amazon.awssdk.services.timestreamwrite.model.MultiMeasureMappings;
import software.amazon.awssdk.services.timestreamwrite.model.ReportConfiguration;
import software.amazon.awssdk.services.timestreamwrite.model.ReportS3Configuration;
import software.amazon.awssdk.services.timestreamwrite.model.ScalarMeasureValueType;
import software.amazon.awssdk.services.timestreamwrite.model.TimeUnit;
import software.amazon.awssdk.services.timestreamwrite.TimestreamWriteClient;

public class BatchLoadExample {
    public static final String DATABASE_NAME = <database name>;
    public static final String TABLE_NAME = <table name>;
    public static final String INPUT_BUCKET = <S3 location>;
    public static final String INPUT_OBJECT_KEY_PREFIX = <CSV filename>;
    public static final String REPORT_BUCKET = <S3 location>;
    public static final long HT_TTL_HOURS = 24L;
    public static final long CT_TTL_DAYS = 7L;

    TimestreamWriteClient amazonTimestreamWrite;

    public BatchLoadExample(TimestreamWriteClient client) {
        this.amazonTimestreamWrite = client;
    }

    public String createBatchLoadTask() {
        System.out.println("Creating batch load task");

        CreateBatchLoadTaskRequest request = CreateBatchLoadTaskRequest.builder()
                .dataModelConfiguration(DataModelConfiguration.builder()
                        .dataModel(DataModel.builder()
                                .timeColumn("timestamp")
                                .timeUnit(TimeUnit.SECONDS)
                                .dimensionMappings(Arrays.asList(
```

```
                                        DimensionMapping.builder()
                                                .sourceColumn("vehicle")
                                                .build(),
                                        DimensionMapping.builder()
                                                .sourceColumn("registration")
                                                .destinationColumn("license")
                                                .build()))
                                .multiMeasureMappings(MultiMeasureMappings.builder()
                                        .targetMultiMeasureName("mva_measure_name")

.multiMeasureAttributeMappings(Arrays.asList(

MultiMeasureAttributeMapping.builder()
                                                                .sourceColumn("wgt")

.targetMultiMeasureAttributeName("weight")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                                .build(),

MultiMeasureAttributeMapping.builder()
                                                                .sourceColumn("spd")

.targetMultiMeasureAttributeName("speed")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                                .build(),

MultiMeasureAttributeMapping.builder()
                                                                .sourceColumn("fuel")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                                .build(),

MultiMeasureAttributeMapping.builder()
                                                                .sourceColumn("miles")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                                .build()))
                                        .build())
                                .build())
                        .build())
                .dataSourceConfiguration(DataSourceConfiguration.builder()
                        .dataSourceS3Configuration(
```

```
                                DataSourceS3Configuration.builder()
                                        .bucketName(INPUT_BUCKET)
                                        .objectKeyPrefix(INPUT_OBJECT_KEY_PREFIX)
                                        .build())
                                .dataFormat("CSV")
                                .build())
                        .reportConfiguration(ReportConfiguration.builder()
                                .reportS3Configuration(ReportS3Configuration.builder()
                                        .bucketName(REPORT_BUCKET)
                                        .build())
                                .build())
                        .targetDatabaseName(DATABASE_NAME)
                        .targetTableName(TABLE_NAME)
                        .build();
                try {
                    final CreateBatchLoadTaskResponse createBatchLoadTaskResponse =
 amazonTimestreamWrite.createBatchLoadTask(request);
                    String taskId = createBatchLoadTaskResponse.taskId();
                    System.out.println("Successfully created batch load task: " + taskId);
                    return taskId;
                } catch (Exception e) {
                    System.out.println("Failed to create batch load task: " + e);
                    throw e;
                }
            }
        }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite/types"
)

func main() {
```

```
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{})(aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID &&  region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &  aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.CreateBatchLoadTask(context.TODO(), &
 timestreamwrite.CreateBatchLoadTaskInput{
            TargetDatabaseName: aws.String("BatchLoadExampleDatabase"),
            TargetTableName: aws.String("BatchLoadExampleTable"),
            RecordVersion: aws.Int64(1),
            DataModelConfiguration: &  types.DataModelConfiguration{
                DataModel: &  types.DataModel{
                    TimeColumn: aws.String("timestamp"),
                    TimeUnit: types.TimeUnitMilliseconds,
                    DimensionMappings: []types.DimensionMapping{
                        {
                            SourceColumn: aws.String("registration"),
                            DestinationColumn: aws.String("license"),
                        },
                    },
                    MultiMeasureMappings: &  types.MultiMeasureMappings{
                        TargetMultiMeasureName: aws.String("mva_measure_name"),
                        MultiMeasureAttributeMappings:
 []types.MultiMeasureAttributeMapping{
                            {
                                SourceColumn: aws.String("wgt"),
```

```
                                TargetMultiMeasureAttributeName:
 aws.String("weight"),
                                MeasureValueType:
 types.ScalarMeasureValueTypeDouble,
                            },
                            {
                                SourceColumn: aws.String("spd"),
                                TargetMultiMeasureAttributeName:
 aws.String("speed"),
                                MeasureValueType:
 types.ScalarMeasureValueTypeDouble,
                            },
                            {
                                SourceColumn: aws.String("fuel_consumption"),
                                TargetMultiMeasureAttributeName: aws.String("fuel"),
                                MeasureValueType:
 types.ScalarMeasureValueTypeDouble,
                            },
                        },
                    },
                },
            },
           },
           DataSourceConfiguration: &  types.DataSourceConfiguration{
                DataSourceS3Configuration: &  types.DataSourceS3Configuration{
                    BucketName: aws.String("test-batch-load-west-2"),
                    ObjectKeyPrefix: aws.String("sample.csv"),
                },
                DataFormat: types.BatchLoadDataFormatCsv,
           },
           ReportConfiguration: & types.ReportConfiguration{
                ReportS3Configuration: & types.ReportS3Configuration{
                    BucketName: aws.String("test-batch-load-report-west-2"),
                    EncryptionOption: types.S3EncryptionOptionSseS3,
                },
           },
 })

 fmt.Println(aws.ToString(response.TaskId))
}
```

Python

```
import boto3
```

```
from botocore.config import Config

INGEST_ENDPOINT = "<URL>"
REGION = "us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
DATABASE_NAME = "<database name>"
TABLE_NAME = "<table name>"
INPUT_BUCKET_NAME = "<S3 location>"
INPUT_OBJECT_KEY_PREFIX = "<CSV file name>"
REPORT_BUCKET_NAME = "<S3 location>"


def create_batch_load_task(client, database_name, table_name, input_bucket_name,
 input_object_key_prefix, report_bucket_name):
    try:
        result = client.create_batch_load_task(TargetDatabaseName=database_name,
 TargetTableName=table_name,
                                               DataModelConfiguration={"DataModel":
 {
                                                   "TimeColumn": "timestamp",
                                                   "TimeUnit": "SECONDS",
                                                   "DimensionMappings": [
                                                       {
                                                           "SourceColumn": "vehicle"
                                                       },
                                                       {
                                                           "SourceColumn":
 "registration",
                                                           "DestinationColumn":
 "license"
                                                       }
                                                   ],
                                                   "MultiMeasureMappings": {
                                                       "TargetMultiMeasureName":
 "metrics",

 "MultiMeasureAttributeMappings": [
                                                           {
                                                               "SourceColumn":
 "wgt",
                                                               "MeasureValueType":
 "DOUBLE"
                                                           },
```

```
                                                        {
                                                            "SourceColumn":
"spd",
                                                            "MeasureValueType":
"DOUBLE"
                                                        },
                                                        {
                                                            "SourceColumn":
"fuel_consumption",

"TargetMultiMeasureAttributeName": "fuel",
                                                            "MeasureValueType":
"DOUBLE"
                                                        },
                                                        {
                                                            "SourceColumn":
"miles",
                                                            "MeasureValueType":
"DOUBLE"
                                                        }
                                                    ]}
                                            }
                                            },
                                            DataSourceConfiguration={
                                                "DataSourceS3Configuration": {
                                                    "BucketName":
input_bucket_name,
                                                    "ObjectKeyPrefix":
input_object_key_prefix
                                                },
                                                "DataFormat": "CSV"
                                            },
                                            ReportConfiguration={
                                                "ReportS3Configuration": {
                                                    "BucketName":
report_bucket_name,
                                                    "EncryptionOption": "SSE_S3"
                                                }
                                            }
                                            )

        task_id = result["TaskId"]
        print("Successfully created batch load task: ", task_id)
        return task_id
```

```
    except Exception as err:
        print("Create batch load task job failed:", err)
        return None


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write',
                                  endpoint_url=INGEST_ENDPOINT, region_name=REGION,
                                  config=Config(read_timeout=20,
 max_pool_connections=5000, retries={'max_attempts': 10}))

    task_id = create_batch_load_task(write_client, DATABASE_NAME, TABLE_NAME,
                                     INPUT_BUCKET_NAME, INPUT_OBJECT_KEY_PREFIX,
 REPORT_BUCKET_NAME)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

有关API详细信息，请参阅类 CreateBatchLoadCommand和CreateBatchLoadTask。

```
import { TimestreamWriteClient, CreateBatchLoadTaskCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-west-2", endpoint:
 "https://gamma-ingest-cell3.timestream.us-west-2.amazonaws.com" });

const params = {
    TargetDatabaseName: "BatchLoadExampleDatabase",
    TargetTableName: "BatchLoadExampleTable",
 RecordVersion: 1,
 DataModelConfiguration: {
  DataModel: {
   TimeColumn: "timestamp",
            TimeUnit: "MILLISECONDS",
            DimensionMappings: [
                {
                    SourceColumn: "registration",
                    DestinationColumn: "license"
                }
            ],
            MultiMeasureMappings: {
```

```
                TargetMultiMeasureName: "mva_measure_name",
                MultiMeasureAttributeMappings: [
                    {
                        SourceColumn: "wgt",
                        TargetMultiMeasureAttributeName: "weight",
                        MeasureValueType: "DOUBLE"
                    },
                    {
                        SourceColumn: "spd",
                        TargetMultiMeasureAttributeName: "speed",
                        MeasureValueType: "DOUBLE"
                    },
                    {
                        SourceColumn: "fuel_consumption",
                        TargetMultiMeasureAttributeName: "fuel",
                        MeasureValueType: "DOUBLE"
                    }
                ]
            }
        }
    },
 DataSourceConfiguration: {
        DataSourceS3Configuration: {
            BucketName: "test-batch-load-west-2",
            ObjectKeyPrefix: "sample.csv"
        },
        DataFormat: "CSV"
    },
    ReportConfiguration: {
        ReportS3Configuration: {
            BucketName: "test-batch-load-report-west-2",
            EncryptionOption: "SSE_S3"
        }
    }
};

const command = new CreateBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Created batch load task ` + data.TaskId);
} catch (error) {
 console.log("Error creating table. ", error);
 throw error;
```

```
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class CreateBatchLoadTaskExample
    {
        public const string DATABASE_NAME = "<database name>";
        public const string TABLE_NAME = "<table name>";
        public const string INPUT_BUCKET = "<input bucket name>";
        public const string INPUT_OBJECT_KEY_PREFIX = "<CSV file name>";
        public const string REPORT_BUCKET = "<report bucket name>";
        public const long HT_TTL_HOURS = 24L;
        public const long CT_TTL_DAYS = 7L;
        private readonly AmazonTimestreamWriteClient writeClient;

        public CreateBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task CreateBatchLoadTask()
        {
            try
            {
                var createBatchLoadTaskRequest = new CreateBatchLoadTaskRequest
                {
                    DataModelConfiguration = new DataModelConfiguration
                    {
                        DataModel = new DataModel
                        {
                            TimeColumn = "timestamp",
                            TimeUnit = TimeUnit.SECONDS,
                            DimensionMappings = new List<DimensionMapping>()
                            {
```

```
                                new()
                                {
                                        SourceColumn = "vehicle"
                                },
                                new()
                                {
                                        SourceColumn = "registration",
                                        DestinationColumn = "license"
                                }
                            },
                            MultiMeasureMappings = new MultiMeasureMappings
                            {
                                TargetMultiMeasureName = "mva_measure_name",
                                MultiMeasureAttributeMappings = new
List<MultiMeasureAttributeMapping>()
                                {
                                        new()
                                        {
                                                SourceColumn = "wgt",
                                                TargetMultiMeasureAttributeName =
"weight",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
                                                SourceColumn = "spd",
                                                TargetMultiMeasureAttributeName =
"speed",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
                                                SourceColumn = "fuel",
                                                TargetMultiMeasureAttributeName =
"fuel",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
                                                SourceColumn = "miles",
```

```
                                                TargetMultiMeasureAttributeName =
 "miles",
                                                MeasureValueType =
 ScalarMeasureValueType.DOUBLE
                                            }
                                    }
                                }
                            }
                        },
                        DataSourceConfiguration = new DataSourceConfiguration
                        {
                            DataSourceS3Configuration = new DataSourceS3Configuration
                            {
                                BucketName = INPUT_BUCKET,
                                ObjectKeyPrefix = INPUT_OBJECT_KEY_PREFIX
                            },
                            DataFormat = "CSV"
                        },
                        ReportConfiguration = new ReportConfiguration
                        {
                            ReportS3Configuration = new ReportS3Configuration
                            {
                                BucketName = REPORT_BUCKET
                            }
                        },
                        TargetDatabaseName = DATABASE_NAME,
                        TargetTableName = TABLE_NAME
                    };

                    CreateBatchLoadTaskResponse response = await
 writeClient.CreateBatchLoadTaskAsync(createBatchLoadTaskRequest);
                    Console.WriteLine($"Task created: " + response.TaskId);
                }
                catch (Exception e)
                {
                    Console.WriteLine("Create batch load task failed:" + e.ToString());
                }
            }
        }
    }
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
```

```
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new CreateBatchLoadTaskExample(writeClient);
            await example.CreateBatchLoadTask();
        }
    }
}
```

描述批量加载任务

您可以使用以下代码片段来描述批量加载任务。

Java

```
    public void describeBatchLoadTask(String taskId) {
            final DescribeBatchLoadTaskResponse batchLoadTaskResponse =
 amazonTimestreamWrite

 .describeBatchLoadTask(DescribeBatchLoadTaskRequest.builder()
                                             .taskId(taskId)
                                             .build());

            System.out.println("Task id: " +
 batchLoadTaskResponse.batchLoadTaskDescription().taskId());
            System.out.println("Status: " +
 batchLoadTaskResponse.batchLoadTaskDescription().taskStatusAsString());
            System.out.println("Records processed: "
                               +
 batchLoadTaskResponse.batchLoadTaskDescription().progressReport().recordsProcessed());
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
```

```
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.DescribeBatchLoadTask(context.TODO(),
 &timestreamwrite.DescribeBatchLoadTaskInput{
  TaskId: aws.String("<TaskId>"),
 })

 fmt.Println(aws.ToString(response.BatchLoadTaskDescription.TaskId))
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT="<url>"
REGION="us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
TASK_ID = "<task id>"

def describe_batch_load_task(client, task_id):
    try:
        result = client.describe_batch_load_task(TaskId=task_id)
        print("Successfully described batch load task: ", result)
    except Exception as err:
        print("Describe batch load task job failed:", err)
```

```
if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write', \
        endpoint_url=INGEST_ENDPOINT, region_name=REGION, \
        config=Config(read_timeout=20, max_pool_connections = 5000,
 retries={'max_attempts': 10}))

    describe_batch_load_task(write_client, TASK_ID)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

有关API详细信息，请参阅类 DescribeBatchLoadCommand和DescribeBatchLoadTask。

```
import { TimestreamWriteClient, DescribeBatchLoadTaskCommand } from "@aws-sdk/
client-timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    TaskId: "<TaskId>"
};

const command = new DescribeBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Batch load task has id ` + data.BatchLoadTaskDescription.TaskId);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Batch load task doesn't exist.");
    } else {
        console.log("Describe batch load task failed.", error);
        throw error;
    }
}
```

.NET

```
using System;
```

```
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class DescribeBatchLoadTaskExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public DescribeBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task DescribeBatchLoadTask(String taskId)
        {
            try
            {
                var describeBatchLoadTaskRequest = new DescribeBatchLoadTaskRequest
                {
                    TaskId = taskId
                };
                DescribeBatchLoadTaskResponse response = await
 writeClient.DescribeBatchLoadTaskAsync(describeBatchLoadTaskRequest);
                Console.WriteLine($"Task has id:
{response.BatchLoadTaskDescription.TaskId}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Batch load task does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe batch load task failed:" +
 e.ToString());
            }
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new DescribeBatchLoadTaskExample(writeClient);
            await example.DescribeBatchLoadTask("<batch load task id>");
        }
    }
}
```

列出批量加载任务

您可以使用以下代码片段列出批量加载任务。

Java

```
    public void listBatchLoadTasks() {
            final ListBatchLoadTasksResponse listBatchLoadTasksResponse =
 amazonTimestreamWrite
                            .listBatchLoadTasks(ListBatchLoadTasksRequest.builder()
                                                .maxResults(15)
                                                .build());

            for (BatchLoadTask batchLoadTask :
 listBatchLoadTasksResponse.batchLoadTasks()) {
                    System.out.println(batchLoadTask.taskId());
            }
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
```

```
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)
 listBatchLoadTasksMaxResult := int32(15)

 response, err := client.ListBatchLoadTasks(context.TODO(),
 &timestreamwrite.ListBatchLoadTasksInput{
  MaxResults: &listBatchLoadTasksMaxResult,
 })

 for i, task := range response.BatchLoadTasks {
  fmt.Println(i, aws.ToString(task.TaskId))
 }
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT = "<url>"
REGION = "us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7


def print_batch_load_tasks(batch_load_tasks):
    for batch_load_task in batch_load_tasks:
        print(batch_load_task['TaskId'])


def list_batch_load_tasks(client):
    print("\nListing batch load tasks")
    try:
        response = client.list_batch_load_tasks(MaxResults=10)
```

```
        print_batch_load_tasks(response['BatchLoadTasks'])
        next_token = response.get('NextToken', None)
        while next_token:
            response = client.list_batch_load_tasks(
                NextToken=next_token, MaxResults=10)
            print_batch_load_tasks(response['BatchLoadTasks'])
            next_token = response.get('NextToken', None)
    except Exception as err:
        print("List batch load tasks failed:", err)
        raise err


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write',
                                  endpoint_url=INGEST_ENDPOINT, region_name=REGION,
                                  config=Config(read_timeout=20,
 max_pool_connections=5000, retries={'max_attempts': 10}))

    list_batch_load_tasks(write_client)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

有关API详细信息，请参阅类 DescribeBatchLoadCommand和DescribeBatchLoadTask。

```
import { TimestreamWriteClient, ListBatchLoadTasksCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    MaxResults: <15>
};

const command = new ListBatchLoadTasksCommand(params);

getBatchLoadTasksList(null);

async function getBatchLoadTasksList(nextToken) {
    if (nextToken) {
```

```
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.BatchLoadTasks.forEach(function (task) {
            console.log(task.TaskId);
        });

        if (data.NextToken) {
            return getBatchLoadTasksList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing batch load tasks", error);
    }
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class ListBatchLoadTasksExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public ListBatchLoadTasksExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task ListBatchLoadTasks()
        {
            Console.WriteLine("Listing batch load tasks");

            try
```

```
            {
                var listBatchLoadTasksRequest = new ListBatchLoadTasksRequest
                {
                    MaxResults = 15
                };

                ListBatchLoadTasksResponse response = await
 writeClient.ListBatchLoadTasksAsync(listBatchLoadTasksRequest);

                PrintBatchLoadTasks(response.BatchLoadTasks);
                var nextToken = response.NextToken;

                while (nextToken != null)
                {
                    listBatchLoadTasksRequest.NextToken = nextToken;
                    response = await
 writeClient.ListBatchLoadTasksAsync(listBatchLoadTasksRequest);
                    PrintBatchLoadTasks(response.BatchLoadTasks);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List batch load tasks failed:" + e.ToString());
            }
        }

        private void PrintBatchLoadTasks(List<BatchLoadTask> tasks)
        {
            foreach (BatchLoadTask task in tasks)
                Console.WriteLine($"Task:{task.TaskId}");
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
```

```
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new ListBatchLoadTasksExample(writeClient);
            await example.ListBatchLoadTasks();
        }
    }
}
```

恢复批量加载任务

您可以使用以下代码片段来恢复批量加载任务。

Java

```
    public void resumeBatchLoadTask(String taskId) {
```

```
            try {
                    amazonTimestreamWrite

 .resumeBatchLoadTask(ResumeBatchLoadTaskRequest.builder()
                                                    .taskId(taskId)
                                                    .build());

                    System.out.println("Successfully resumed batch load task.");
            } catch (ValidationException validationException) {
                    System.out.println(validationException.getMessage());
            }
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))
```

```
 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.ResumeBatchLoadTask(context.TODO(),
 &timestreamwrite.ResumeBatchLoadTaskInput{
  TaskId: aws.String("TaskId"),
 })

 if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
 } else {
  fmt.Println("Resume batch load task is successful")
  fmt.Println(response)
 }
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT="<url>"
REGION="us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
TASK_ID = "<TaskId>"

def resume_batch_load_task(client, task_id):
    try:
        result = client.resume_batch_load_task(TaskId=task_id)
        print("Successfully resumed batch load task: ", result)
    except Exception as err:
        print("Resume batch load task failed:", err)


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write', \
```

```
        endpoint_url=INGEST_ENDPOINT, region_name=REGION, \
        config=Config(read_timeout=20, max_pool_connections = 5000,
 retries={'max_attempts': 10}))

    resume_batch_load_task(write_client, TASK_ID)
```

Node.js

以下代码段用AWSSDK于 JavaScript v3。有关如何安装客户端和用法的更多信息，请参阅 Timestream Write Client-f AWS SDK or JavaScript v3。

有关API详细信息，请参阅类 CreateBatchLoadCommand和CreateBatchLoadTask。

```
import { TimestreamWriteClient, ResumeBatchLoadTaskCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    TaskId: "<TaskId>"
};

const command = new ResumeBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Resumed batch load task");
} catch (error) {
    console.log("Resume batch load task failed.", error);
 throw error;
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
```

```
    public class ResumeBatchLoadTaskExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public ResumeBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task ResumeBatchLoadTask(String taskId)
        {
            try
            {
                var resumeBatchLoadTaskRequest = new ResumeBatchLoadTaskRequest
                {
                    TaskId = taskId
                };
                ResumeBatchLoadTaskResponse response = await
 writeClient.ResumeBatchLoadTaskAsync(resumeBatchLoadTaskRequest);
                Console.WriteLine("Successfully resumed batch load task.");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Batch load task does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Resume batch load task failed: " + e.ToString());
            }
        }
    }
}
```

创建计划查询

您可以使用以下代码片段创建具有多度量映射的计划查询。

Java

```
public static String DATABASE_NAME = "devops_sample_application";
public static String TABLE_NAME = "host_metrics_sample_application";
public static String HOSTNAME = "host-24Gju";
```

```
public static String SQ_NAME = "daily-sample";
public static String SCHEDULE_EXPRESSION = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public static String QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
"ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
"FROM " +  DATABASE_NAME + "." +  TABLE_NAME + " " +
"WHERE measure_name = 'metrics' " +
"AND hostname = '" + HOSTNAME + "' " +
"AND time > ago(2h) " +
"GROUP BY region, hostname, az, BIN(time, 15s) " +
"ORDER BY binned_timestamp ASC " +
"LIMIT 5";


public String createScheduledQuery(String topic_arn,
    String role_arn,
    String database_name,
    String table_name) {
    System.out.println("Creating Scheduled Query");

    List<Pair<String, MeasureValueType>> sourceColToMeasureValueTypes =
 Arrays.asList(
        Pair.of("avg_cpu_utilization", DOUBLE),
        Pair.of("p90_cpu_utilization", DOUBLE),
        Pair.of("p95_cpu_utilization", DOUBLE),
        Pair.of("p99_cpu_utilization", DOUBLE));

    CreateScheduledQueryRequest createScheduledQueryRequest = new
 CreateScheduledQueryRequest()
            .withName(SQ_NAME)
            .withQueryString(QUERY)
            .withScheduleConfiguration(new ScheduleConfiguration()
                    .withScheduleExpression(SCHEDULE_EXPRESSION))
            .withNotificationConfiguration(new NotificationConfiguration()
                    .withSnsConfiguration(new SnsConfiguration()
                            .withTopicArn(topic_arn)))
            .withTargetConfiguration(new
 TargetConfiguration().withTimestreamConfiguration(new TimestreamConfiguration()
```

```
                    .withDatabaseName(database_name)
                    .withTableName(table_name)
                    .withTimeColumn("binned_timestamp")
                    .withDimensionMappings(Arrays.asList(
                            new DimensionMapping()
                                    .withName("region")
                                    .withDimensionValueType("VARCHAR"),
                            new DimensionMapping()
                                    .withName("az")
                                    .withDimensionValueType("VARCHAR"),
                            new DimensionMapping()
                                    .withName("hostname")
                                    .withDimensionValueType("VARCHAR")
                    ))
                    .withMultiMeasureMappings(new MultiMeasureMappings()
                        .withTargetMultiMeasureName("multi-metrics")
                        .withMultiMeasureAttributeMappings(
                            sourceColToMeasureValueTypes.stream()
                            .map(pair -> new MultiMeasureAttributeMapping()
                                .withMeasureValueType(pair.getValue().name())
                                .withSourceColumn(pair.getKey()))
                            .collect(Collectors.toList()))))
            .withErrorReportConfiguration(new ErrorReportConfiguration()
                    .withS3Configuration(new S3Configuration()

 .withBucketName(timestreamDependencyHelper.getS3ErrorReportBucketName())))
            .withScheduledQueryExecutionRoleArn(role_arn);

    try {
        final CreateScheduledQueryResult createScheduledQueryResult =
 queryClient.createScheduledQuery(createScheduledQueryRequest);
        final String scheduledQueryArn = createScheduledQueryResult.getArn();
        System.out.println("Successfully created scheduled query : " +
 scheduledQueryArn);
        return scheduledQueryArn;
    }
    catch (Exception e) {
        System.out.println("Scheduled Query creation failed: " + e);
        throw e;
    }
}
```

Java v2

```
public static String DATABASE_NAME = "testJavaV2DB";
public static String TABLE_NAME = "testJavaV2Table";
public static String HOSTNAME = "host-24Gju";
public static String SQ_NAME = "daily-sample";
public static String SCHEDULE_EXPRESSION = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public static String VALID_QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
"ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
"FROM " +  DATABASE_NAME + "." +  TABLE_NAME + " " +
"WHERE measure_name = 'metrics' " +
"AND hostname = '" + HOSTNAME + "' " +
"AND time > ago(2h) " +
"GROUP BY region, hostname, az, BIN(time, 15s) " +
"ORDER BY binned_timestamp ASC " +
"LIMIT 5";


private String createScheduledQueryHelper(String topicArn, String roleArn,
        String s3ErrorReportBucketName, String query,
        TargetConfiguration targetConfiguration) {
    System.out.println("Creating Scheduled Query");

    CreateScheduledQueryRequest createScheduledQueryRequest =
 CreateScheduledQueryRequest.builder()
            .name(SQ_NAME)
            .queryString(query)
            .scheduleConfiguration(ScheduleConfiguration.builder()
                    .scheduleExpression(SCHEDULE_EXPRESSION)
                    .build())
            .notificationConfiguration(NotificationConfiguration.builder()
                    .snsConfiguration(SnsConfiguration.builder()
                            .topicArn(topicArn)
                            .build())
                    .build())
            .targetConfiguration(targetConfiguration)
            .errorReportConfiguration(ErrorReportConfiguration.builder()
```

```
                    .s3Configuration(S3Configuration.builder()
                            .bucketName(s3ErrorReportBucketName)
                            .objectKeyPrefix(SCHEDULED_QUERY_EXAMPLE)
                            .build())
                    .build())
            .scheduledQueryExecutionRoleArn(roleArn)
            .build();

    try {
        final CreateScheduledQueryResponse response =
 queryClient.createScheduledQuery(createScheduledQueryRequest);
        final String scheduledQueryArn = response.arn();
        System.out.println("Successfully created scheduled query : " +
 scheduledQueryArn);
        return scheduledQueryArn;
    }
    catch (Exception e) {
        System.out.println("Scheduled Query creation failed: " + e);
        throw e;
    }
}

public String createScheduledQuery(String topicArn, String roleArn,
        String databaseName, String tableName, String s3ErrorReportBucketName) {
    List<Pair<String, MeasureValueType>> sourceColToMeasureValueTypes =
 Arrays.asList(
            Pair.of("avg_cpu_utilization", DOUBLE),
            Pair.of("p90_cpu_utilization", DOUBLE),
            Pair.of("p95_cpu_utilization", DOUBLE),
            Pair.of("p99_cpu_utilization", DOUBLE));

    TargetConfiguration targetConfiguration = TargetConfiguration.builder()
            .timestreamConfiguration(TimestreamConfiguration.builder()
            .databaseName(databaseName)
            .tableName(tableName)
            .timeColumn("binned_timestamp")
            .dimensionMappings(Arrays.asList(
                    DimensionMapping.builder()
                            .name("region")
                            .dimensionValueType("VARCHAR")
                            .build(),
                    DimensionMapping.builder()
                            .name("az")
                            .dimensionValueType("VARCHAR")
```

```
                            .build(),
                    DimensionMapping.builder()
                            .name("hostname")
                            .dimensionValueType("VARCHAR")
                            .build()
            ))
            .multiMeasureMappings(MultiMeasureMappings.builder()
                    .targetMultiMeasureName("multi-metrics")
                    .multiMeasureAttributeMappings(
                            sourceColToMeasureValueTypes.stream()
                                    .map(pair ->
 MultiMeasureAttributeMapping.builder()

 .measureValueType(pair.getValue().name())
                                            .sourceColumn(pair.getKey())
                                            .build())
                                    .collect(Collectors.toList()))
                    .build())
            .build())
            .build();

    return createScheduledQueryHelper(topicArn, roleArn, s3ErrorReportBucketName,
 VALID_QUERY, targetConfiguration);
}}
```

Go

```
SQ_ERROR_CONFIGURATION_S3_BUCKET_NAME_PREFIX = "sq-error-configuration-sample-s3-
bucket-"
HOSTNAME            = "host-24Gju"
SQ_NAME             = "daily-sample"
SCHEDULE_EXPRESSION = "cron(0/1 * * * ? *)"
QUERY               = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
    "ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
    "FROM %s.%s " +
    "WHERE measure_name = 'metrics' " +
    "AND hostname = '" + HOSTNAME + "' " +
    "AND time > ago(2h) " +
    "GROUP BY region, hostname, az, BIN(time, 15s) " +
```

```
    "ORDER BY binned_timestamp ASC " +
    "LIMIT 5"
s3BucketName = utils.SQ_ERROR_CONFIGURATION_S3_BUCKET_NAME_PREFIX +
 generateRandomStringWithSize(5)

func generateRandomStringWithSize(size int) string {
     rand.Seed(time.Now().UnixNano())
     alphaNumericList := []rune("abcdefghijklmnopqrstuvwxyz0123456789")
     randomPrefix := make([]rune, size)
     for i := range randomPrefix {
         randomPrefix[i] = alphaNumericList[rand.Intn(len(alphaNumericList))]
     }
     return string(randomPrefix)
 }

func (timestreamBuilder TimestreamBuilder) createScheduledQuery(topicArn string,
 roleArn string, s3ErrorReportBucketName string,
query string, targetConfiguration timestreamquery.TargetConfiguration) (string,
 error) {

createScheduledQueryInput := &timestreamquery.CreateScheduledQueryInput{
    Name:        aws.String(SQ_NAME),
    QueryString: aws.String(query),
    ScheduleConfiguration: &timestreamquery.ScheduleConfiguration{
        ScheduleExpression: aws.String(SCHEDULE_EXPRESSION),
    },
    NotificationConfiguration: &timestreamquery.NotificationConfiguration{
        SnsConfiguration: &timestreamquery.SnsConfiguration{
            TopicArn: aws.String(topicArn),
        },
    },
    TargetConfiguration: &targetConfiguration,
    ErrorReportConfiguration: &timestreamquery.ErrorReportConfiguration{
        S3Configuration: &timestreamquery.S3Configuration{
            BucketName: aws.String(s3ErrorReportBucketName),
        },
    },
    ScheduledQueryExecutionRoleArn: aws.String(roleArn),
}

createScheduledQueryOutput, err :=
 timestreamBuilder.QuerySvc.CreateScheduledQuery(createScheduledQueryInput)

if err != nil {
```

```
    fmt.Printf("Error: %s", err.Error())
} else {
    fmt.Println("createScheduledQueryResult is successful")
    return *createScheduledQueryOutput.Arn, nil
}
return "", err
}

 func (timestreamBuilder TimestreamBuilder) CreateValidScheduledQuery(topicArn
 string, roleArn string, s3ErrorReportBucketName string,
     sqDatabaseName string, sqTableName string, databaseName string, tableName
 string) (string, error) {

     targetConfiguration := timestreamquery.TargetConfiguration{
         TimestreamConfiguration: &timestreamquery.TimestreamConfiguration{
             DatabaseName: aws.String(sqDatabaseName),
             TableName:    aws.String(sqTableName),
             TimeColumn:   aws.String("binned_timestamp"),
             DimensionMappings: []*timestreamquery.DimensionMapping{
                 {
                     Name:               aws.String("region"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
                 {
                     Name:               aws.String("az"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
                 {
                     Name:               aws.String("hostname"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
             },
             MultiMeasureMappings: &timestreamquery.MultiMeasureMappings{
                 TargetMultiMeasureName: aws.String("multi-metrics"),
                 MultiMeasureAttributeMappings:
 []*timestreamquery.MultiMeasureAttributeMapping{
                     {
                         SourceColumn:     aws.String("avg_cpu_utilization"),
                         MeasureValueType:
 aws.String(timestreamquery.MeasureValueTypeDouble),
                     },
                     {
                         SourceColumn:     aws.String("p90_cpu_utilization"),
```

```
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                    {
                        SourceColumn:     aws.String("p95_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                    {
                        SourceColumn:     aws.String("p99_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                },
            },
        },
    }
    return timestreamBuilder.createScheduledQuery(topicArn, roleArn,
s3ErrorReportBucketName,
        fmt.Sprintf(QUERY, databaseName, tableName), targetConfiguration)
}
```

Python

```
HOSTNAME = "host-24Gju"
SQ_NAME = "daily-sample"
ERROR_BUCKET_NAME = "scheduledquerysamplerrorbucket" +
 ''.join([choice(ascii_lowercase) for _ in range(5)])
QUERY = \
    "SELECT region, az, hostname, BIN(time, 15s) AS binned_timestamp, " \
    "    ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, "
 \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization,
 " \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization "
 \
    "FROM " + database_name + "." + table_name + " " \
    "WHERE measure_name = 'metrics' " \
    "AND hostname = '" + self.HOSTNAME + "' " \
    "AND time > ago(2h) " \
    "GROUP BY region, hostname, az, BIN(time, 15s) " \
    "ORDER BY binned_timestamp ASC " \
```

```
    "LIMIT 5"

def create_scheduled_query_helper(self, topic_arn, role_arn, query,
 target_configuration):
    print("\nCreating Scheduled Query")
    schedule_configuration = {
        'ScheduleExpression': 'cron(0/2 * * * ? *)'
    }
    notification_configuration = {
        'SnsConfiguration': {
            'TopicArn': topic_arn
        }
    }
    error_report_configuration = {
        'S3Configuration': {
            'BucketName': ERROR_BUCKET_NAME
        }
    }

    try:
        create_scheduled_query_response = \
            query_client.create_scheduled_query(Name=self.SQ_NAME,
                 QueryString=query,
                 ScheduleConfiguration=schedule_configuration,
                 NotificationConfiguration=notification_configuration,
                 TargetConfiguration=target_configuration,
                 ScheduledQueryExecutionRoleArn=role_arn,
                 ErrorReportConfiguration=error_report_configuration
                 )
        print("Successfully created scheduled query : ",
 create_scheduled_query_response['Arn'])
        return create_scheduled_query_response['Arn']
    except Exception as err:
        print("Scheduled Query creation failed:", err)
        raise err

def create_valid_scheduled_query(self, topic_arn, role_arn):
    target_configuration = {
        'TimestreamConfiguration': {
            'DatabaseName': self.sq_database_name,
            'TableName': self.sq_table_name,
            'TimeColumn': 'binned_timestamp',
            'DimensionMappings': [
                {'Name': 'region', 'DimensionValueType': 'VARCHAR'},
```

```
                {'Name': 'az', 'DimensionValueType': 'VARCHAR'},
                {'Name': 'hostname', 'DimensionValueType': 'VARCHAR'}
            ],
            'MultiMeasureMappings': {
                'TargetMultiMeasureName': 'target_name',
                'MultiMeasureAttributeMappings': [
                    {'SourceColumn': 'avg_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'avg_cpu_utilization'},
                    {'SourceColumn': 'p90_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p90_cpu_utilization'},
                    {'SourceColumn': 'p95_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p95_cpu_utilization'},
                    {'SourceColumn': 'p99_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p99_cpu_utilization'},
                ]
            }
        }
    }

    return self.create_scheduled_query_helper(topic_arn, role_arn, QUERY,
 target_configuration)
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
const DATABASE_NAME = 'devops_sample_application';
const TABLE_NAME = 'host_metrics_sample_application';
const SQ_DATABASE_NAME = 'sq_result_database';
const SQ_TABLE_NAME = 'sq_result_table';
const HOSTNAME = "host-24Gju";
const SQ_NAME = "daily-sample";
const SCHEDULE_EXPRESSION = "cron(0/1 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
const VALID_QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
```

```
    " ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
    "FROM " + DATABASE_NAME + "." + TABLE_NAME + " " +
    "WHERE measure_name = 'metrics' " +
    " AND hostname = '" + HOSTNAME + "' " +
    " AND time > ago(2h) " +
    "GROUP BY region, hostname, az, BIN(time, 15s) " +
    "ORDER BY binned_timestamp ASC " +
    "LIMIT 5";

async function createScheduledQuery(topicArn, roleArn, s3ErrorReportBucketName) {
    console.log("Creating Valid Scheduled Query");
    const DimensionMappingList = [{
            'Name': 'region',
            'DimensionValueType': 'VARCHAR'
        },
        {
            'Name': 'az',
            'DimensionValueType': 'VARCHAR'
        },
        {
            'Name': 'hostname',
            'DimensionValueType': 'VARCHAR'
        }
    ];

    const MultiMeasureMappings = {
        TargetMultiMeasureName: "multi-metrics",
        MultiMeasureAttributeMappings: [{
                'SourceColumn': 'avg_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p90_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p95_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p99_cpu_utilization',
```

```
                'MeasureValueType': 'DOUBLE'
            },
        ]
    }

    const timestreamConfiguration = {
        DatabaseName: SQ_DATABASE_NAME,
        TableName: SQ_TABLE_NAME,
        TimeColumn: "binned_timestamp",
        DimensionMappings: DimensionMappingList,
        MultiMeasureMappings: MultiMeasureMappings
    }

    const createScheduledQueryRequest = {
        Name: SQ_NAME,
        QueryString: VALID_QUERY,
        ScheduleConfiguration: {
            ScheduleExpression: SCHEDULE_EXPRESSION
        },
        NotificationConfiguration: {
            SnsConfiguration: {
                TopicArn: topicArn
            }
        },
        TargetConfiguration: {
            TimestreamConfiguration: timestreamConfiguration
        },
        ScheduledQueryExecutionRoleArn: roleArn,
        ErrorReportConfiguration: {
            S3Configuration: {
                BucketName: s3ErrorReportBucketName
            }
        }
    };
    try {
        const data = await
queryClient.createScheduledQuery(createScheduledQueryRequest).promise();
        console.log("Successfully created scheduled query: " + data.Arn);
        return data.Arn;
    } catch (err) {
        console.log("Scheduled Query creation failed: ", err);
        throw err;
    }
```

```
}
```

.NET

```
public const string Hostname = "host-24Gju";
public const string SqName = "timestream-sample";
public const string SqDatabaseName = "sq_result_database";
public const string SqTableName = "sq_result_table";

public const string ErrorConfigurationS3BucketNamePrefix = "error-configuration-
sample-s3-bucket-";
public const string ScheduleExpression = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public const string ValidQuery = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
      "ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, "
 +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
      "FROM " + Constants.DATABASE_NAME + "." + Constants.TABLE_NAME + " " +
      "WHERE measure_name = 'metrics' " +
      "AND hostname = '" + Hostname + "' " +
      "AND time > ago(2h) " +
      "GROUP BY region, hostname, az, BIN(time, 15s) " +
      "ORDER BY binned_timestamp ASC " +
      "LIMIT 5";

private async Task<String> CreateValidScheduledQuery(string topicArn, string
 roleArn,
             string databaseName, string tableName, string s3ErrorReportBucketName)
 {
     List<MultiMeasureAttributeMapping> sourceColToMeasureValueTypes =
         new List<MultiMeasureAttributeMapping>()
         {
             new()
             {
                 SourceColumn = "avg_cpu_utilization",
                 MeasureValueType = MeasureValueType.DOUBLE.Value
             },
             new()
```

```
            {
                SourceColumn = "p90_cpu_utilization",
                MeasureValueType = MeasureValueType.DOUBLE.Value
            },
            new()
            {
                SourceColumn = "p95_cpu_utilization",
                MeasureValueType = MeasureValueType.DOUBLE.Value
            },
            new()
            {
                SourceColumn = "p99_cpu_utilization",
                MeasureValueType = MeasureValueType.DOUBLE.Value
            }
        };

    TargetConfiguration targetConfiguration = new TargetConfiguration()
    {
        TimestreamConfiguration = new TimestreamConfiguration()
        {
            DatabaseName = databaseName,
            TableName = tableName,
            TimeColumn = "binned_timestamp",
            DimensionMappings = new List<DimensionMapping>()
            {
                new()
                {
                    Name = "region",
                    DimensionValueType = "VARCHAR"
                },
                new()
                {
                    Name = "az",
                    DimensionValueType = "VARCHAR"
                },
                new()
                {
                    Name = "hostname",
                    DimensionValueType = "VARCHAR"
                }
            },
            MultiMeasureMappings = new MultiMeasureMappings()
            {
                TargetMultiMeasureName = "multi-metrics",
```

```
                MultiMeasureAttributeMappings = sourceColToMeasureValueTypes
            }
        }
    };
    return await CreateScheduledQuery(topicArn, roleArn, s3ErrorReportBucketName,
        ScheduledQueryConstants.ValidQuery, targetConfiguration);
}

private async Task<String> CreateScheduledQuery(string topicArn, string roleArn,
            string s3ErrorReportBucketName, string query, TargetConfiguration
 targetConfiguration)
 {
    try
    {
        Console.WriteLine("Creating Scheduled Query");
        CreateScheduledQueryResponse response = await
 _amazonTimestreamQuery.CreateScheduledQueryAsync(
            new CreateScheduledQueryRequest()
            {
                Name = ScheduledQueryConstants.SqName,
                QueryString = query,
                ScheduleConfiguration = new ScheduleConfiguration()
                {
                    ScheduleExpression = ScheduledQueryConstants.ScheduleExpression
                },
                NotificationConfiguration = new NotificationConfiguration()
                {
                    SnsConfiguration = new SnsConfiguration()
                    {
                        TopicArn = topicArn
                    }
                },
                TargetConfiguration = targetConfiguration,
                ErrorReportConfiguration = new ErrorReportConfiguration()
                {
                    S3Configuration = new S3Configuration()
                    {
                        BucketName = s3ErrorReportBucketName
                    }
                },
                ScheduledQueryExecutionRoleArn = roleArn
            });
        Console.WriteLine($"Successfully created scheduled query :
 {response.Arn}");
```

```
            return response.Arn;
        }
        catch (Exception e)
        {
            Console.WriteLine($"Scheduled Query creation failed: {e}");
            throw;
        }
    }
```

列出计划查询

您可以使用以下代码片段来列出您的计划查询。

Java

```
public void listScheduledQueries() {
    System.out.println("Listing Scheduled Query");
    try {
        String nextToken = null;
        List<String> scheduledQueries = new ArrayList<>();

        do {
            ListScheduledQueriesResult listScheduledQueriesResult =
                    queryClient.listScheduledQueries(new
 ListScheduledQueriesRequest()
                            .withNextToken(nextToken).withMaxResults(10));
            List<ScheduledQuery> scheduledQueryList =
 listScheduledQueriesResult.getScheduledQueries();

            printScheduledQuery(scheduledQueryList);
            nextToken = listScheduledQueriesResult.getNextToken();
        } while (nextToken != null);
    }
    catch (Exception e) {
        System.out.println("List Scheduled Query failed: " + e);
        throw e;
    }
}

public void printScheduledQuery(List<ScheduledQuery> scheduledQueryList) {
    for (ScheduledQuery scheduledQuery: scheduledQueryList) {
        System.out.println(scheduledQuery.getArn());
```

```
    }
}
```

Java v2

```
public void listScheduledQueries() {
    System.out.println("Listing Scheduled Query");
    try {
        String nextToken = null;

        do {
            ListScheduledQueriesResponse listScheduledQueriesResult =

 queryClient.listScheduledQueries(ListScheduledQueriesRequest.builder()
                            .nextToken(nextToken).maxResults(10)
                            .build());
            List<ScheduledQuery> scheduledQueryList =
 listScheduledQueriesResult.scheduledQueries();

            printScheduledQuery(scheduledQueryList);
            nextToken = listScheduledQueriesResult.nextToken();
        } while (nextToken != null);
    }
    catch (Exception e) {
        System.out.println("List Scheduled Query failed: " + e);
        throw e;
    }
}

public void printScheduledQuery(List<ScheduledQuery> scheduledQueryList) {
    for (ScheduledQuery scheduledQuery: scheduledQueryList) {
        System.out.println(scheduledQuery.arn());
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) ListScheduledQueries()
 ([]*timestreamquery.ScheduledQuery, error) {

     var nextToken *string = nil
     var scheduledQueries []*timestreamquery.ScheduledQuery
     for ok := true; ok; ok = nextToken != nil {
```

```
        listScheduledQueriesInput := &timestreamquery.ListScheduledQueriesInput{
            MaxResults: aws.Int64(15),
        }
        if nextToken != nil {
            listScheduledQueriesInput.NextToken = aws.String(*nextToken)
        }

        listScheduledQueriesOutput, err :=
 timestreamBuilder.QuerySvc.ListScheduledQueries(listScheduledQueriesInput)
        if err != nil {
            fmt.Printf("Error: %s", err.Error())
            return nil, err
        }
        scheduledQueries = append(scheduledQueries,
 listScheduledQueriesOutput.ScheduledQueries...)
        nextToken = listScheduledQueriesOutput.NextToken
    }
    return scheduledQueries, nil
}
```

Python

```
def list_scheduled_queries(self):
    print("\nListing Scheduled Queries")
    try:
        response = self.query_client.list_scheduled_queries(MaxResults=10)
        self.print_scheduled_queries(response['ScheduledQueries'])
        next_token = response.get('NextToken', None)
        while next_token:
            response =
 self.query_client.list_scheduled_queries(NextToken=next_token, MaxResults=10)
            self.print_scheduled_queries(response['ScheduledQueries'])
            next_token = response.get('NextToken', None)
    except Exception as err:
        print("List scheduled queries failed:", err)
        raise err

@staticmethod
def print_scheduled_queries(scheduled_queries):
    for scheduled_query in scheduled_queries:
        print(scheduled_query['Arn'])
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function listScheduledQueries() {
     console.log("Listing Scheduled Query");
     try {
         var nextToken = null;
         do {
             var params = {
                 MaxResults: 10,
                 NextToken: nextToken
             }
             var data = await queryClient.listScheduledQueries(params).promise();
             var scheduledQueryList = data.ScheduledQueries;
             printScheduledQuery(scheduledQueryList);
             nextToken = data.NextToken;
         }
         while (nextToken != null);
     }  catch (err) {
         console.log("List Scheduled Query failed: ", err);
         throw err;
     }
 }

 async function printScheduledQuery(scheduledQueryList) {
     scheduledQueryList.forEach(element => console.log(element.Arn));
 }
```

.NET

```
private async Task ListScheduledQueries()
 {
     try
     {
         Console.WriteLine("Listing Scheduled Query");
         string nextToken;
         do
         {
             ListScheduledQueriesResponse response =
                 await _amazonTimestreamQuery.ListScheduledQueriesAsync(new
 ListScheduledQueriesRequest());
```

```
            foreach (var scheduledQuery in response.ScheduledQueries)
            {
                Console.WriteLine($"{scheduledQuery.Arn}");
            }

            nextToken = response.NextToken;
        } while (nextToken != null);
    }
    catch (Exception e)
    {
        Console.WriteLine($"List Scheduled Query failed: {e}");
        throw;
    }
}
```

描述计划查询

您可以使用以下代码片段来描述计划查询。

Java

```
public void describeScheduledQueries(String scheduledQueryArn) {
    System.out.println("Describing Scheduled Query");
    try {
        DescribeScheduledQueryResult describeScheduledQueryResult =
 queryClient.describeScheduledQuery(new
 DescribeScheduledQueryRequest().withScheduledQueryArn(scheduledQueryArn));
        System.out.println(describeScheduledQueryResult);
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Describe Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void describeScheduledQueries(String scheduledQueryArn) {
```

```
    System.out.println("Describing Scheduled Query");
    try {
        DescribeScheduledQueryResponse describeScheduledQueryResult =

 queryClient.describeScheduledQuery(DescribeScheduledQueryRequest.builder()
                        .scheduledQueryArn(scheduledQueryArn)
                        .build());
        System.out.println(describeScheduledQueryResult);
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Describe Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) DescribeScheduledQuery(scheduledQueryArn
 string) error {

    describeScheduledQueryInput := &timestreamquery.DescribeScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
    }
    describeScheduledQueryOutput, err :=
 timestreamBuilder.QuerySvc.DescribeScheduledQuery(describeScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", err.Error())
            }
        } else {
            fmt.Printf("Error: %s", aerr.Error())
        }
        return err
```

```
    } else {
        fmt.Println("DescribeScheduledQuery is successful, below is the output:")
        fmt.Println(describeScheduledQueryOutput.ScheduledQuery)
        return nil
    }
}
```

Python

```
def describe_scheduled_query(self, scheduled_query_arn):
    print("\nDescribing Scheduled Query")
    try:
        response =
 self.query_client.describe_scheduled_query(ScheduledQueryArn=scheduled_query_arn)
        if 'ScheduledQuery' in response:
            response = response['ScheduledQuery']
            for key in response:
                print("{} :{}".format(key, response[key]))
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query describe failed:", err)
        raise err
```

Node.js

以下代码段使用 for JavaScript V2 样式。 AWS SDK它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function describeScheduledQuery(scheduledQueryArn) {
    console.log("Describing Scheduled Query");
    var params = {
        ScheduledQueryArn: scheduledQueryArn
    }
    try {
        const data = await queryClient.describeScheduledQuery(params).promise();
        console.log(data.ScheduledQuery);
    } catch (err) {
        console.log("Describe Scheduled Query failed: ", err);
        throw err;
    }
}
```

.NET

```
private async Task DescribeScheduledQuery(string scheduledQueryArn)
 {
     try
     {
         Console.WriteLine("Describing Scheduled Query");
         DescribeScheduledQueryResponse response = await
 _amazonTimestreamQuery.DescribeScheduledQueryAsync(
             new DescribeScheduledQueryRequest()
             {
                 ScheduledQueryArn = scheduledQueryArn
             });

 Console.WriteLine($"{JsonConvert.SerializeObject(response.ScheduledQuery)}");
     }
     catch (ResourceNotFoundException e)
     {
         Console.WriteLine($"Scheduled Query doesn't exist: {e}");
         throw;
     }
     catch (Exception e)
     {
         Console.WriteLine($"Describe Scheduled Query failed: {e}");
         throw;
     }
 }
```

执行预设查询

您可以使用以下代码片段来运行计划查询。

Java

```
public void executeScheduledQueries(String scheduledQueryArn, Date invocationTime) {
    System.out.println("Executing Scheduled Query");
    try {
        ExecuteScheduledQueryResult executeScheduledQueryResult =
 queryClient.executeScheduledQuery(new ExecuteScheduledQueryRequest()
                .withScheduledQueryArn(scheduledQueryArn)
                .withInvocationTime(invocationTime)
        );
```

```
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void executeScheduledQuery(String scheduledQueryArn) {
    System.out.println("Executing Scheduled Query");
    try {
        ExecuteScheduledQueryResponse executeScheduledQueryResult =
 queryClient.executeScheduledQuery(ExecuteScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn)
                .invocationTime(Instant.now())
                .build()
        );

        System.out.println("Execute ScheduledQuery response code: " +
 executeScheduledQueryResult.sdkHttpResponse().statusCode());

    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) ExecuteScheduledQuery(scheduledQueryArn
 string, invocationTime time.Time) error {
```

```
    executeScheduledQueryInput := &timestreamquery.ExecuteScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
        InvocationTime:    aws.Time(invocationTime),
    }
    executeScheduledQueryOutput, err :=
timestreamBuilder.QuerySvc.ExecuteScheduledQuery(executeScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("ExecuteScheduledQuery is successful, below is the output:")
        fmt.Println(executeScheduledQueryOutput.GoString())
        return nil
    }
}
```

Python

```
def execute_scheduled_query(self, scheduled_query_arn, invocation_time):
    print("\nExecuting Scheduled Query")
    try:

 self.query_client.execute_scheduled_query(ScheduledQueryArn=scheduled_query_arn,
 InvocationTime=invocation_time)
        print("Successfully started executing scheduled query")
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query execution failed:", err)
        raise err
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function executeScheduledQuery(scheduledQueryArn, invocationTime) {
     console.log("Executing Scheduled Query");
     var params = {
         ScheduledQueryArn: scheduledQueryArn,
         InvocationTime: invocationTime
     }
     try {
         await queryClient.executeScheduledQuery(params).promise();
     } catch (err) {
         console.log("Execute Scheduled Query failed: ", err);
         throw err;
     }
 }
```

.NET

```
private async Task ExecuteScheduledQuery(string scheduledQueryArn, DateTime
 invocationTime)
 {
     try
     {
         Console.WriteLine("Running Scheduled Query");
         await _amazonTimestreamQuery.ExecuteScheduledQueryAsync(new
 ExecuteScheduledQueryRequest()
         {
             ScheduledQueryArn = scheduledQueryArn,
             InvocationTime = invocationTime
         });
         Console.WriteLine("Successfully started manual run of scheduled query");
     }
     catch (ResourceNotFoundException e)
     {
         Console.WriteLine($"Scheduled Query doesn't exist: {e}");
         throw;
     }
     catch (Exception e)
     {
         Console.WriteLine($"Execute Scheduled Query failed: {e}");
```

```
            throw;
        }
    }
```

更新计划查询

您可以使用以下代码片段来更新计划查询。

Java

```
public void updateScheduledQueries(String scheduledQueryArn) {
    System.out.println("Updating Scheduled Query");
    try {
        queryClient.updateScheduledQuery(new UpdateScheduledQueryRequest()
                .withScheduledQueryArn(scheduledQueryArn)
                .withState(ScheduledQueryState.DISABLED));
        System.out.println("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void updateScheduledQuery(String scheduledQueryArn, ScheduledQueryState
 state) {
    System.out.println("Updating Scheduled Query");
    try {
        queryClient.updateScheduledQuery(UpdateScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn)
                .state(state)
                .build());
        System.out.println("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
```

```
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) UpdateScheduledQuery(scheduledQueryArn
 string) error {

    updateScheduledQueryInput := &timestreamquery.UpdateScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
        State:             aws.String(timestreamquery.ScheduledQueryStateDisabled),
    }
    _, err :=
 timestreamBuilder.QuerySvc.UpdateScheduledQuery(updateScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("UpdateScheduledQuery is successful")
        return nil
    }
}
```

Python

```
def update_scheduled_query(self, scheduled_query_arn, state):
    print("\nUpdating Scheduled Query")
```

```
    try:

 self.query_client.update_scheduled_query(ScheduledQueryArn=scheduled_query_arn,
                                                    State=state)
        print("Successfully update scheduled query state to", state)
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query deletion failed:", err)
        raise err
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function updateScheduledQueries(scheduledQueryArn) {
     console.log("Updating Scheduled Query");
     var params = {
         ScheduledQueryArn: scheduledQueryArn,
         State: "DISABLED"
     }
     try {
         await queryClient.updateScheduledQuery(params).promise();
         console.log("Successfully update scheduled query state");
     } catch (err) {
         console.log("Update Scheduled Query failed: ", err);
         throw err;
     }
 }
```

.NET

```
private async Task UpdateScheduledQuery(string scheduledQueryArn,
 ScheduledQueryState state)
 {
     try
     {
         Console.WriteLine("Updating Scheduled Query");
         await _amazonTimestreamQuery.UpdateScheduledQueryAsync(new
 UpdateScheduledQueryRequest()
         {
```

```
            ScheduledQueryArn = scheduledQueryArn,
            State = state
        });
        Console.WriteLine("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e)
    {
        Console.WriteLine($"Scheduled Query doesn't exist: {e}");
        throw;
    }
    catch (Exception e)
    {
        Console.WriteLine($"Update Scheduled Query failed: {e}");
        throw;
    }
}
```

删除计划查询

您可以使用以下代码段来删除计划查询。

Java

```
public void deleteScheduledQuery(String scheduledQueryArn) {
    System.out.println("Deleting Scheduled Query");

    try {
        queryClient.deleteScheduledQuery(new
 DeleteScheduledQueryRequest().withScheduledQueryArn(scheduledQueryArn));
        System.out.println("Successfully deleted scheduled query");
    }
    catch (Exception e) {
        System.out.println("Scheduled Query deletion failed: " + e);
    }
}
```

Java v2

```
public void deleteScheduledQuery(String scheduledQueryArn) {
    System.out.println("Deleting Scheduled Query");

    try {
```

```
        queryClient.deleteScheduledQuery(DeleteScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn).build());
        System.out.println("Successfully deleted scheduled query");
    }
    catch (Exception e) {
        System.out.println("Scheduled Query deletion failed: " + e);
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) DeleteScheduledQuery(scheduledQueryArn
 string) error {

    deleteScheduledQueryInput := &timestreamquery.DeleteScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
    }
    _, err :=
 timestreamBuilder.QuerySvc.DeleteScheduledQuery(deleteScheduledQueryInput)

    if err != nil {
        fmt.Println("Error:")
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("DeleteScheduledQuery is successful")
        return nil
    }
}
```

Python

```
def delete_scheduled_query(self, scheduled_query_arn):
```

```
    print("\nDeleting Scheduled Query")
    try:

 self.query_client.delete_scheduled_query(ScheduledQueryArn=scheduled_query_arn)
        print("Successfully deleted scheduled query :", scheduled_query_arn)
    except Exception as err:
        print("Scheduled Query deletion failed:", err)
        raise err
```

Node.js

以下代码段使用了 for AWS SDK JavaScript V2 样式。它基于 Node.js 示例 Amazon Timestream 中的示例 LiveAnalytics 应用程序，供其上使用。 GitHub

```
async function deleteScheduleQuery(scheduledQueryArn) {
     console.log("Deleting Scheduled Query");
     const params = {
         ScheduledQueryArn: scheduledQueryArn
     }
     try {
         await queryClient.deleteScheduledQuery(params).promise();
         console.log("Successfully deleted scheduled query");
     } catch (err) {
         console.log("Scheduled Query deletion failed: ", err);
     }
 }
```

.NET

```
private async Task DeleteScheduledQuery(string scheduledQueryArn)
 {
     try
     {
         Console.WriteLine("Deleting Scheduled Query");
         await _amazonTimestreamQuery.DeleteScheduledQueryAsync(new
 DeleteScheduledQueryRequest()
         {
             ScheduledQueryArn = scheduledQueryArn
         });
         Console.WriteLine($"Successfully deleted scheduled query :
 {scheduledQueryArn}");
     }
     catch (Exception e)
```

```
    {
        Console.WriteLine($"Scheduled Query deletion failed: {e}");
        throw;
    }
}
```

在 Timestream 中使用批量加载 LiveAnalytics

通过 Amazon Timestream 的批量加载 LiveAnalytics，您可以将存储在 Amazon S3 中的CSV文件批量提取到 Timestream 中。借助这项新功能，您 LiveAnalytics 无需依赖其他工具或编写自定义代码，即可在 Timestream 中保存数据。您可以使用批量加载来回填具有灵活等待时间的数据，例如查询或分析不需要立即使用的数据。

您可以使用、和 AWS Management Console AWS CLI，创建批量加载任务 AWS SDKs。有关更多信息，请参阅 在控制台中使用批量加载、将批量加载与 AWS CLI 和 将批量加载与 AWS SDKs。

除了批量加载外，您还可以在 WriteRecords API操作中同时写入多条记录。有关使用哪个的指导，请参阅在 WriteRecords API操作和批量加载之间进行选择。

主题

- Timestream 中的批量加载概念
- 批量加载先决条件
- Batch 加载最佳实践
- 准备批量加载数据文件
- 批量加载的数据模型映射
- 在控制台中使用批量加载
- 将批量加载与 AWS CLI
- 将批量加载与 AWS SDKs
- 使用批量加载错误报告

Timestream 中的批量加载概念

查看以下概念，以更好地了解批量加载功能。

批量加载任务 — 在 Amazon Timestream 中定义您的源数据和目标的任务。在创建批量加载任务时，您可以指定其他配置，例如数据模型。您可以通过 AWS Management Console AWS CLI、和创建批量加载任务 AWS SDKs。

导入目标 — Timestream 中的目标数据库和表。有关创建数据库和表的信息，请参见创建 数据库和创建表。

数据源-存储在 S3 存储桶中的源CSV文件。有关准备数据文件的信息，请参见准备批量加载数据文件。有关 S3 定价的信息，请参阅 Amazon S3 定价。

批量加载错误报告-存储有关批量加载任务错误信息的报告。作为批量加载任务的一部分，您可以定义批量加载错误报告的 S3 位置。有关报告中的信息的信息，请参阅使用批量加载错误报告。

数据模型映射-时间、维度和度量的批量加载映射，从 S3 位置的数据源到 LiveAnalytics 表的目标时间流。有关更多信息，请参阅 批量加载的数据模型映射。

批量加载先决条件

这是使用批量加载的先决条件列表。有关最佳实践，请参阅Batch 加载最佳实践。

- 批量加载源数据以带有标题的CSV格式存储在 Amazon S3 中。
- 对于每个 Amazon S3 源存储桶，您必须在附加的策略中拥有以下权限：

```
"s3:GetObject",
"s3:GetBucketAcl"
"s3:ListBucket"
```

同样，对于写入报告的每个 Amazon S3 输出存储桶，您必须在附加的策略中拥有以下权限：

```
"s3:PutObject",
"s3:GetBucketAcl"
```

例如：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject",
```

```
        "s3:GetBucketAcl"
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::inputs-source-bucket-name-A”
        "arn:aws:s3:::inputs-source-bucket-name-B”
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:PutObject",
        "s3:GetBucketAcl"
      ],
      "Resource": [
        "arn:aws:s3:::reports-output-bucket-name”
      ]
      "Effect": "Allow"
    }
  ]
}
```

- Timestream fo CSV r 通过将数据模型中提供的信息映射到标题来 LiveAnalytics CSV解析。数据必须包含代表时间戳的列、至少一个维度列和至少一个度量列。
- 用于批量加载的 S3 存储桶必须与用于批量加载的 LiveAnalytics 表的 Timestream 位于同一区域，并且来自同一账户。
- 该`timestamp`列必须是长数据类型，表示自 Unix 时代以来的时间。例如，时间戳`2021-03-25T08:45:21Z`将表示为。`1616661921`Timestream 支持秒、毫秒、微秒和纳秒，以实现时间戳精度。使用查询语言时，您可以使用诸如之类的函数在格式之间进行转换`to_unixtime`。有关更多信息，请参阅 日期/时间函数。
- Timestream 支持维度值的字符串数据类型。它支持度量列的长整型、双精度、字符串和布尔型数据类型。

有关批量加载限制和配额，请参阅批量加载。

Batch 加载最佳实践

在遵守以下条件和建议时，Batch 加载效果最佳（高吞吐量）：

1. CSV为了提高并行性和摄取速度，提交供摄取的文件很小，特别是文件大小为 100MB—1GB。

2. 在批量加载过程中，避免将数据同时提取到同一个表中（例如使用 WriteRecords API操作或计划查询）。这可能会导致限制，并且批量加载任务将失败。
3. 在批处理加载任务运行时，请勿在批量加载中使用的 S3 存储桶中添加、修改或删除文件。
4. 请勿从表或源中删除或撤消权限，也不要报告已计划或正在执行批量加载任务的 S3 存储桶。
5. 摄取具有高基数维度值集的数据时，请按照中的指导进行操作。对多度量记录进行分区的建议
6. 请务必通过提交一个小文件来测试数据的正确性。无论数据是否正确，您都需要为提交到批量加载的任何数据付费。有关定价的更多信息，请参阅亚马逊 Timestream 定价。
7. 除非低于 250，否则`ActiveMagneticStorePartitions`不要恢复批量加载任务。该作业可能会受到限制并失败。同时为同一个数据库提交多个作业应该可以减少数量。

以下是控制台最佳实践：

1. 仅使用构建器进行更简单的数据建模，即多度量记录仅使用一个度量名称。
2. 要进行更复杂的数据建模，请使用JSON。例如，JSON在使用多度量记录时使用多个度量名称时使用。

有关 LiveAnalytics 最佳实践的更多信息 Timestream，请参阅最佳实践。

准备批量加载数据文件

源数据文件具有分隔符分隔的值。更具体的术语通常使用逗号分隔的值 (CSV)。有效的列分隔符包括逗号和竖线。记录用新行分隔。文件必须存储在 Amazon S3 中。创建新的批量加载任务时，源数据的位置由文件指定。ARN文件包含标题。一列代表时间戳。至少还有一列代表一个度量。

用于批量加载的 S3 存储桶必须与用于批量加载的 LiveAnalytics 表的时间流位于同一个区域中。提交批量加载任务后，请勿在批量加载中使用的 S3 存储桶中添加或删除文件。有关使用 S3 存储桶的信息，请参阅 Amazon S3 入门。

> **Note**
>
> CSV由某些应用程序（例如 Excel）生成的文件可能包含与预期编码冲突的字节顺序标记 (BOM)。 LiveAnalytics 批量加载任务的时间流，这些任务引用了在以编程方式处理时会引BOM发错误的CSV文件。为避免这种情况，您可以移除BOM，这是一个不可见的字符。例如，您可以从允许您指定新编码的应用程序（例如 Notepad++）中保存文件。您也可以使用编程选项来读取第一行，从该行中删除字符，然后将新值写入文件中的第一行。

从 Excel 保存时，有多个CSV选项。使用其他CSV选项保存可能会防止出现上述问题。但是你应该检查结果，因为编码的变化可能会影响某些字符。

CSV格式参数

当你表示一个原本由格式参数保留的值时，你可以使用转义字符。例如，如果引号字符是双引号，则要在数据中表示双引号，请将转义字符放在双引号之前。

有关在创建批量加载任务时何时指定这些内容的信息，请参阅创建批量加载任务。

参数	Options
列分隔符	(逗号 (',') | 竖线 ('|') | 分号 (';') | Tab ('/t') | 空格 ('))
转义字符	none
引用字符	控制台：（双引号 (“) | 单引号 ('))
空值	空格 (')
修剪空白	主机：（否 | 是）

批量加载的数据模型映射

以下内容讨论了数据模型映射的架构，并给出了示例。

数据模型映射架构

`CreateBatchLoadTask`请求语法和调用返回的`BatchLoadTaskDescription`对象，该`DataModelConfiguration`对象`DescribeBatchLoadTask`包含`DataModel`用于批量加载的对象。`DataModel`定义了从以 S3 位置CSV格式存储的源数据到 LiveAnalytics 数据库和表的目标 Timestream 的映射。

该`TimeColumn`字段表示要映射到 Timestream 中目标表`time`列的值的 LiveAnalytics源数据的位置。`TimeUnit`指定了的单位`TimeColumn`，并且可以是`MILLISECONDS`、`SECONDSMICROSECONDS`、或之一`NANOSECONDS`。还有维度和度量的映射。维度映射由源列和目标字段组成。

有关更多信息，请参阅DimensionMapping。度量的映射有两个选项，MixedMeasureMappings和。MultiMeasureMappings

总而言之，a DataModel 包含从 S3 位置的数据源到以下 LiveAnalytics 表的目标时间流的映射。

- 时间
- 尺寸
- 度量

如果可能，我们建议您将测量数据映射到 Timestream 中的多度量记录中。 LiveAnalytics有关多度量记录的好处的信息，请参见多重测量记录。

如果源数据中的多个度量存储在一行中，则可以将这些多个度量映射到 Timestream 中的多度量记录以供使用 LiveAnalytics 。MultiMeasureMappings如果存在必须映射到单度量记录的值，则可以使用MixedMeasureMappings。

MixedMeasureMappingsMultiMeasureMappings两者都包括MultiMeasureAttributeMappings。无论是否需要单度量记录，都支持多度量记录。

如果 Timestream 中只需要多度量目标记录 LiveAnalytics，则可以在以下结构中定义度量映射。

```
CreateBatchLoadTask
    MeasureNameColumn
    MultiMeasureMappings
        TargetMultiMeasureName
        MultiMeasureAttributeMappings array
```

> **Note**
>
> 我们建议MultiMeasureMappings尽可能使用。

如果 Timestream 中需要单度量目标记录 LiveAnalytics，则可以在以下结构中定义度量映射。

```
CreateBatchLoadTask
    MeasureNameColumn
    MixedMeasureMappings array
        MixedMeasureMapping
            MeasureName
```

```
            MeasureValueType
            SourceColumn
            TargetMeasureName
            MultiMeasureAttributeMappings array
```

使用时MultiMeasureMappings，MultiMeasureAttributeMappings数组始终是必需的。当你使用MixedMeasureMappings数组时，如果MeasureValueType是给MULTI定的MixedMeasureMapping，则MultiMeasureAttributeMappings是必需的MixedMeasureMapping。否则，MeasureValueType表示单项测量记录的度量类型。

无论哪种方式，都有一系列MultiMeasureAttributeMapping可用。您可以MultiMeasureAttributeMapping按如下方式定义每条中多度量记录的映射：

SourceColumn

源数据中位于 Amazon S3 中的列。

TargetMultiMeasureAttributeName

目标表中目标多指标名称的名称。如果未提供此输入MeasureNameColumn，则需要此输入。如果提供，MeasureNameColumn则该列中的值将用作多度量名称。

MeasureValueType

、DOUBLEBIGINTBOOLEANVARCHAR、或之一TIMESTAMP。

数据模型映射与示例 MultiMeasureMappings

此示例演示了映射到多度量记录的首选方法，它将每个度量值存储在专用列中。你可以在样本CSV处下载样本CSV。该示例具有以下标题，可映射到表的 Timestream 中的目标列。 LiveAnalytics

- time
- measure_name
- region
- location
- hostname
- memory_utilization
- cpu_utilization

识别CSV文件中的time和measure_name列。在这种情况下，它们直接映射到同名 LiveAnalytics 表列的时间流。

- time映射到 time
- measure_name映射到measure_name（或您选择的值）

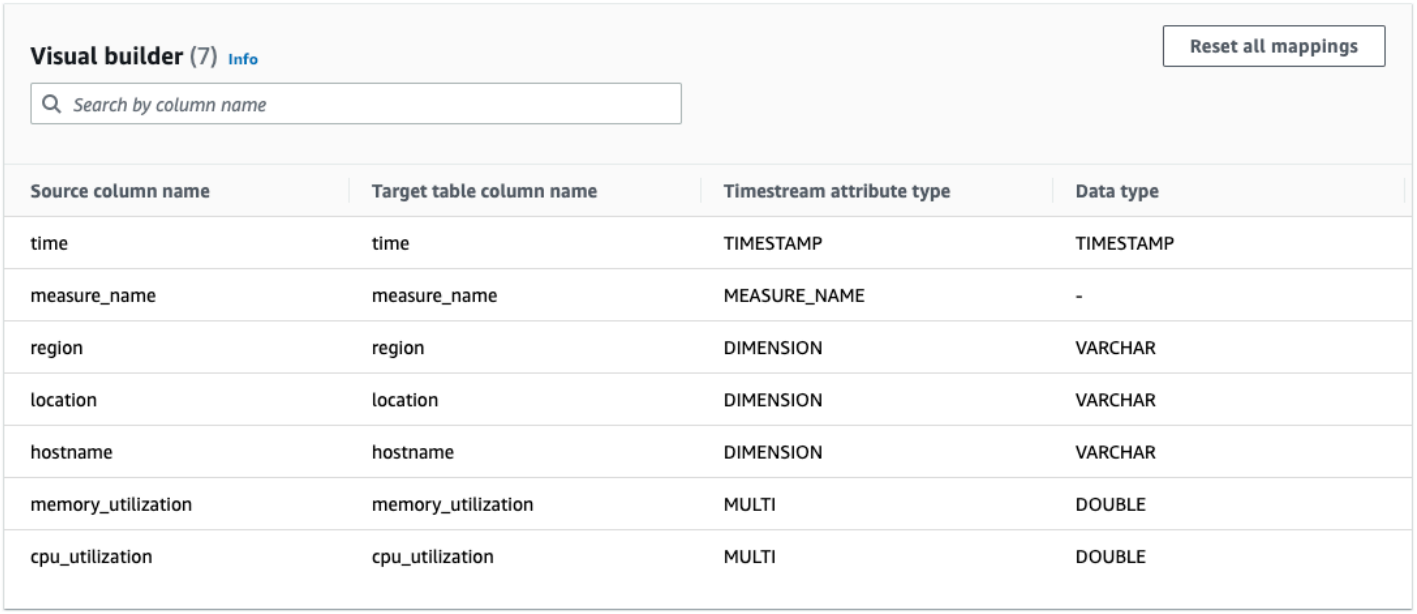
使用时API，可以在TimeColumn字段time中指定支持的时间单位值，例如MILLISECONDS在TimeUnit字段中。它们对应于控制台中输入的源列名称和时间戳时间。您可以使用MeasureNameColumn密钥定义measure_name的记录分组或分区。

在样本中region，location、和hostname是维度。维度映射到一个DimensionMapping对象数组中。

对于度量，该值TargetMultiMeasureAttributeName将变为 “时间流” LiveAnalytics 表中的一列。您可以保留相同的名称，如本例所示。或者你可以指定一个新的。MeasureValueType是、DOUBLE、BIGINTBOOLEANVARCHAR、或之一TIMESTAMP。

```
{
  "TimeColumn": "time",
  "TimeUnit": "MILLISECONDS",
  "DimensionMappings": [
    {
      "SourceColumn": "region",
      "DestinationColumn": "region"
    },
    {
      "SourceColumn": "location",
      "DestinationColumn": "location"
    },
    {
      "SourceColumn": "hostname",
      "DestinationColumn": "hostname"
    }
  ],
  "MeasureNameColumn": "measure_name",
  "MultiMeasureMappings": {
    "MultiMeasureAttributeMappings": [
      {
        "SourceColumn": "memory_utilization",
        "TargetMultiMeasureAttributeName": "memory_utilization",
        "MeasureValueType": "DOUBLE"
```

```
      },
      {
        "SourceColumn": "cpu_utilization",
        "TargetMultiMeasureAttributeName": "cpu_utilization",
        "MeasureValueType": "DOUBLE"
      }
    ]
  }
}
```



Visual builder (7) Info

Reset all mappings

Search by column name

Source column name	Target table column name	Timestream attribute type	Data type
time	time	TIMESTAMP	TIMESTAMP
measure_name	measure_name	MEASURE_NAME	-
region	region	DIMENSION	VARCHAR
location	location	DIMENSION	VARCHAR
hostname	hostname	DIMENSION	VARCHAR
memory_utilization	memory_utilization	MULTI	DOUBLE
cpu_utilization	cpu_utilization	MULTI	DOUBLE

数据模型映射与示例 `MixedMeasureMappings`

我们建议您仅在需要映射到 Timestream 中的单度记录时才使用此方法。 LiveAnalytics

在控制台中使用批量加载

以下是使用批量加载的步骤 AWS Management Console。你可以在样本CSV处下载样本CSV。

主题

- 访问批量加载
- 创建批量加载任务
- 恢复批量加载任务
- 使用可视化生成器

访问批量加载

按照以下步骤使用访问批量加载 AWS Management Console。

1. 打开亚马逊 Timestream 控制台。
2. 在导航窗格中，选择管理工具，然后选择批量加载任务。
3. 在这里，您可以查看批量加载任务列表，并深入研究给定任务以获取更多详细信息。您也可以创建和恢复任务。

创建批量加载任务

按照以下步骤使用创建批量加载任务 AWS Management Console。

1. 打开亚马逊 Timestream 控制台。
2. 在导航窗格中，选择管理工具，然后选择批量加载任务。
3. 选择 “创建批量加载任务”。
4. 在导入目标中，选择以下选项。

   - 目标数据库-选择在中创建的数据库的名称创建 数据库。
   - 目标表-选择在中创建的表的名称创建表。

   如有必要，您可以使用 “创建新表” 按钮从此面板添加表格。
5. 从数据源中的数据源 S3 位置中，选择存储源数据的 S3 存储桶。使用浏览 S3 按钮查看活跃AWS账户有权访问的 S3 资源，或者输入 S3 位置URL。数据源必须位于同一区域。
6. 在文件格式设置（可展开部分）中，您可以使用默认设置来解析输入数据。您也可以选择 “高级设置”。从那里你可以选择CSV格式参数，也可以选择参数来解析输入数据。有关这些参数的信息，请参见CSV格式参数。
7. 在配置数据模型映射中，配置数据模型。有关其他数据模型指南，请参阅 批量加载的数据模型映射

   - 在数据模型映射中，选择映射配置输入，然后选择以下选项之一。
     - 可视化生成器-要直观地映射数据，请选择TargetMultiMeasureName或MeasureNameColumn。然后从可视化生成器中映射列。

       当选择单个文件作为数据源时，Visual Builder 会自动检测并加载数据源CSV文件中的源列标题。选择要创建映射的属性和数据类型。

有关使用可视化生成器的信息，请参见使用可视化生成器。

- JSON编辑器-用于配置数据模型的自由格式JSON编辑器。如果您熟悉 Timestream，LiveAnalytics 并且想要构建高级数据模型映射，请选择此选项。
- JSON来自 S3 的文件-选择存储在 S3 中的JSON模型文件。如果您已经配置了数据模型并希望将其重复用于其他批量加载，请选择此选项。

8. 从错误日志报告中的错误日志 S3 位置中，选择将用于报告错误的 S3 位置。有关如何使用此报告的信息，请参阅使用批量加载错误报告。
9. 对于加密密钥类型，请选择以下选项之一。
   - Amazon S3 托管密钥 (SSE-S3) — Amazon S3 为您创建、管理和使用的加密密钥。
   - AWS KMS key (SSE-KMS)- 受 AWS Key Management Service (AWS KMS) 保护的加密密钥。
10. 选择下一步。
11. 在 “查看并创建” 页面上，查看设置并根据需要进行编辑。

> **Note**
>
> 创建任务后，您无法更改批量加载任务设置。任务完成时间将根据导入的数据量而有所不同。

12. 选择 “创建批量加载任务”。

恢复批量加载任务

当您选择状态为 “进度已停止” 且仍可恢复的批量加载任务时，系统会提示您继续执行该任务。当您查看这些任务的详细信息时，还有一个带有 “继续任务” 按钮的横幅。可恢复的任务有 “恢复截止日期”。在该日期到期后，任务将无法恢复。

使用可视化生成器

您可以使用可视化生成器将存储在 S3 存储桶中的一个或多个CSV文件源数据列映射到 LiveAnalytics 表的 Timestream 中的目标列。

> **Note**
>
> 您的角色需要文件`SelectObjectContent`权限。否则，您将需要手动添加和删除列。

自动加载源列模式

如果您只指定一个存储桶，Timestream for LiveAnalytics 可以自动扫描源CSV文件中的列名。如果没有现有的映射，则可以选择 “导入源列”。

1. 从 “映射” 配置输入设置中选择 “Visual Builder” 选项后，设置时间戳时间输入。`Milliseconds`是默认设置。
2. 单击 “加载源列” 按钮以导入源数据文件中的列标题。该表将使用数据源文件中的源列标题名称进行填充。
3. 为每个源列选择目标表列名、Timestream 属性类型和数据类型。

   有关这些列和可能值的详细信息，请参见映射字段。
4. 使用该 drag-to-fill功能一次性设置多列的值。

手动添加源列

如果您使用的是存储桶或CSV前缀，而不是单个存储桶或前缀CSV，则可以使用 “添加列映射” 和 “删除列映射” 按钮在可视化编辑器中添加和删除列映射。还有一个用于重置映射的按钮。

映射字段

- 源列名-源文件中表示要导入的度量的列的名称。当您使用导入源列时，Timestream LiveAnalytics 可以自动填充此值。
- 目标表列名-可选输入，用于指示目标表中度量的列名。
- Timestream 属性类型-指定源列中数据的属性类型，`DIMENSION`例如。
  - TIMESTAMP— 指定何时收集度量。
  - MULTI— 表示多个度量。
  - DIMENSION— 时间序列元数据。
  - MEASURE_ NAME — 对于单小节记录，这是度量名称。
- 数据类型-“时间流” 列的类型，例如`BOOLEAN`。
  - BIGINT— 一个 64 位整数。
  - BOOLEAN— 逻辑的两个真值——真和假。
  - DOUBLE— 64 位可变精度数字。
  - TIMESTAMP— 一个时间实例，它使用纳秒精度的时间UTC，并跟踪自 Unix 时代以来的时间。

将批量加载与 AWS CLI

设置

要开始使用批量加载，请执行以下步骤。

1. 按照中的说明 AWS CLI 进行安装访问 Amazon Timestream 以使用 LiveAnalytics AWS CLI。
2. 运行以下命令以验证 Timestream CLI 命令是否已更新。验证 create-batch-load-task是否在列表中。

   ```
   aws timestream-write help
   ```
3. 按照中的说明准备数据源准备批量加载数据文件。
4. 按照中的说明创建数据库和表访问 Amazon Timestream 以使用 LiveAnalytics AWS CLI。
5. 为报告输出创建 S3 存储桶。存储桶必须位于同一区域。有关存储桶的更多信息，请参阅创建、配置和使用 Amazon S3 存储桶。
6. 创建批量加载任务。要查看步骤，请参阅 创建批量加载任务。
7. 确认任务的状态。要查看步骤，请参阅 描述批量加载任务。

创建批量加载任务

您可以使用create-batch-load-task命令创建批量加载任务。使用创建批量加载任务时CLI，您可以使用JSON参数cli-input-json，该参数允许您将参数聚合到单个JSON片段中。您还可以使用其他几个参数（包括data-model-configuration、data-source-configurationreport-configurationtarget-database-name、和）将这些细节分开target-table-name。

有关示例，请参阅 创建批量加载任务示例。

描述批量加载任务

您可以按如下方式检索批量加载任务描述。

```
aws timestream-write describe-batch-load-task --task-id <value>
```

以下是示例响应。

```
{
    "BatchLoadTaskDescription": {
        "TaskId": "<TaskId>",
        "DataSourceConfiguration": {
```

```
            "DataSourceS3Configuration": {
                "BucketName": "test-batch-load-west-2",
                "ObjectKeyPrefix": "sample.csv"
            },
            "CsvConfiguration": {},
            "DataFormat": "CSV"
        },
        "ProgressReport": {
            "RecordsProcessed": 2,
            "RecordsIngested": 0,
            "FileParseFailures": 0,
            "RecordIngestionFailures": 2,
            "FileFailures": 0,
            "BytesIngested": 119
        },
        "ReportConfiguration": {
            "ReportS3Configuration": {
                "BucketName": "test-batch-load-west-2",
                "ObjectKeyPrefix": "<ObjectKeyPrefix>",
                "EncryptionOption": "SSE_S3"
            }
        },
        "DataModelConfiguration": {
            "DataModel": {
                "TimeColumn": "timestamp",
                "TimeUnit": "SECONDS",
                "DimensionMappings": [
                    {
                        "SourceColumn": "vehicle",
                        "DestinationColumn": "vehicle"
                    },
                    {
                        "SourceColumn": "registration",
                        "DestinationColumn": "license"
                    }
                ],
                "MultiMeasureMappings": {
                    "TargetMultiMeasureName": "test",
                    "MultiMeasureAttributeMappings": [
                        {
                            "SourceColumn": "wgt",
                            "TargetMultiMeasureAttributeName": "weight",
                            "MeasureValueType": "DOUBLE"
                        },
```

```
                        {
                            "SourceColumn": "spd",
                            "TargetMultiMeasureAttributeName": "speed",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "SourceColumn": "fuel",
                            "TargetMultiMeasureAttributeName": "fuel",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "SourceColumn": "miles",
                            "TargetMultiMeasureAttributeName": "miles",
                            "MeasureValueType": "DOUBLE"
                        }
                    ]
                }
            }
        },
        "TargetDatabaseName": "BatchLoadExampleDatabase",
        "TargetTableName": "BatchLoadExampleTable",
        "TaskStatus": "FAILED",
        "RecordVersion": 1,
        "CreationTime": 1677167593.266,
        "LastUpdatedTime": 1677167602.38
    }
}
```

列出批量加载任务

您可以按如下方式列出批量加载任务。

```
aws timestream-write list-batch-load-tasks
```

输出如下所示。

```
{
    "BatchLoadTasks": [
        {
            "TaskId": "<TaskId>",
            "TaskStatus": "FAILED",
            "DatabaseName": "BatchLoadExampleDatabase",
            "TableName": "BatchLoadExampleTable",
```

```
            "CreationTime": 1677167593.266,
            "LastUpdatedTime": 1677167602.38
        }
    ]
}
```

恢复批量加载任务

您可以按如下方式继续执行批量加载任务。

```
aws timestream-write resume-batch-load-task --task-id <value>
```

响应可以表示成功或包含错误信息。

创建批量加载任务示例

Example

1. 为名为的 LiveAnalytics 数据库BatchLoad和名为的表创建时间流。BatchLoadTest验证并根据需要调整MemoryStoreRetentionPeriodInHours和的值MagneticStoreRetentionPeriodInDays。

   ```
   aws timestream-write create-database --database-name BatchLoad \

   aws timestream-write create-table --database-name BatchLoad \
   --table-name BatchLoadTest \
   --retention-properties "{\"MemoryStoreRetentionPeriodInHours\": 12,
    \"MagneticStoreRetentionPeriodInDays\": 100}"
   ```

2. 使用控制台创建 S3 存储桶并将sample.csv文件复制到该位置。你可以在样本CSV处下载样本CSV。
3. 使用控制台为 Timestream 创建 S3 存储桶 LiveAnalytics ，以便在批量加载任务完成时出现错误时编写报告。
4. 创建批量加载任务。请务必更换 *$INPUT_BUCKET* 以及 *$REPORT_BUCKET* 使用您在前面的步骤中创建的存储桶。

   ```
   aws timestream-write create-batch-load-task \
   --data-model-configuration "{\
               \"DataModel\": {\
                 \"TimeColumn\": \"timestamp\",\
                 \"TimeUnit\": \"SECONDS\",\
   ```

```
              \"DimensionMappings\": [\
                {\
                  \"SourceColumn\": \"vehicle\"\
                },\
                {\
                  \"SourceColumn\": \"registration\",\
                  \"DestinationColumn\": \"license\"\
                }\
              ],
              \"MultiMeasureMappings\": {\
                \"TargetMultiMeasureName\": \"mva_measure_name\",\
                \"MultiMeasureAttributeMappings\": [\
                  {\
                    \"SourceColumn\": \"wgt\",\
                    \"TargetMultiMeasureAttributeName\": \"weight\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"spd\",\
                    \"TargetMultiMeasureAttributeName\": \"speed\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"fuel_consumption\",\
                    \"TargetMultiMeasureAttributeName\": \"fuel\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"miles\",\
                    \"MeasureValueType\": \"BIGINT\"\
                  }\
                ]\
              }\
            }\
          }" \
--data-source-configuration "{
            \"DataSourceS3Configuration\": {\
              \"BucketName\": \"$INPUT_BUCKET\",\
              \"ObjectKeyPrefix\": \"$INPUT_OBJECT_KEY_PREFIX\"
            },\
            \"DataFormat\": \"CSV\"\
          }" \
--report-configuration "{\
            \"ReportS3Configuration\": {\
```

```
              \"BucketName\": \"$REPORT_BUCKET\",\
              \"EncryptionOption\": \"SSE_S3\"\
            }\
          }" \
--target-database-name BatchLoad \
--target-table-name BatchLoadTest
```

前面的命令返回以下输出。

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

5. 检查任务的进度。请务必更换 *$TASK_ID* 使用在上一步中返回的任务 ID。

```
aws timestream-write describe-batch-load-task --task-id $TASK_ID
```

示例输出

```
{
    "BatchLoadTaskDescription": {
        "ProgressReport": {
            "BytesIngested": 1024,
            "RecordsIngested": 2,
            "FileFailures": 0,
            "RecordIngestionFailures": 0,
            "RecordsProcessed": 2,
            "FileParseFailures": 0
        },
        "DataModelConfiguration": {
            "DataModel": {
                "DimensionMappings": [
                    {
                        "SourceColumn": "vehicle",
                        "DestinationColumn": "vehicle"
                    },
                    {
                        "SourceColumn": "registration",
                        "DestinationColumn": "license"
                    }
                ],
```

```
                "TimeUnit": "SECONDS",
                "TimeColumn": "timestamp",
                "MultiMeasureMappings": {
                    "MultiMeasureAttributeMappings": [
                        {
                            "TargetMultiMeasureAttributeName": "weight",
                            "SourceColumn": "wgt",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "TargetMultiMeasureAttributeName": "speed",
                            "SourceColumn": "spd",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "TargetMultiMeasureAttributeName": "fuel",
                            "SourceColumn": "fuel_consumption",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "TargetMultiMeasureAttributeName": "miles",
                            "SourceColumn": "miles",
                            "MeasureValueType": "DOUBLE"
                        }
                    ],
                    "TargetMultiMeasureName": "mva_measure_name"
                }
            }
        },
        "TargetDatabaseName": "BatchLoad",
        "CreationTime": 1672960381.735,
        "TaskStatus": "SUCCEEDED",
        "RecordVersion": 1,
        "TaskId": "TaskId ",
        "TargetTableName": "BatchLoadTest",
        "ReportConfiguration": {
            "ReportS3Configuration": {
                "EncryptionOption": "SSE_S3",
                "ObjectKeyPrefix": "ObjectKeyPrefix ",
                "BucketName": "test-report-bucket"
            }
        },
        "DataSourceConfiguration": {
            "DataSourceS3Configuration": {
```

```
                "ObjectKeyPrefix": "sample.csv",
                "BucketName": "test-input-bucket"
            },
            "DataFormat": "CSV",
            "CsvConfiguration": {}
        },
        "LastUpdatedTime": 1672960387.334
    }
}
```

将批量加载与 AWS SDKs

有关如何使用创建、描述和列出批量加载任务的示例 AWS SDKs，请参阅创建批量加载任务描述批量加载任务、列出批量加载任务、和恢复批量加载任务。

使用批量加载错误报告

Batch Load 任务具有以下状态值之一：

- CREATED（已创建）-任务已创建。
- IN_PROGRESS（进行中）-任务正在进行中。
- FAILED（失败）-任务已完成。但是检测到一个或多个错误。
- SUCCEEDED（已完成）-任务已完成，未出现任何错误。
- PROGRESS_STOPPED（进度已停止）-任务已停止但尚未完成。您可以尝试继续执行任务。
- PENDING_RESUME（待恢复）-任务处于待恢复状态。

出现错误时，将在为此定义的 S3 存储桶中创建错误日志报告。错误被归类为单独的数组 taskErrors 或 fileErrors 分为不同的数组。以下是错误报告示例。

```
{
    "taskId": "9367BE28418C5EF902676482220B631C",
    "taskErrors": [],
    "fileErrors": [
        {
            "fileName": "example.csv",
            "errors": [
                {
                    "reason": "The record timestamp is outside the time range of the
 data ingestion window.",
```

```
                    "lineRanges": [
                        [
                            2,
                            3
                        ]
                    ]
                }
            ]
        }
    ]
}
```

在 Timestream 中使用预定查询 LiveAnalytics

Amazon Timestream 中的计划查询功能 LiveAnalytics 是一种完全托管、无服务器且可扩展的解决方案，用于计算和存储通常用于操作控制面板、业务报告、临时分析和其他应用程序的聚合、汇总和其他形式的预处理数据。计划查询可以提高实时分析的性能和成本效益，因此您可以从数据中获得更多见解，并可以继续做出更好的业务决策。

使用计划查询，您可以定义实时分析查询，这些查询用于计算数据的聚合、汇总和其他操作，而 Amazon Timestream 会 LiveAnalytics 定期自动运行这些查询，并将查询结果可靠地写入单独的表中。数据通常会在几分钟内计算出来并更新到这些表中。

然后，您可以将仪表板和报告指向查询包含聚合数据的表，而不必查询大得多的源表。这会带来超出数量级的性能和成本收益。这是因为具有聚合数据的表所包含的数据比源表少得多，因此它们提供了更快的查询速度和更低的数据存储空间。

此外，带有计划查询的表还提供了 Timestream 的所有现有 LiveAnalytics 表功能。例如，您可以使用查询表SQL。您可以使用 Grafana 可视化存储在表中的数据。你也可以使用 Amazon Kinesis、Amazon、 AWS IoT Core 和 Teleg MSK raf 将数据提取到表中。您可以在这些表上配置数据保留策略，以实现自动数据生命周期管理。

由于包含聚合数据的表的数据保留与源表的数据保留完全分离，因此您也可以选择减少源表的数据保留时间，将聚合数据保留更长的时间，而成本仅为数据存储成本的一小部分。计划查询使实时分析更快、更便宜，因此更便于更多客户访问，因此他们可以监控自己的应用程序并推动更好的数据驱动型业务决策。

主题

- 定时查询的好处
- 计划查询用例

定时查询的好处

以下是定时查询的好处：

- 操作简便-计划查询是无服务器的，并且完全托管。
- 性能和成本-由于计划查询会预先计算数据的聚合、汇总或其他实时分析操作并将结果存储在表中，因此访问由计划查询填充的表的查询所包含的数据少于源表。因此，在这些表上运行的查询更快、更便宜。由计划计算填充的表所包含的数据少于其源表，因此有助于降低存储成本。您也可以在内存存储中将这些数据保留更长的时间，而成本只是将源数据保留在内存存储中的一小部分。
- 互操作性 — 由计划查询填充的表格提供了 Timestream 的所有现有 LiveAnalytics 表格功能，并且可以与所有与 Timestream 配合使用的服务和工具一起使用。 LiveAnalytics有关详细信息，请参阅使用其他服务。

计划查询用例

您可以将计划查询用于汇总应用程序中最终用户活动的业务报告，这样您就可以训练机器学习模型以实现个性化。您还可以使用计划查询来检测异常、网络入侵或欺诈活动的警报，以便您可以立即采取补救措施。

此外，您还可以使用计划查询来实现更有效的数据治理。为此，您可以向源表授予对计划查询的独占访问权限，并仅向开发人员提供对由计划查询填充的表的访问权限。这样可以最大限度地减少无意的、长时间运行的查询的影响。

示例：使用实时分析来检测欺诈性付款并做出更好的业务决策

以一种支付系统为例，该系统可以处理从分布在美国主要大都市的多个 point-of-sale终端发送的交易。您想使用 Amazon Timestream LiveAnalytics 来存储和分析交易数据，以便检测欺诈性交易并运行实时

分析查询。这些查询可以帮助您回答业务问题，例如确定每小时最繁忙和最少使用的 point-of-sale航站楼、每个城市一天中最繁忙的时段以及每小时交易量最多的城市。

系统每分钟处理大约 10 万笔交易。存储在 Amazon Timestream 中的每笔交易都 LiveAnalytics是 100 字节。您已经配置了 10 个查询，每分钟运行一次，以检测各种欺诈性付款。您还创建了 25 个查询，这些查询可以按不同维度对数据进行汇总和切片/切分，以帮助回答您的业务问题。这些查询中的每一个都处理最后一个小时的数据。

您已经创建了一个仪表板来显示这些查询生成的数据。仪表板包含 25 个小部件，每小时刷新一次，通常在任何给定时间都有 10 个用户访问。最后，您的内存存储配置为 2 小时的数据保留期，将磁性存储配置为有 6 个月的数据保留期。

在这种情况下，您可以使用实时分析查询，在每次访问和刷新仪表板时重新计算数据，或者使用仪表板的派生表。基于实时分析查询的仪表板的查询费用为每月 120.70 美元。相比之下，由派生表支持的仪表板查询的费用为每月12.27美元（定价参见 Amazon Timestream）。 LiveAnalytics在这种情况下，使用派生表可以将查询成本降低大约 10 倍。

定时查询的概念

查询字符串-这是您要预先计算其结果并将其存储在表的另一个 Timestream 中的查询。 LiveAnalytics 您可以使用 Timestream 的完整SQL表面区域来定义定时查询 LiveAnalytics，这使您可以灵活地使用常用表表达式、嵌套查询、窗口函数或 Timestream 为 LiveAnalytics 查询语言支持的任何类型的聚合和标量函数编写查询。

计划表达式-允许您指定计划查询实例的运行时间。您可以使用 cron 表达式（例如每天上午 8 点运行）或速率表达式（例如UTC每 10 分钟运行一次）来指定表达式。

目标配置-允许您指定如何将计划查询的结果映射到存储此计划查询结果的目标表。

通知配置-Timestream 用于根据您的计划表达式 LiveAnalytics 自动运行计划查询的实例。针对您在创建计划查询时配置的SNS主题运行的每个此类查询，您都会收到一条通知。此通知指定实例是否成功运行或遇到任何错误。此外，它还提供诸如计量字节、写入目标表的数据、下次调用时间等信息。

以下是此类通知消息的示例。

```
{
    "type":"AUTO_TRIGGER_SUCCESS",
    "arn":"arn:aws:timestream:us-east-1:123456789012:scheduled-query/
 PT1mPerMinutePerRegionMeasureCount-9376096f7309",
    "nextInvocationEpochSecond":1637302500,
    "scheduledQueryRunSummary":
```

```
    {
        "invocationEpochSecond":1637302440,
        "triggerTimeMillis":1637302445697,
        "runStatus":"AUTO_TRIGGER_SUCCESS",
        "executionStats":
        {
            "executionTimeInMillis":21669,
            "dataWrites":36864,
            "bytesMetered":13547036820,
            "recordsIngested":1200,
            "queryResultRows":1200
        }
    }
}
```

在此通知消息中，bytesMetered是查询在源表上扫描的字节， dataWrites 也是写入目标表的字节。

> **Note**
>
> 如果您以编程方式使用这些通知，请注意，将来可能会在通知消息中添加新的字段。

错误报告位置-计划查询异步运行并将数据存储在目标表中。如果实例遇到任何错误（例如，无法存储的无效数据），则遇到错误的记录将写入错误报告中，该错误报告位于您在创建计划查询时指定的错误报告位置。您可以为该位置指定 S3 存储桶和前缀。Timestream 将调度查询名称和调用时间 LiveAnalytics 附加到此前缀，以帮助您识别与计划查询的特定实例相关的错误。

标记-您可以选择指定可以与计划查询关联的标签。有关更多详细信息，请参阅为[资源标记时间流。 LiveAnalytics]

示例

在以下示例中，您使用计划查询计算简单聚合：

```
SELECT region, bin(time, 1m) as minute,
    SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m
GROUP BY bin(time, 1m), region
```

@scheduled_runtime parameter-在此示例中，您会注意到查询接受特殊命名参数@scheduled_runtime。这是服务在调用定时查询的特定实例时设置的特殊参数（类型为

Timestamp），这样您就可以确定性地控制定时查询的特定实例分析源表中数据的时间范围。你可以在查询`@scheduled_runtime`中需要使用时间戳类型的任何位置使用。

举一个你设置计划表达式的例子：cron (0/5 * *? *) 其中，计划查询将在每小时的第 0、5、10、15、20、25、30、35、40、45、50、55 分钟运行。对于在 2021-12-01 00:05:00:00 触发的实例，@scheduled_runtime 参数已初始化为该值，因此此时的实例将对 2021-11-30 23:55:00 到 2021-12-01 00:06:00 范围内的数据进行操作。

时间范围重叠的实例-正如您将在本示例中看到的那样，计划查询的两个后续实例的时间范围可能会重叠。您可以根据自己的需求、您指定的时间谓词和计划表达式来控制这一点。在这种情况下，这种重叠允许这些计算根据到达稍有延迟（在本例中最多 10 分钟）的任何数据来更新聚合。2021-12-01 00:00:00 触发的查询运行将涵盖2021-11-30 23:50:00 到 2021-12-30 00:01:00 的时间范围，2021-12-01 00:05:00 触发的查询运行将覆盖2021-11-30 23:55:00 到 2021-12-01 00:06:00 的时间范围。

为了确保正确性并确保存储在目标表中的聚合与从源表计算出的聚合相匹配，Timestream for LiveAnalytics 确保只有在 2021-12-01 00:00:00:00 的计算完成后才会执行 2021-12-01 00:05:00 的计算。如果生成了较新的值，则后一种计算的结果可以更新任何先前实现的聚合。在内部，Timestream for LiveAnalytics 使用记录版本，其中由计划查询的后一个实例生成的记录将被分配更高的版本号。因此，假设源表中有更新的数据，则在 2021-12-01 00:05:00 通过调用计算的聚合可以在 2021-12-01 00:00:00:00 更新通过调用计算的聚合。

自动触发器与手动触发器-创建计划查询后，Timestream LiveAnalytics 将根据指定的计划自动运行实例。此类自动触发器完全由该服务管理。

但是，在某些情况下，您可能需要手动启动计划查询的某些实例。例如，如果特定实例在查询运行中失败，自动调度运行后源表中是否有延迟到达的数据或更新，或者您是否要针对自动查询运行未涵盖的时间范围（例如，针对创建计划查询之前的时间范围）更新目标表。

您可以使用通过传递参数（用于 @scheduled_runtime InvocationTime 参数的值）来手动启动计划查询的特定实例。 ExecuteScheduledQuery API以下是使用时的一些重要注意事项 ExecuteScheduledQuery API：

- 如果您要触发多个这样的调用，则需要确保这些调用不会在重叠的时间范围内生成结果。如果您无法确保时间范围不重叠，请确保这些查询按顺序依次启动。如果您同时启动多个在时间范围内重叠的查询运行，则可以看到触发器失败，您可能会在这些查询运行的错误报告中看到版本冲突。
- 你可以使用 @scheduled_runtime 的任何时间戳值启动调用。因此，您有责任适当地设置值，以便在目标表中更新相应的时间范围，与源表中数据的更新范围相对应。

计划查询的计划表达式

对于使用 cron 或速率表达式的计划查询，您可以使用 Amazon Timestream 按自动 LiveAnalytics 计划创建计划查询。所有计划查询都使用UTC时区，计划的最低可能精度为 1 分钟。

指定计划表达式的两种方法是 cron 和 rat e。Cron 表达式提供了更精细的日程控制，而速率表达式更易于表达，但缺乏细粒度的控制。

例如，使用 cron 表达式，您可以定义一个计划查询，该查询在每周或每月的某一天的指定时间触发，或者仅在周一至周五每小时的指定分钟触发，依此类推。相比之下，速率表达式以常规速率启动定时查询，例如每分钟、每小时或每天一次，从创建计划查询的确切时间开始。

Cron 表达式

- 语法

```
cron(fields)
```

Cron 表达式有六个必填字段，之间以空格分隔。

字段	值	通配符
分钟	0-59	, - * /
小时	0-23	, - * /
D ay-of-month	1-31	, - * ? / L W
月	1-12 或 JAN-DEC	, - * /
D ay-of-week	1-7 或 SUN-SAT	, - * ? L #
年	1970-2199	, - * /

通配符

- * ，* （逗号）通配符包括其他值。在 “月” 字段中JAN，FEB，MAR将包括一月、二月和三月。
- *-* （破折号）通配符指定范围。在“日”字段中，1-15 将包含指定月份的 1 - 15 日。

- *** （星号）通配符包括该字段中的所有值。在 “小时” 字段中，*** 将包括每小时。不能同时在Day-of-month和 Day-of-week字段中使用***。如果您将其合而为一，则必须使用*？ * 在另一个。
- */* （正斜杠）通配符指定增量。在 “分钟” 字段中，可以输入 1/10 来指定每隔 10 分钟，从一小时的第一分钟开始（例如，第 11、21 和 31 分钟，依此类推）。
- *？ *（问号）通配符指定一个或另一个。在 Day-of-month字段中你可以输入 *7* 如果你不在乎第 7 天是哪一天，你可以输入 *？ * 在现 Day-of-week场。
- Day-of-month或 Day-of-week字段中的*L* 通配符指定一个月或一周的最后一天。
- Day-of-month字段中的 W 通配符指定工作日。在该 Day-of-month字段中，3W 指定最接近该月第三天的工作日。
- 该 Day-of-week字段中的*#* 通配符指定一个月内一周中指定某一天的特定实例。例如，3#2 指该月的第二个星期二：3 指的是星期二，因为它是每周的第三天，2 是指该月内该类型的第二天。

> **Note**
>
> 如果使用 '#' 字符，则只能在 day-of-week字段中定义一个表达式。例如，“3#1,6#3”是无效的，因为它被解释为两个表达式。

限制

- 您不能在同一 cron 表达式中指定 Day-of-month和 Day-of-week字段。如果您在其中一个字段中指定值（或*），则必须使用*？ *（问号）在另一个中。
- 不支持产生的速率快于 1 分钟的 Cron 表达式。

示例

分钟	小时	日期	月份	星期几	年	含义
0	10	*	*	?	*	每天上午 10:00 跑步 (UTC)。
15	12	*	*	?	*	每天下午 12:15 跑步 (UTC)。

分钟	小时	日期	月份	星期几	年	含义
0	18	?	*	MON-FRI	*	每周一至周五下午 6:00 (UTC) 跑步。
0	8	1	*	?	*	每月第一天上午 8:00 (UTC) 跑步。
0/15	*	*	*	?	*	每 15 分钟运行一次。
0/10	*	*	*	MON-FRI	*	周一至周五每 10 分钟跑一次。
0/5	8-17	?	*	MON-FRI	*	周一至周五上午 8:00 至下午 5:55 之间，每 5 分钟运行一次 () UTC。

Rate 表达式

- Rate 表达式在创建计划事件规则时启动，然后按照其定义的计划运行。Rate 表达式有两个必需字段。这些字段用空格分隔。

  语法

```
rate(value unit)
```

- `value`: 正数。
- `unit`: 时间单位。值为 1（例如，分钟）和大于 1 的值（例如，分钟）需要不同的单位。有效值：minute | minutes | hour | hours | day | days

计划查询的数据模型映射

Timestream for LiveAnalytics 支持对其表中的数据进行灵活建模，同样的灵活性也适用于在表格的另一个 Timestream 中实现的定时查询的结果。 LiveAnalytics 通过计划查询，您可以查询任何表，无论该表包含多度量记录还是单度量记录中的数据，并使用多度量记录或单度量记录写入查询结果。

您可以在定时查询的规范 TargetConfiguration 中使用将查询结果映射到目标派生表中的相应列。以下各节描述了指定此值 TargetConfiguration 以在派生表中实现不同数据模型的不同方法。具体而言，你会看到：

- 当查询结果没有度量名称并且您在中指定了目标度量名称时，如何写入多度量记录。 TargetConfiguration
- 如何在查询结果中使用度量名称来写入多度量记录。
- 如何定义一个模型来写入具有不同多度量属性的多条记录。
- 如何定义模型以写入派生表中的单度量记录。
- 如何在计划查询中查询单度量记录和/或多度量记录并将结果具体化为单度量记录或多度量记录，这使您可以灵活选择数据模型的灵活性。

示例：多度量记录的目标度量名称

在此示例中，您将看到查询正在从包含多度量数据的表中读取数据，并使用多度量记录将结果写入另一个表。计划查询结果没有自然度量名称列。在这里，您可以使用中的 TargetMultiMeasureName 属性在派生表中指定度量名称 TargetConfiguration。 TimestreamConfiguration。

```
{
    "Name" : "CustomMultiMeasureName",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(memory_cached)
 as avg_mem_cached_1h, MIN(memory_free) as min_mem_free_1h, MAX(memory_used) as
 max_mem_used_1h, SUM(disk_io_writes) as sum_1h, AVG(disk_used) as avg_disk_used_1h,
 AVG(disk_free) as avg_disk_free_1h, MAX(cpu_user) as max_cpu_user_1h, MIN(cpu_idle) as
```

```
min_cpu_idle_1h, MAX(cpu_system) as max_cpu_system_1h FROM raw_data.devops_multi WHERE
time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h
AND measure_name = 'metrics' GROUP BY region, bin(time, 1h)",
   "ScheduleConfiguration" : {
       "ScheduleExpression" : "cron(0 0/1 * * ? *)"
   },
   "NotificationConfiguration" : {
       "SnsConfiguration" : {
           "TopicArn" : "******"
       }
   },
   "ScheduledQueryExecutionRoleArn": "******",
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName" : "derived",
           "TableName" : "dashboard_metrics_1h_agg_1",
           "TimeColumn" : "hour",
           "DimensionMappings" : [
               {
                   "Name": "region",
                   "DimensionValueType" : "VARCHAR"
               }
           ],
           "MultiMeasureMappings" : {
               "TargetMultiMeasureName": "dashboard-metrics",
               "MultiMeasureAttributeMappings" : [
                   {
                       "SourceColumn" : "avg_mem_cached_1h",
                       "MeasureValueType" : "DOUBLE",
                       "TargetMultiMeasureAttributeName" : "avgMemCached"
                   },
                   {
                       "SourceColumn" : "min_mem_free_1h",
                       "MeasureValueType" : "DOUBLE"
                   },
                   {
                       "SourceColumn" : "max_mem_used_1h",
                       "MeasureValueType" : "DOUBLE"
                   },
                   {
                       "SourceColumn" : "sum_1h",
                       "MeasureValueType" : "DOUBLE",
                       "TargetMultiMeasureAttributeName" : "totalDiskWrites"
                   },
```

```
                    {
                        "SourceColumn" : "avg_disk_used_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "avg_disk_free_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_cpu_user_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "CpuUserP100"
                    },
                    {
                        "SourceColumn" : "min_cpu_idle_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_cpu_system_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "CpuSystemP100"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

本示例中的映射创建了一条包含度量名称仪表板指标和属性名称的多度量记录，min_mem_free_1h、max_mem_used_1h、、avg_disk_used_1h avgMemCached、avg_disk_free_1h、P100、min_cpu_idle_1h、P100。 totalDiskWrites CpuUser CpuSystem请注意，可以选择使用 TargetMultiMeasureAttributeName 将查询输出列重命名为用于结果实现的不同属性名称。

以下是实现此定时查询后目标表的架构。如以下结果中 LiveAnalytics 属性类型的 Timestream 所示，结果将具体化为具有单度量名称的多度量记录dashboard-metrics，如度量架构所示。

列	类型	LiveAnalytics 属性类型的时间流
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
时间	时间戳	TIMESTAMP
CpuSystemP100	double	MULTI
avgMemCached	double	MULTI
min_cpu_idle_1h	double	MULTI
avg_disk_free_1h	double	MULTI
avg_disk_used_1h	double	MULTI
totalDiskWrites	double	MULTI
max_mem_used_1h	double	MULTI
min_mem_free_1h	double	MULTI
CpuUserP100	double	MULTI

以下是通过SHOWMEASURES查询获得的相应度量。

measure_name	data_type	尺寸
仪表板指标	多	[{'dimension_name': 'region'，'data_type'：'varchar'}]

示例：在多度量记录中使用计划查询中的度量名称

在此示例中，您将看到一个查询从包含单度量记录的表中读取数据，并将结果具体化为多度量记录。在这种情况下，调度查询结果中有一列，该列的值可用作目标表中的度量名称，而定时查询的结果将在该

表中实现定时查询的结果。然后，您可以使用中的 MeasureNameColumn 属性为派生表中的多度量记录指定度量名称。 TargetConfiguration TimestreamConfiguration。

```
{
    "Name" : "UsingMeasureNameFromQueryResult",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, measure_name, AVG(CASE WHEN
 measure_name IN ('memory_cached', 'disk_used', 'disk_free') THEN measure_value::double
 ELSE NULL END) as avg_1h, MIN(CASE WHEN measure_name IN ('memory_free', 'cpu_idle')
 THEN measure_value::double ELSE NULL END) as min_1h, SUM(CASE WHEN measure_name
 IN ('disk_io_writes') THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name IN ('memory_used', 'cpu_user', 'cpu_system') THEN
 measure_value::double ELSE NULL END) as max_1h FROM raw_data.devops WHERE time
 BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND
 measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
 'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
 measure_name, bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_2",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MeasureNameColumn" : "measure_name",
            "MultiMeasureMappings" : {
                "MultiMeasureAttributeMappings" : [
                    {
                        "SourceColumn" : "avg_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
```

```
                    {
                        "SourceColumn" : "min_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "p0_1h"
                    },
                    {
                        "SourceColumn" : "sum_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "p100_1h"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

本示例中的映射将创建具有属性 avg_1h、p0_1h、sum_1h、p100_1h 的多度量记录，并将使用查询结果中 measure_name 列的值作为目标表中多度量记录的度量名称。此外，请注意，前面的示例可以选择使用 TargetMultiMeasureAttributeName 带有映射子集的来重命名属性。例如，min_1h 被重命名为 p0_1h，max_1h 重命名为 p100_1h。

以下是实现此定时查询后目标表的架构。正如您在以下结果中从 LiveAnalytics 属性类型的时间流中看到的那样，结果被具体化为多度量记录。如果您查看度量架构，则提取了九个不同的度量名称，它们与查询结果中看到的值相对应。

列	类型	LiveAnalytics 属性类型的时间流
region	varchar	DIMENSION

列	类型	LiveAnalytics 属性类型的时间流
measure_name	varchar	MEASURE_NAME
时间	时间戳	TIMESTAMP
sum_1h	double	MULTI
p100_1h	double	MULTI
p0_1h	double	MULTI
avg_1h	double	MULTI

以下是通过SHOWMEASURES查询获得的相应度量。

measure_name	data_type	尺寸
cpu_idle	多	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
CPU_system	多	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
cpu_user	多	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
disk_free	多	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
disk_io_writes	多	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
disk_used	多	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
内存缓存	多	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]

measure_name	data_type	尺寸
内存_免费	多	[{'dimension_name': 'region'，'data_type'：'varchar'}]
内存_免费	多	[{'dimension_name': 'region'，'data_type'：'varchar'}]

示例：将结果映射到具有不同属性的不同多指标记录

以下示例说明如何将查询结果中的不同列映射到具有不同度量名称的不同多度量记录中。如果你看到以下定时查询定义，则查询结果包含以下列：区域、小时、avg_mem_cached_1h、min_mem_free_1h、max_mem_used_1h、total_disk_io_writes_1h、avg_disk_fr
region映射到维度hour，并映射到时间列。

中的 MixedMeasureMappings 房产 TargetConfiguration。 TimestreamConfiguration指定如何将度量映射到派生表中的多度量记录。

在这个具体的示例中，在一条多度量记录中使用 avg_mem_cached_1h、min_mem_free_1h、max_mem_used_1h，度量名称为 mem_aggregates，total_disk_writes_1h，avg_disk_free_1h 用于另一条度量名称为 disk_aggregates 的多度量记录中，最后 max_cpu_user_1h、max_cpu_system_1h、min_cpu_system_1h 被用于另一条度量名称为 cpu_aggregates 的多度量记录中。

在这些映射中，您还可以选择使用 TargetMultiMeasureAttributeName 重命名查询结果列，使其在目标表中使用不同的属性名称。例如，结果列 avg_mem_cached_1h 被重命名为，total_disk_io_writes_1h 被重命名为，等等。 avgMemCached totalIOWrites

在为多度量记录定义映射时， LiveAnalytics Timestream for 会检查查询结果中的每一行，并自动忽略具有值的列值。NULL因此，对于具有多个度量名称的映射，如果映射中该组的所有列值都NULL针对给定行，则不会为该行提取该度量名称的值。

例如，在以下映射中，avg_mem_cached_1h、min_mem_free_1h 和 max_mem_used_1h 映射到度量名称 mem_aggrates。如果对于查询结果的给定行，所有这些列值都NULL是，则 Timestream for 将 LiveAnalytics 不会提取该行的度量 mem_aggregates。如果给定行的所有九列都是NULL，那么您将在错误报告中看到用户错误报告。

```
{
    "Name" : "AggsInDifferentMultiMeasureRecords",
```

```
   "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(CASE WHEN measure_name
= 'memory_cached' THEN measure_value::double ELSE NULL END) as avg_mem_cached_1h,
MIN(CASE WHEN measure_name = 'memory_free' THEN measure_value::double ELSE
NULL END) as min_mem_free_1h, MAX(CASE WHEN measure_name = 'memory_used' THEN
measure_value::double ELSE NULL END) as max_mem_used_1h, SUM(CASE WHEN measure_name =
'disk_io_writes' THEN measure_value::double ELSE NULL END) as total_disk_io_writes_1h,
AVG(CASE WHEN measure_name = 'disk_used' THEN measure_value::double ELSE NULL END) as
avg_disk_used_1h, AVG(CASE WHEN measure_name = 'disk_free' THEN measure_value::double
ELSE NULL END) as avg_disk_free_1h, MAX(CASE WHEN measure_name = 'cpu_user' THEN
measure_value::double ELSE NULL END) as max_cpu_user_1h, MAX(CASE WHEN measure_name
= 'cpu_system' THEN measure_value::double ELSE NULL END) as max_cpu_system_1h,
MIN(CASE WHEN measure_name = 'cpu_idle' THEN measure_value::double ELSE NULL END)
as min_cpu_system_1h FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime,
1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND measure_name IN ('memory_cached',
'memory_free', 'memory_used', 'disk_io_writes', 'disk_used', 'disk_free', 'cpu_user',
'cpu_system', 'cpu_idle') GROUP BY region, bin(time, 1h)",
   "ScheduleConfiguration" : {
       "ScheduleExpression" : "cron(0 0/1 * * ? *)"
   },
   "NotificationConfiguration" : {
       "SnsConfiguration" : {
           "TopicArn" : "******"
       }
   },
   "ScheduledQueryExecutionRoleArn": "******",
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName" : "derived",
           "TableName" : "dashboard_metrics_1h_agg_3",
           "TimeColumn" : "hour",
           "DimensionMappings" : [
               {
                   "Name": "region",
                   "DimensionValueType" : "VARCHAR"
               }
           ],
           "MixedMeasureMappings" : [
               {
                   "MeasureValueType" : "MULTI",
                   "TargetMeasureName" : "mem_aggregates",
                   "MultiMeasureAttributeMappings" : [
                       {
                           "SourceColumn" : "avg_mem_cached_1h",
                           "MeasureValueType" : "DOUBLE",
```

```
                        "TargetMultiMeasureAttributeName": "avgMemCached"
                    },
                    {
                        "SourceColumn" : "min_mem_free_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_mem_used_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "maxMemUsed"
                    }
                ]
            },
            {
                "MeasureValueType" : "MULTI",
                "TargetMeasureName" : "disk_aggregates",
                "MultiMeasureAttributeMappings" : [
                    {
                        "SourceColumn" : "total_disk_io_writes_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "totalIOWrites"
                    },
                    {
                        "SourceColumn" : "avg_disk_used_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "avg_disk_free_1h",
                        "MeasureValueType" : "DOUBLE"
                    }
                ]
            },
            {
                "MeasureValueType" : "MULTI",
                "TargetMeasureName" : "cpu_aggregates",
                "MultiMeasureAttributeMappings" : [
                    {
                        "SourceColumn" : "max_cpu_user_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_cpu_system_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
```

```
                        {
                            "SourceColumn" : "min_cpu_idle_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "minCpuIdle"
                        }
                    ]
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

以下是实现此定时查询后目标表的架构。

列	类型	LiveAnalytics 属性类型的时间流
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
时间	时间戳	TIMESTAMP
minCpuIdle	double	MULTI
max_cpu_system_1h	double	MULTI
max_cpu_user_1h	double	MULTI
avgMemCached	double	MULTI
maxMemUsed	double	MULTI
min_mem_free_1h	double	MULTI

列	类型	LiveAnalytics 属性类型的时间流
avg_disk_free_1h	double	MULTI
avg_disk_used_1h	double	MULTI
totalIOWrites	double	MULTI

以下是通过SHOWMEASURES查询获得的相应度量。

measure_name	data_type	尺寸
cpu_Aggregates	多	[{'dimension_name': 'region'，'data_type'：'varchar'}]
磁盘聚合	多	[{'dimension_name': 'region'，'data_type'：'varchar'}]
mem_aggregates	多	[{'dimension_name': 'region'，'data_type'：'varchar'}]

示例：将结果映射到带有查询结果中度量名称的单度量记录

以下是一个定时查询的示例，其结果将具体化为单一测量记录。在此示例中，查询结果具有 measure_name 列，其值将用作目标表中的度量名称。您可以在中使用该 MixedMeasureMappings 属性 TargetConfiguration。 TimestreamConfiguration 指定查询结果列与目标表中标量度量的映射。

在以下示例定义中，查询结果应为九个不同的 measure_name 值。您可以在映射中列出所有这些度量名称，并指定要使用哪一列作为该度量名称的单度量值。例如，在此映射中，如果看到给定结果行的 memory_cached 的度量名称，则在将数据写入目标表时，avg_1h 列中的值将用作度量的值。您可以选择使用 TargetMeasureName 为该值提供新的度量名称。

```
{
    "Name" : "UsingMeasureNameColumnForSingleMeasureMapping",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, measure_name, AVG(CASE WHEN
 measure_name IN ('memory_cached', 'disk_used', 'disk_free') THEN measure_value::double
 ELSE NULL END) as avg_1h, MIN(CASE WHEN measure_name IN ('memory_free', 'cpu_idle')
```

```
THEN measure_value::double ELSE NULL END) as min_1h, SUM(CASE WHEN measure_name
IN ('disk_io_writes') THEN measure_value::double ELSE NULL END) as sum_1h,
MAX(CASE WHEN measure_name IN ('memory_used', 'cpu_user', 'cpu_system') THEN
measure_value::double ELSE NULL END) as max_1h FROM raw_data.devops WHERE time
BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND
measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
bin(time, 1h), measure_name",
   "ScheduleConfiguration" : {
       "ScheduleExpression" : "cron(0 0/1 * * ? *)"
   },
   "NotificationConfiguration" : {
       "SnsConfiguration" : {
           "TopicArn" : "******"
       }
   },
   "ScheduledQueryExecutionRoleArn": "******",
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName" : "derived",
           "TableName" : "dashboard_metrics_1h_agg_4",
           "TimeColumn" : "hour",
           "DimensionMappings" : [
               {
                   "Name": "region",
                   "DimensionValueType" : "VARCHAR"
               }
           ],
           "MeasureNameColumn" : "measure_name",
           "MixedMeasureMappings" : [
               {
                   "MeasureName" : "memory_cached",
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "avg_1h",
                   "TargetMeasureName" : "AvgMemCached"
               },
               {
                   "MeasureName" : "disk_used",
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "avg_1h"
               },
               {
                   "MeasureName" : "disk_free",
                   "MeasureValueType" : "DOUBLE",
```

```
                    "SourceColumn" : "avg_1h"
                },
                {
                    "MeasureName" : "memory_free",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_1h",
                    "TargetMeasureName" : "MinMemFree"
                },
                {
                    "MeasureName" : "cpu_idle",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_1h"
                },
                {
                    "MeasureName" : "disk_io_writes",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "sum_1h",
                    "TargetMeasureName" : "total-disk-io-writes"
                },
                {
                    "MeasureName" : "memory_used",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h",
                    "TargetMeasureName" : "maxMemUsed"
                },
                {
                    "MeasureName" : "cpu_user",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h"
                },
                {
                    "MeasureName" : "cpu_system",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h"
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
```

```
    }
}
```

以下是实现此定时查询后目标表的架构。从架构中可以看出，该表使用的是单度量记录。如果您列出表的度量架构，您将看到根据规范中提供的映射写入的九个度量。

列	类型	LiveAnalytics 属性类型的时间流
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
时间	时间戳	TIMESTAMP
measure_value::double	double	MEASURE_VALUE

以下是通过SHOWMEASURES查询获得的相应度量。

measure_name	data_type	尺寸
AvgMemCached	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
MinMemFree	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
cpu_idle	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
CPU_system	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
cpu_user	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
disk_free	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]

measure_name	data_type	尺寸
disk_used	double	[{'dimension_name': 'region'，'data_type'：'varchar'}]
maxMemUsed	double	[{'dimension_name': 'region'，'data_type'：'varchar'}]
total-disk-io-writes	double	[{'dimension_name': 'region'，'data_type'：'varchar'}]

示例：将结果映射到以查询结果列作为度量名称的单度量记录

在此示例中，您的查询的结果没有度量名称列。相反，在将输出映射到单度量记录时，您希望将查询结果列名称作为度量名称。前面有一个例子，将类似的结果写入多度量记录中。在此示例中，如果符合您的应用场景，您将看到如何将其映射到单度量记录。

同样，您可以使用中的 MixedMeasureMappings 属性来指定此映射 TargetConfiguration。TimestreamConfiguration。在以下示例中，您会看到查询结果有九列。您可以使用结果列作为度量名称，将值用作单度量值。

例如，对于查询结果中的给定行，列名 avg_mem_cached_1h 用作与列关联的列名和值，avg_mem_cached_1h 用作单度量记录的度量值。您也可以使用 TargetMeasureName 在目标表中使用不同的度量名称。例如，对于 sum_1h 列中的值，映射指定使用 total_disk_io_writes_1h 作为目标表中的度量名称。如果任何列的值为NULL，则忽略相应的度量。

```
{
    "Name" : "SingleMeasureMappingWithoutMeasureNameColumnInQueryResult",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(CASE WHEN measure_name
 = 'memory_cached' THEN measure_value::double ELSE NULL END) as avg_mem_cached_1h,
 AVG(CASE WHEN measure_name = 'disk_used' THEN measure_value::double ELSE NULL END) as
 avg_disk_used_1h, AVG(CASE WHEN measure_name = 'disk_free' THEN measure_value::double
 ELSE NULL END) as avg_disk_free_1h, MIN(CASE WHEN measure_name = 'memory_free' THEN
 measure_value::double ELSE NULL END) as min_mem_free_1h, MIN(CASE WHEN measure_name =
 'cpu_idle' THEN measure_value::double ELSE NULL END) as min_cpu_idle_1h, SUM(CASE WHEN
 measure_name = 'disk_io_writes' THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name = 'memory_used' THEN measure_value::double ELSE NULL END)
 as max_mem_used_1h, MAX(CASE WHEN measure_name = 'cpu_user' THEN measure_value::double
 ELSE NULL END) as max_cpu_user_1h, MAX(CASE WHEN measure_name = 'cpu_system' THEN
 measure_value::double ELSE NULL END) as max_cpu_system_1h FROM raw_data.devops WHERE
```

```
time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h
AND measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
bin(time, 1h)",
   "ScheduleConfiguration" : {
       "ScheduleExpression" : "cron(0 0/1 * * ? *)"
   },
   "NotificationConfiguration" : {
       "SnsConfiguration" : {
           "TopicArn" : "******"
       }
   },
   "ScheduledQueryExecutionRoleArn": "******",
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName" : "derived",
           "TableName" : "dashboard_metrics_1h_agg_5",
           "TimeColumn" : "hour",
           "DimensionMappings" : [
               {
                   "Name": "region",
                   "DimensionValueType" : "VARCHAR"
               }
           ],
           "MixedMeasureMappings" : [
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "avg_mem_cached_1h"
               },
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "avg_disk_used_1h"
               },
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "avg_disk_free_1h"
               },
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "min_mem_free_1h"
               },
               {
                   "MeasureValueType" : "DOUBLE",
                   "SourceColumn" : "min_cpu_idle_1h"
```

```
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "sum_1h",
                    "TargetMeasureName" : "total_disk_io_writes_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_mem_used_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_cpu_user_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_cpu_system_1h"
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

以下是实现此定时查询后目标表的架构。如您所见，目标表正在存储双精度量值为双精度的记录。同样，该表的度量架构显示了九个度量名称。另请注意，由于映射将 sum_1h 重命名为 total_disk_io_writes_1h，因此存在度量名称 total_disk_io_writes_1h。

列	类型	LiveAnalytics 属性类型的时间流
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
时间	时间戳	TIMESTAMP

列	类型	LiveAnalytics 属性类型的时间流
measure_value::double	double	MEASURE_VALUE

以下是通过SHOWMEASURES查询获得的相应度量。

measure_name	data_type	尺寸
avg_disk_free_1h	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
avg_disk_used_1h	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
avg_mem_cached_1h	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
max_cpu_system_1h	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
max_cpu_user_1h	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
max_mem_used_1h	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
min_cpu_idle_1h	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
min_mem_free_1h	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]
total-disk-io-writes	double	[{'dimension_name': 'region' , 'data_type' : 'varchar'}]

预设查询通知消息

本节介绍创建、删除、运行或更新计划查询状态 LiveAnalytics 时由 Timestream 发送的消息。

通知消息名称	结构	描述
CreatingNotificationMessage	`CreatingNotificationMessage { String arn; NotificationType type; }`	此通知消息是在发送回复之前发送的`CreateScheduledQuery` 。发送此通知后，将启用计划查询。 arn-正在创建ARN的计划查询的。 类型-SCHEDULED _ QUERY _ CREATING
UpdateNotificationMessage	`UpdateNotificationMessage { String arn; NotificationType type; QueryState state; }`	此通知消息是在计划查询更新时发送的。Timestream for LiveAnalytics 可以自动禁用定时查询，以防遇到不可恢复的错误，例如： • AssumeRole 失败 • 指定客户托管KMS密钥时与通信KMS时遇到的任何 4xx 错误。 • 运行计划查询期间遇到的任何 4xx 错误。 • 在提取查询结果期间遇到的任何 4xx 错误 arn-正在更新的计划查询的。ARN 类型-SCHEDULED _ QUERY _ UPDATE

通知消息名称	结构	描述
		州-ENABLED 或 DISABLED
DeleteNotificationMessage	`DeletionNotificationMessage { String arn; NotificationType type; }`	此通知消息是在计划查询被删除后发送的。 arn-正在创建ARN的计划查询的。 类型-SCHEDULED _ QUERY _ DELETED

通知消息名称	结构	描述

SuccessNotificationMessage	SuccessNotificationMessage {
	    NotificationType type;
	    String arn;
	    Date nextInvocationEpochSecond;
	    ScheduledQueryRunSummary runSummary;
	}
	
	ScheduledQueryRunSummary {
	    Date invocationTime;
	    Date triggerTime;
	    String runStatus;
	    ExecutionStats executionstats;
	    ErrorReportLocation errorReportLocation;
	    String failureReason;
	}
	
	ExecutionStats {
	    Long bytesMetered;
	    Long dataWrites;
	    Long queryResultRows;
	    Long recordsIngested;
	    Long executionTimeInMillis;
	}
	
	ErrorReportLocation {
		此通知消息是在运行计划查询并成功提取结果之后发送的。
		
		ARN-正在删除ARN的计划查询。
		
		NotificationType-AUTO _ TRIGGER _ SUCCESS 或 MANUAL _ TRIGGER _ SUCCESS。
		
		nextInvocationEpoch第二-下次 Timestream LiveAnalytics 将运行计划查询。
		
		runSummary-有关计划查询运行的信息。

通知消息名称	结构	描述

	S3ReportLocation s3ReportLocation;
	}
	
	S3ReportLocation {
	String bucketName;
	String objectKey;
	}	

通知消息名称	结构	描述

FailureNotificationMessage	FailureNotificationMessage {
	    NotificationType type;
	    String arn;
	    ScheduledQueryRunSummary runSummary;
	}
	
	ScheduledQueryRunSummary {
	    Date invocationTime;
	    Date triggerTime;
	    String runStatus;
	    ExecutionStats executionstats;
	    ErrorReportLocation errorReportLocation;
	    String failureReason;
	}
	
	ExecutionStats {
	    Long bytesMetered;
	    Long dataWrites;
	    Long queryResultRows;
	    Long recordsIngested;
	    Long executionTimeInMillis;
	}
	
	ErrorReportLocation {
	    S3ReportLocation s3ReportLocation;
	}
		在计划查询运行期间或获取查询结果时遇到故障时，会发送此通知消息。
		
		arn-正在运行ARN的计划查询的。
		
		键入-AUTO TRIGGER _ _ FAILURE 或 MANUAL _ TRIGGER _ FAILURE。
		
		runSummary-有关计划查询运行的信息。

通知消息名称	结构	描述
	`S3ReportLocation { String bucketName; String objectKey; }`	

预设查询错误报告

本节介绍运行计划查询遇到错误时由 Timestream 生成的错误报告的位置、格式和原因 LiveAnalytics 。

主题

- 计划查询错误报告原因
- 预设查询错误报告位置
- 计划查询错误报告格式
- 计划查询错误类型
- 计划查询错误报告示例

计划查询错误报告原因

为可恢复的错误生成错误报告。对于不可恢复的错误，不会生成错误报告。遇到不可恢复的错误时，Timestream for LiveAnalytics 可以自动禁用计划查询。其中包括：

- `AssumeRole`失败
- 指定客户管理的KMS密钥时与KMS通信时遇到的任何 4xx 错误
- 运行计划查询时遇到的任何 4xx 错误
- 在提取查询结果期间遇到的任何 4xx 错误

对于不可恢复的错误，Timestream for LiveAnalytics 会发送包含不可恢复错误消息的失败通知。还会发送更新通知，表示计划查询已禁用。

预设查询错误报告位置

计划查询错误报告位置的命名约定如下：

```
s3://customer-bucket/customer-prefix/
```

以下是计划查询的示例ARN：

```
arn:aws:timestream:us-east-1:000000000000:scheduled-query/test-query-hd734tegrgfd
```

```
s3://customer-bucket/customer-prefix/test-query-hd734tegrgfd/<InvocationTime>/<Auto or
 Manual>/<Actual Trigger Time>
```

Auto 表示 Timestream 为和自动安排的 LiveAnalytics 计划查询 *Manual* 表示用户通过 Amazon Timestream 查询中的`ExecuteScheduledQuery`API操作手动触发的 LiveAnalytics 计划查询。有关的更多信息`ExecuteScheduledQuery`，请参阅ExecuteScheduledQuery。

计划查询错误报告格式

错误报告采用以下JSON格式：

```
{
    "reportId": <String>,            // A unique string ID for all error reports
 belonging to a particular scheduled query run
    "errors": [ <Error>, ... ],      // One or more errors
}
```

计划查询错误类型

该`Error`对象可以是以下三种类型之一：

- 记录摄取错误

```
{
    "reason": <String>,              // The error message String
    "records": [ <Record>, ... ],    // One or more rejected records )
}
```

- 行解析和验证错误

```
{
```

```
    "reason": <String>,        // The error message String
    "rawLine": <String>,       // [Optional] The raw line String that is being parsed
 into record(s) to be ingested. This line has encountered the above-mentioned parse
 error.
}
```

- 一般错误

```
{
    "reason": <String>,        // The error message
}
```

计划查询错误报告示例

以下是由于摄取错误而生成的错误报告的示例。

```
{
    "reportId": "C9494AABE012D1FBC162A67EA2C18255",
    "errors": [
        {
            "reason": "The record timestamp is outside the time range
 [2021-11-12T14:18:13.354Z, 2021-11-12T16:58:13.354Z) of the memory store.",
            "records": [
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_1",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_1",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "3.141592653589793",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175635000000",
                    "timeUnit": "NANOSECONDS",
```

```
                    "version": null
                },
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_2",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_2",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "6.283185307179586",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175636000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                },
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_3",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_3",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "9.42477796076938",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175637000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                },
```

```
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_4",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_4",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "12.566370614359172",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175638000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                }
            ]
        }
    ]
}
```

计划查询模式和示例

本节介绍定时查询的使用模式和示 end-to-end例。

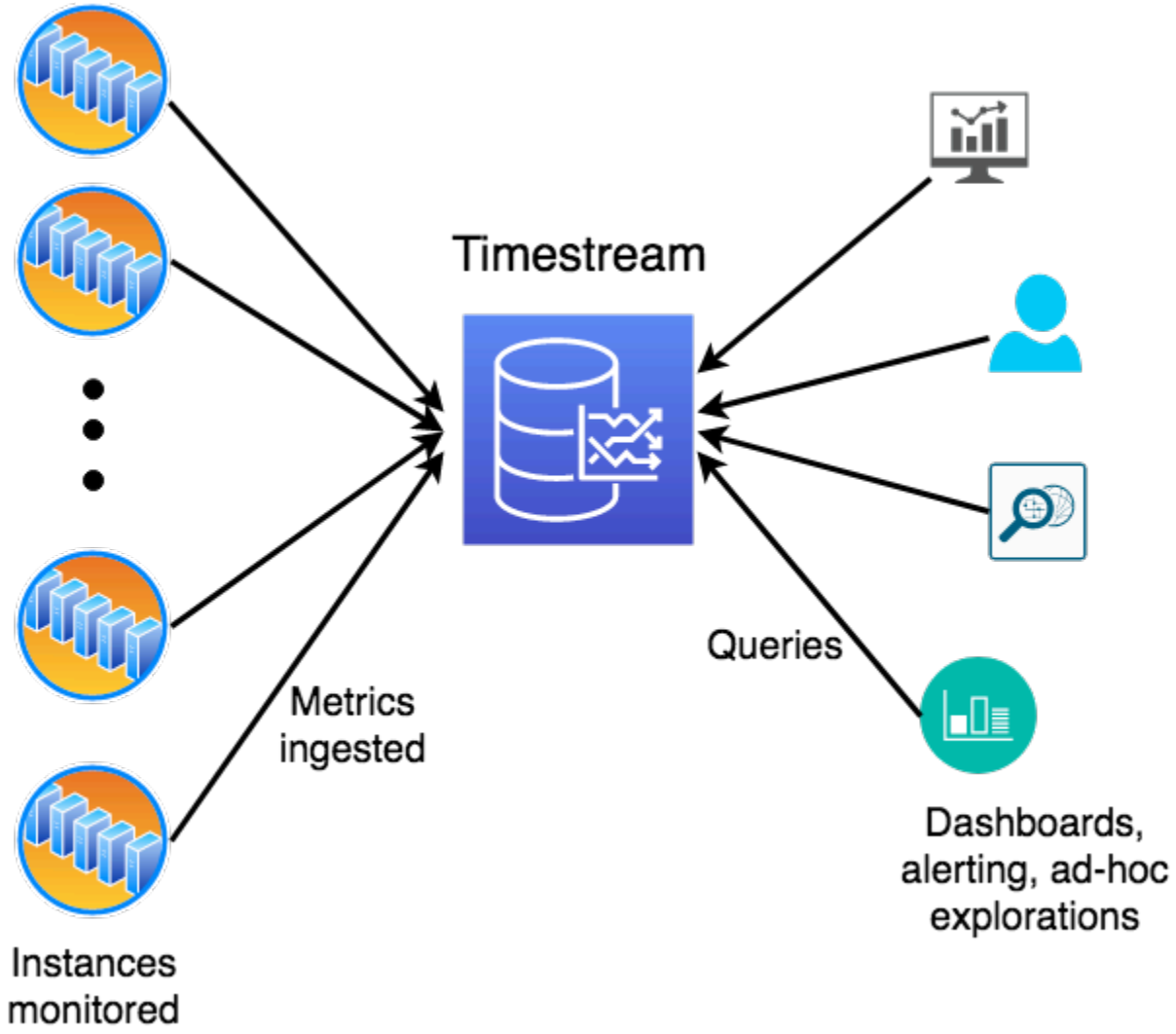
主题

- 计划查询示例架构
- 计划查询模式
- 定时查询示例

计划查询示例架构

在此示例中，我们将使用一个示例应用程序，该应用程序模仿来自大量服务器的 DevOps 场景监控指标。用户希望就异常资源使用情况发出警报，创建有关聚合车队行为和利用率的仪表板，并对最近和

历史数据进行复杂的分析以找出相关性。下图说明了一组受监控实例向 Timestream 发送指标的设置。LiveAnalytics另一组并发用户对警报、仪表板或临时分析发出查询，其中查询和摄取并行运行。



所监控的应用程序被建模为一项高度扩展的服务，部署在全球多个地区。每个区域进一步细分为多个缩放单元，称为单元，这些单元在区域内的基础设施方面具有一定程度的隔离。每个单元进一步细分为孤岛，这代表了软件隔离的程度。每个思洛存储器都有五个微服务，这些微服务构成了一个孤立的服务实例。每个微服务都有几台服务器，这些服务器具有不同的实例类型和操作系统版本，这些服务器部署在三个可用区中。这些标识发出指标的服务器的属性在 Timestream 中建模为的维度。 LiveAnalytics在这个架构中，我们有一个维度层次结构（例如区域、单元、筒仓和 microservice_name）和其他跨越层次结构的维度（例如 instance_type 和 availability_zone）。

该应用程序会发出各种指标（例如 cpu_user 和 memory_free）和事件（例如 task_completed 和 gc_reclevied）。每个指标或事件都与八个维度（例如区域或小区）相关联，这些维度唯一标识发出该指标或事件的服务器。写入数据时将 20 个指标一起存储在带有度量名称指标的多度量记录中，所有 5

个事件一起存储在另一个带有度量名称事件的多指标记录中。数据模型、架构和数据生成可以在开源数据生成器中找到。除了架构和数据分布外，数据生成器还提供了一个使用多个写入器并行摄取数据的示例，使用 Timestream 的摄取缩放每秒采集数百万个测量值。 LiveAnalytics 下面我们显示架构（表和度量架构）以及数据集中的一些示例数据。

主题

- 多重测量记录
- 单项测量记录

多重测量记录

表架构

以下是使用多度量记录摄取数据后的表架构。它是DESCRIBE查询的输出。假设数据被提取到数据库 raw_data 和表 devops 中，则查询如下。

```
DESCRIBE "raw_data"."devops"
```

列	类型	LiveAnalytics 属性类型的时间流
availability_zone	varchar	DIMENSION
微服务名	varchar	DIMENSION
实例名	varchar	DIMENSION
进程_名称	varchar	DIMENSION
os_version	varchar	DIMENSION
jdk_version	varchar	DIMENSION
细胞	varchar	DIMENSION
region	varchar	DIMENSION
筒仓	varchar	DIMENSION

列	类型	LiveAnalytics 属性类型的时间流
instance_type	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
时间	时间戳	TIMESTAMP
内存_免费	double	MULTI
cpu_steal	double	MULTI
cpu_iowait	double	MULTI
cpu_user	double	MULTI
内存_cached	double	MULTI
disk_io_reads	double	MULTI
cpu_Hi	double	MULTI
每次读取延迟	double	MULTI
网络字节输出	double	MULTI
cpu_idle	double	MULTI
disk_free	double	MULTI
内存_已用	double	MULTI
cpu_system	double	MULTI
文件描述符_in_use	double	MULTI
disk_used	double	MULTI
cpu_nice	double	MULTI
disk_io_writes	double	MULTI

列	类型	LiveAnalytics 属性类型的时间流
cpu_si	double	MULTI
每次写入延迟	double	MULTI
网络字节_in	double	MULTI
任务结束状态	varchar	MULTI
gc_pause	double	MULTI
任务_已完成	bigint	MULTI
gc_recleved	double	MULTI

度量架构

以下是SHOWMEASURES查询返回的度量架构。

```
SHOW MEASURES FROM "raw_data"."devops"
```

measure_name	data_type	尺寸
events	多	[{“data_type”: “varchar”、“dimension_name”: “availability_zone”}、{“data_type”: “varchar”、“dimension_name”: “instance_type”: “varchar”、“dimension_name”: “instance_name”}、{“data_type”: “varchar”、“varchar”、“dimension_name”: “instance_name”}、{“data_tyd imension_name”: “process_name”}、{“data_type”: “varchar”、“dimension_name”:

measure_name	data_type	尺寸
		“jdimension_name”: “jdimen_name”: “jdimen_type”: “jdimension_name”: “region”}, {“data_type”: “region”}, {“data_type”: “dimension_type”: “region”} , {“data_type”: “dimension_“ varchar” , “dimension_name”: “silo”}]
指标	多	[{“data_type”: “varchar” 、“dimension_name”: “availability_zone”}、{“data_type”: “varchar”、“dimension_name”: “instance_type”: “varchar”、“dimension_name”: “instance_name”}、{“data_type”: “varchar”、“varchar”、“dimension_name”: “instance_name”}、{“data_tyd imension_name”: “os_version”} , {“data_type”: “varchar” , “dimension_name”: “cell”} , {“data_type”: “dimension_name”: “silo”} , {“data_type”: “varchar” , “dimension_name”: “silo”} , {“data_type”: “varchar” , “dimension_name”: “silo”} , {“data_type”: “varchchar” , “dimension_name”: “instance_type”}]

示例数据

re	单元格	管仓	ait	微服务名	实例名	int	on	进程_名称	jo	ma	时间	c	c	c	c	c	c	c	c	c	内存_已用	内存_	de	每次请取延迟	dri	每次写入延迟	d	d	网络字节_i	网络字节输出	文件描述符_us	内存_免费	归多结束状态	g	归多_已完成	gc_recle ed
us	us-c	us-cs ilo	us-1	at	i-c -1 si za au -e 26 w	me	AP			排板	11 1:	6	0	3	0	0	0	0	0	5	8	5	9	3	2	6	2	8	4	3	5				
us	us-c	us-cs ilo	us-1	at	i-c -1 si za au -e 26 w	me	AP			排板	11 1:	5	0	3	0	0	0	0	0	9	3	5	3	2	9	5	3	7	5	6	2				
us	us-	us-	us-1	at	i-c -1	me	AP			排板	11 1:	4	0	4	0	0	0	0	0	9	4	5	3	8	3	7	6	8	5	4	8				

re	单元格	值仓	ati	微服务名	实例名	int	on	进程_名称	jo	ma	时间	c	cm	c	c	ct	c	c	c	内存_已用	内存_	de	每次读取延迟	dri	每次写入延迟	d	d	网络字节_i	网络字节输出	文件描述符_us	内存_免费	归多结束状态	g	归多_已完成	gse_ecled
	-cc	cs ilc			si za at us - e 2- 6 w																														
us e	us e- c	us e- cs ilc	us e -1	at	i- c -1 si za at us - e 2- 6 w	m e	A			排标	1 1 1	3	0	5	0	0	0	0	0	4	7	2	4	4	3	8	6	2	1	2	1				

re	单元格	管仓	ait	微服务名	实例名	int	on	进程_名称	jo	ma	时间	c	c	c	c	c	c	c	c	c	内存_已用	内存_	de	每次请取延迟	dri	每次写入延迟	d	d	网络字节_i	网络字节输出	文件描述符_us	内存_免费	归多结束状态	g	归多_已完成	se_recle
us	us-c	us-cs	us-1	at	i-c-1-si-za-au-e-26w	me	A			排标	11 11:	5	0	3	0	0	0	0	0	0	5	3	8	5	7	1	8	6	7	6	5	3				
us	us-c	us-cs	us-1	at	i-c-1-si-za-au-e-26w			主机管理器	Jl	ev	11 11:																					7	SW_不_R	8	3	64	3.8

re	单元格	管理	ar ity	微服务名	实例名	in ty	on	进程_名称	jo	ma	时间	cl	cl	cl	cl	cl	cl	cl	cl	cl	内存_已用	内存_	de	每次调取延迟	di ri	每次写入延迟	d	d	网络字节_i	网络字节输出	文件描述符_in us	内存_免费	任务状态	g	任务_已完成	gc_recle ed
us e	us e- c	us e- c s il	us e -1	ar	i- c -1 si za at u- e 2- 6 w			主材管理器	JI	e	1 1 1																					7	SI T	2	4	73	45 JL	W
us e	us e- c	us e- c s il	us e -1	ar	i- c -1 si za at u- e 2- 6 w			主材管理器	JI	e	1 1 1																					6	SI T	6	2	72	3 JL	W

re	单元格	管仓	ait	微服务名	实例名	int	on	进程_名称	jo	ma	时间	c	c	c	c	c	c	c	c	c	内存_已用	内存_	de	每次调取延迟	dri	每次写入延迟	d	d	网络字节_i	网络字节输出	文件描述符_us	内存_免费	归多结束状态	g	归多_已完成	gse_recle ed
us	us-co	us-cosilo	us-1	a	i-c-1-siza-us-e-26w			主材管理器	JI	e	111																					2	SIIT	5	1	99	9_WJL
us	us-co	us-cosilo	us-1	a	i-c-1-siza-us-e-26w			主材管理器	JI	e	111																					3	SW_不_R	7	2	82	3.9

单项测量记录

Timestream for LiveAnalytics 还允许您使用每个时间序列记录中的一个度量来摄取数据。以下是使用单一测量记录摄取时的架构详细信息。

表架构

以下是使用多度量记录摄取数据后的表架构。它是DESCRIBE查询的输出。假设数据被提取到数据库 raw_data 和表 devops 中，则查询如下。

```
DESCRIBE "raw_data"."devops_single"
```

列	类型	LiveAnalytics 属性类型的时间流
availability_zone	varchar	DIMENSION
微服务名	varchar	DIMENSION
实例名	varchar	DIMENSION
进程_名称	varchar	DIMENSION
os_version	varchar	DIMENSION
jdk_version	varchar	DIMENSION
细胞	varchar	DIMENSION
region	varchar	DIMENSION
筒仓	varchar	DIMENSION
instance_type	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
时间	时间戳	TIMESTAMP
measure_value::double	double	MEASURE_VALUE

列	类型	LiveAnalytics 属性类型的时间流
measure_value::bigint	bigint	MEASURE_VALUE
measure_value::varchar	varchar	MEASURE_VALUE

度量架构

以下是SHOWMEASURES查询返回的度量架构。

```
SHOW MEASURES FROM "raw_data"."devops_single"
```

measure_name	data_type	尺寸
cpu_Hi	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]
cpu_idle	double	[{'dimension_name ' : ' availability_zone ' , ' data_type

measure_name	data_type	尺寸
		' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
cpu_iowait	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
cpu_nice	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
cpu_si	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
cpu_steal	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
cpu_system	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
cpu_user	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
disk_free	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
disk_io_reads	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
disk_io_writes	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
disk_used	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
文件描述符_in_use	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
gc_pause	double	[{'dimension_name ':' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' process_name '} , {' dimension_name ' : ' process_name '} , {' dimension _name ' : ' process_name ' , ' data_type ':' varchar '} , {' dimension_name ' : ' jdk_version ' , ' data_type ' : ' varchar '} , {' dimension_name ' : ' region ' , ' data_type ' : ' varchar '} , {' dimension_name '} , {' dimension_name ' : ' region ' , ' data_type ' : ' varchar '} , {' dimension_name ':' silo '、' data_type ':' varchar '}]

measure_name	data_type	尺寸
gc_recleved	double	[{'dimension_name ':' availabil ity_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' process_name '} , {' dimension_name ' : ' process_name '} , {' dimension _name ' : ' process_name ' , ' data_type ':' varchar '} , {' dimension_name ' : ' jdk_versi on ' , ' data_type ' : ' varchar '} , {' dimension_name ' : ' region ' , ' data_type ' : ' varchar '} , {' dimension_name '} , {' dimension_name ' : ' region ' , ' data_type ' : ' varchar '} , {' dimension_name ':' silo '、 ' data_type ':' varchar '}]

measure_name	data_type	尺寸
每次读取延迟	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
每次写入延迟	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
内存_cached	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
内存_免费	double	[{'dimension_name ':' availabil ity_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' process_name '} , {' dimension_name ' : ' process_name '} , {' dimension _name ' : ' process_name ' , ' data_type ':' varchar '} , {' dimension_name ' : ' os_version ' , ' data_type ' : ' varchar '} , {' dimension_name ' : ' cell ' , ' data_type ' : ' varchar '} , {' dimension_name ' : ' cell ' , ' data_type ' : ' varchar '} , {' dimension_name ' : ' cell ' , ' data_type ' : ' varchar '} , {' dimensname ':' region '、' data_type ':' varchar '}、{' dimension_name ':' silo '、' data_type ':' varchar '}、{' dimension_name ':' instance_ type '、'data_type': 'varchar'}]

measure_name	data_type	尺寸
内存_已用	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
网络字节_in	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
网络字节输出	double	[{'dimension_name ' : ' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' os_version '} , {' dimension_name ' : ' os_version ' , ' data_type ':' varchar '}、{' dimension_name ' : ' cell '、' data_type ' : ' varchar '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name ' : ' varchar '}、{' dimension _name '}、{' dimension_name ' : ' silo '、' data_type ' : ' varchar '}、{' dimension_name instance_type ' , ' data_type ':' varchar '}]

measure_name	data_type	尺寸
任务_已完成	bigint	[{'dimension_name ':' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' process_name '} , {' dimension_name ' : ' process_name '} , {' dimension _name ' : ' process_name ' , ' data_type ':' varchar '} , {' dimension_name ' : ' jdk_version ' , ' data_type ' : ' varchar '} , {' dimension_name ' : ' region ' , ' data_type ' : ' varchar '} , {' dimension_name '} , {' dimension_name ' : ' region ' , ' data_type ' : ' varchar '} , {' dimension_name ':' silo '、' data_type ':' varchar '}]

measure_name	data_type	尺寸
任务结束状态	varchar	[{'dimension_name ':' availability_zone ' , ' data_type ' : ' varchar '} , {' dimension _name ' : ' microservice_name ' : ' varchar '} , {' dimension _name ' : ' process_name '} , {' dimension_name ' : ' process_name '} , {' dimension _name ' : ' process_name ' , ' data_type ':' varchar '} , {' dimension_name ' : ' jdk_version ' , ' data_type ' : ' varchar '} , {' dimension_name ' : ' region ' , ' data_type ' : ' varchar '} , {' dimension_name '} , {' dimension_name ' : ' region ' , ' data_type ' : ' varchar '} , {' dimension_name ':' silo '、' data_type ':' varchar '}]

示例数据

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west- -1	大力神	i-zaZs\ k--1- cell -9- silo-2		AL20		eu-west- - cell-9	eu-west-	eu-west- - cell-9 s ilo-2	r5.4xl e	cpu_l	34:57	0.871		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
		-0000 .am hercu eu-west azona om												
eu-west- -1	大力神	i-zaZsv k--1-cell -9-silo-2 -0000 .am hercu eu-west azona om		AL20		eu-west- -cell-9	eu-west-	eu-west- -cell-9 silo-2	r5.4xl e	cpu_i	34:57	3.462		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_i t	34:57	0.102		
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_r	34:57	0.630		

availa ity_zo	微 服 务 名	实 例 名	进 程 _名 称	os_ve n	jdk_v on	单 元 格	regio	筒 仓	instar type	meas ame	时 间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大 力 神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s	34:57	0.164		
eu- west- -1	大 力 神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s	34:57	0.107		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s m	34:57	0.457		
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_u	34:57	94.20		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	disk_ 	34:57	72.51		
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	disk_i eads	34:57	81.73		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	disk_i rites	34:57	77.11		
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	disk_	34:57	89.42		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	大力神	i-zaZsv k--1-cell -9-silo-2 -000(.am hercu eu-west azona om		AL20		eu-west- -cell-9	eu-west-	eu-west- -cell-9 silo-2	r5.4xl e	文件描述符_in_use	34:57	30.08		
eu-west--1	大力神	i-zaZsv k--1-cell -9-silo-2 -000(.am hercu eu-west azona om	服务器		JDK_	eu-west- -cell-9	eu-west-	eu-west- -cell-9 silo-2		gc_pa	34:57	60.28		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om	服务器		JDK_	eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2		gc_re ed	34:57	75.28		
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	每次读取延迟	34:57	8.076		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	大力神	i-zaZsv k--1-cell -9-silo-2 -0000 .am hercu eu-west azona om		AL20		eu-west- -cell-9	eu-west-	eu-west- -cell-9 silo-2	r5.4xl e	每次写入延迟	34:57	58.11		
eu-west--1	大力神	i-zaZsv k--1-cell -9-silo-2 -0000 .am hercu eu-west azona om		AL20		eu-west- -cell-9	eu-west-	eu-west- -cell-9 silo-2	r5.4xl e	内存_cach	34:57	87.56		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om	服务器		JDK_	eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2		内存_免费	34:57	18.95		
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	内存_免费	34:57	97.20		

availa ity_zo	微 服 务 名	实 例 名	进 程 _名 称	os_ve n	jdk_v on	单 元 格	regio	筒 仓	instar type	meas ame	时 间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大 力 神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	内 存 _已 用	34:57	12.37		
eu- west- -1	大 力 神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	netwo ytes_	34:57	31.02		

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	网络字节输出	34:57	0.514		
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -0000 .am hercu eu- west azona om	服务器		JDK_	eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2		任务_已完成	34:57		69	

availa ity_zo	微服务名	实例名	进程_名称	os_ve n	jdk_v on	单元格	regio	筒仓	instar type	meas ame	时间	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	大力神	i- zaZsv k--1- cell -9- silo-2 -000(.am hercu eu- west azona om	服务器		JDK_	eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2		任务结束状态	34:57			SUCCESS_ ITH_RESUL T

计划查询模式

在本节中，您将找到一些常见的模式，说明如何使用用于 LiveAnalytics计划查询的 Amazon Timestream 来优化控制面板，从而更快地加载并降低成本。以下示例使用 DevOps 应用场景来说明适用于一般定时查询的关键概念，无论应用场景如何。

Timestream 中的计划查询 LiveAnalytics 允许您使用 Timestream 的整个SQL表面区域来表达您的查询。 LiveAnalytics您的查询可以包含一个或多个源表，执行聚合或 Timestream SQL 语言允许的任何其他查询，然后在 Timestream 的另一个目标表中实现查询结果。 LiveAnalytics LiveAnalytics为便于阐述，本节将定时查询的目标表称为派生表。

以下是本节中介绍的要点。

- 使用简单的队列级聚合来解释如何定义计划查询并理解一些基本概念。
- 如何将定时查询目标（派生表）的结果与源表的结果结合起来，以获得定时查询的成本和性能优势。
- 在配置计划查询的刷新周期时，你有什么权衡取舍。
- 在一些常见场景中使用计划查询。
  - 跟踪特定日期之前每个实例的最后一个数据点。

  - 用于在仪表板中填充变量的维度的不同值。
- 如何在计划查询的上下文中处理延迟到达的数据。
- 如何使用一次性手动执行来处理计划查询的自动触发器未直接涵盖的各种场景。

主题

场景

以下示例使用了中概述的 DevOps 监控场景计划查询示例架构。

这些示例提供了计划查询的定义，您可以在其中插入适当的配置，以了解在何处接收计划查询的执行状态通知，在何处接收执行计划查询期间遇到的错误的报告，以及计划查询用于执行其操作的IAM角色。

在填写上述选项、创建目标（或派生）表并执行完之后，您可以创建这些定时查询AWSCLI。例如，假设计划查询定义存储在文件中`scheduled_query_example.json`。您可以使用CLI命令创建查询。

```
aws timestream-query create-scheduled-query --cli-input-json  file://
scheduled_query_example.json --profile aws_profile --region us-east-1
```

在前面的命令中，使用--profile 选项传递的配置文件必须具有创建计划查询的相应权限。有关策略和权限的详细说明，请参阅针对计划查询的基于身份的策略。

简单的队列级聚合

第一个示例使用计算队列级聚合的简单示例，向您介绍处理定时查询时的一些基本概念。使用此示例，您将学到以下内容。

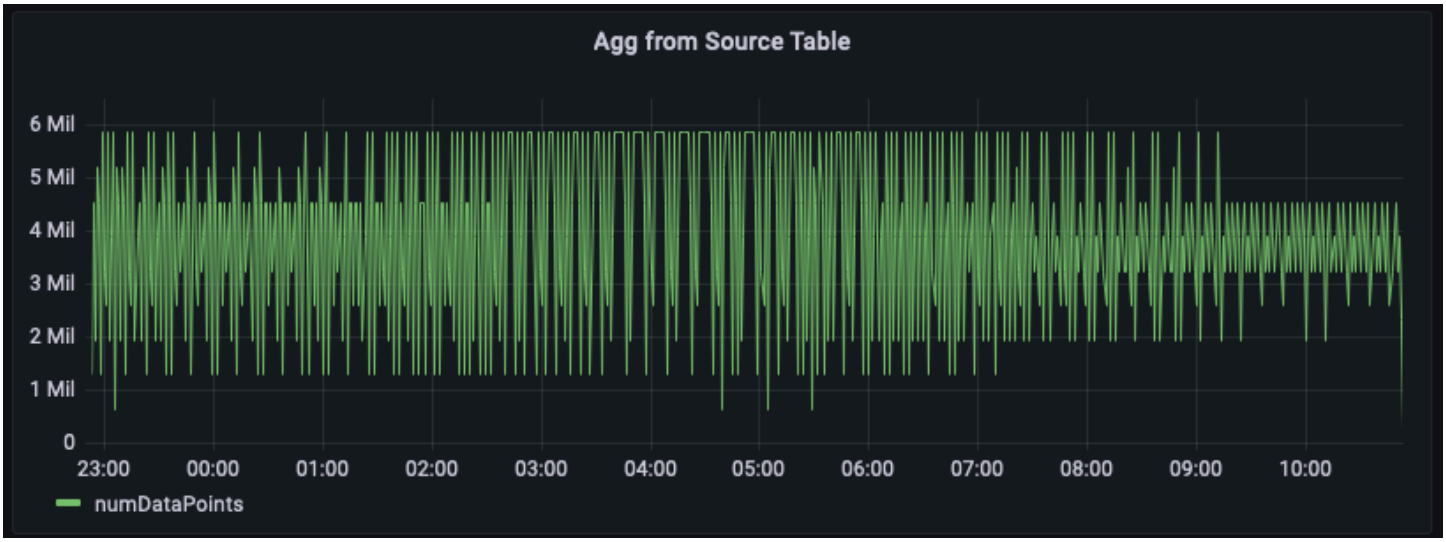
- 如何获取用于获取汇总统计信息的仪表板查询并将其映射到计划查询。
- Timestream 如何 LiveAnalytics 管理您的计划查询的不同实例的执行。
- 如何让不同的计划查询实例在时间范围内重叠，以及如何在目标表上保持数据的正确性，以确保使用计划查询结果的仪表板为您提供与基于原始数据计算的相同聚合相匹配的结果。

- 如何为计划查询设置时间范围和刷新节奏。
- 如何自助跟踪计划查询的结果以对其进行调整，从而使查询实例的执行延迟在刷新仪表板后可接受的延迟范围内。

主题

从源表中聚合

在此示例中，您正在跟踪给定区域内的服务器每分钟发出的指标数量。下图是绘制 us-east-1 区域此时间序列的示例。



以下是根据原始数据计算此聚合的示例查询。它筛选区域 us-east-1 的行，然后通过考虑 20 个指标（如果 measure_name 是指标）或 5 个事件（如果 measure_name 是事件）来计算每分钟总和。在此示例中，图表插图显示发出的指标数量在每分钟 150 万到 600 万之间。在绘制该时间序列数小时（本图中为过去 12 小时）时，此对原始数据的查询将分析数亿行。

```
WITH grouped_data AS (
    SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name = 'metrics' THEN
 20 ELSE 5 END) as numDataPoints
    FROM "raw_data"."devops"
```

```
    WHERE time BETWEEN from_milliseconds(1636699996445) AND
 from_milliseconds(1636743196445)
        AND region = 'us-east-1'
    GROUP BY region, measure_name, bin(time, 1m)
)
SELECT minute, SUM(numDataPoints) AS numDataPoints
FROM grouped_data
GROUP BY minute
ORDER BY 1 desc, 2 desc
```

预先计算聚合的计划查询

如果您想通过扫描更少的数据来优化仪表板以加快加载速度并降低成本，则可以使用计划查询来预先计算这些聚合。Timestream 中的计划查询 LiveAnalytics 允许您在另一个 Timestream 表中实现这些预计算，随后您可以将其用于仪 LiveAnalytics 表板。

创建计划查询的第一步是确定要预先计算的查询。请注意，前面的控制面板是为区域 us-east-1 绘制的。但是，不同的用户可能希望为不同的区域使用相同的聚合，比如 us-west-2 或 eu-west-1。为避免为每个此类查询创建定时查询，您可以预先计算每个区域的聚合，并在表的另一个 Timestream 中实现每个区域的聚合。 LiveAnalytics

下面的查询提供了相应的预计算示例。如你所见，它与查询原始数据时使用的公用表表达式 grouped_data 类似，但有两个区别：1) 它不使用区域谓词，因此我们可以使用一个查询来预先计算所有区域；2) 它使用带有特殊参数 @scheduled_runtime 的参数化时间谓词，详情见下文。

```
SELECT region, bin(time, 1m) as minute,
    SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m
GROUP BY bin(time, 1m), region
```

使用以下规范，可以将前面的查询转换为定时查询。为计划查询分配一个 Name，这是一个用户友好的助记词。然后，它包括 QueryString、a ScheduleConfiguration，这是一个 cron 表达式。它指定 TargetConfiguration 了将查询结果映射到 Timestream 中的目标表的。 LiveAnalytics最后，它指定了许多其他配置，例如 NotificationConfiguration，在查询的各个执行中发送通知，ErrorReportConfiguration 在查询遇到任何错误时写入报告 ScheduledQueryExecutionRoleArn，以及用于对计划查询执行操作的角色。

```
{
    "Name": "MultiPT5mPerMinutePerRegionMeasureCount",
```

```
    "QueryString": "SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name
 = 'metrics' THEN 20 ELSE 5 END) as numDataPoints FROM raw_data.devops WHERE time
 BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m GROUP BY bin(time, 1m),
 region",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/5 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_minute_aggs_pt5m",
            "TimeColumn": "minute",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

在示例中， ScheduleExpression cron (0/5 * * *? *) 表示查询每 5 分钟在每天每小时的 5、10、15、... 分钟执行一次。触发此查询的特定实例时的这些时间戳即为查询中使用的 @scheduled_runtime 参数。例如，考虑这个在 2021-12-01 00:00:00 执行的定时查询的实例。在本例中，在调用查询时，@scheduled_runtime 参数被初始化为时间戳 2021-12-01 00:00:00。因此，此特定实例将在时间戳 2021-12-01 00:00:00 执行，并将计算从 2021-11-30 23:50:00 到 2021-12-01 00:01:00 的时间范围内的每分钟总计。同样，此查询的下一个实例将在时间戳 2021-12-01 00:05:00 触发，在这种情况下，该查询将计算从 2021-11-30 23:55:00 到 2021-12-01 00:06:00 之间的每分钟聚合。因此，@scheduled_runtime 参数提供了一个计划查询，用于使用查询的调用时间预先计算配置的时间范围内的聚合。

请注意，两个后续查询实例的时间范围重叠。这是您可以根据自己的要求进行控制的。在这种情况下，这种重叠允许这些查询根据到达稍有延迟（在本例中最多 5 分钟）的任何数据来更新聚合。为了确保物化查询的正确性，Timestream for LiveAnalytics 确保只有在 2021-12-01 00:00:00 的查询完成之后，才会执行 2021-12-01 00:05:00 的查询，并且如果生成了较新的值，则后一个查询的结果可以使用更新任何先前已实现的聚合。例如，如果时间戳为2021-11-30 23:59:00 的某些数据是在执行2021-12-01 00:00:00 的查询之后，但在 2021-12-01 00:05:00 的查询之前到达，则在 2021-12-01 00:05:00 执行时将重新计算2021-11-30 23:59:00 的聚合，这将导致使用新计算的值更新之前的聚合。你可以依靠计划查询的这些语义来权衡更新预计算的速度和优雅地处理延迟到达的某些数据的方式。下文将讨论其他注意事项，说明如何权衡刷新节奏与数据的新鲜度，以及如何解决更新聚合以获取延迟更长的数据的问题，或者您的计划计算源是否更新了需要重新计算聚合的值。

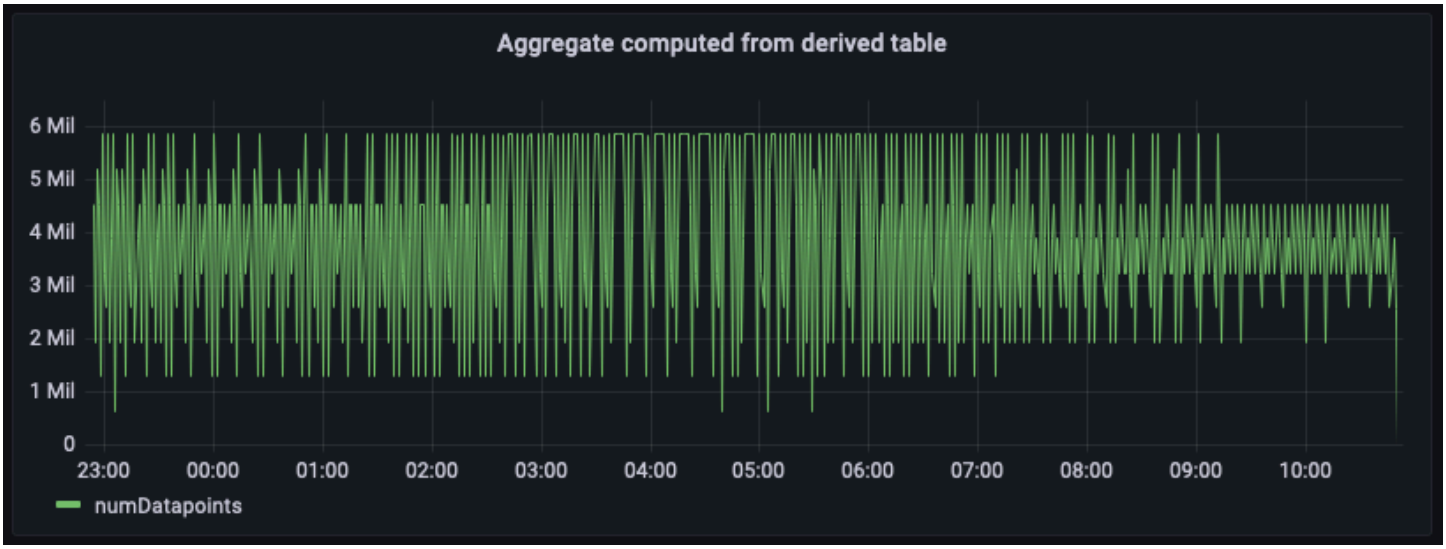
每个计划计算都有一个通知配置，其中 Timestream for LiveAnalytics 会发送每次执行计划配置的通知。您可以为配置一个SNS主题以接收每次调用的通知。除了特定实例的成功或失败状态外，它还有一些统计信息，例如执行此计算所花费的时间、计算扫描的字节数以及计算写入其目标表的字节数。您可以使用这些统计数据来进一步调整查询、安排配置或跟踪计划查询的支出。值得注意的一个方面是实例的执行时间。在此示例中，计划计算配置为每 5 分钟执行一次。执行时间将决定预计算可用的延迟，这也将定义您在仪表板中使用预先计算的数据时仪表板中的延迟。此外，如果此延迟一直高于刷新间隔，例如，如果配置为每 5 分钟刷新一次的计算的执行时间超过 5 分钟，则必须调整计算以更快地运行，以避免仪表板出现进一步的延迟。

从派生表中聚合

现在，您已经设置了计划查询，并且已预先计算聚合并具体化到计划计算的目标配置中指定的 LiveAnalytics 表的另一个时间流，您可以使用该表中的数据编写SQL查询来增强仪表板的功能。以下是使用物化预聚合生成 us-east-1 的每分钟数据点数聚合的查询等效项。

```
SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
FROM "derived"."per_minute_aggs_pt5m"
```

```
WHERE time BETWEEN from_milliseconds(1636699996445) AND
 from_milliseconds(1636743196445)
    AND region = 'us-east-1'
GROUP BY bin(time, 1m)
ORDER BY 1 desc
```



上图绘制了根据聚合表计算出的聚合。将此面板与根据原始源数据计算出的面板进行比较，您会发现它们完全匹配，尽管这些聚合会延迟几分钟，这取决于您为计划计算配置的刷新间隔加上执行时间。

与根据原始源数据计算的聚合相比，这种对预先计算的数据的查询扫描的数据要少几个数量级。根据聚合的粒度，这种减少可以很容易地将成本和查询延迟降低 100 倍。执行此计划计算是有成本的。但是，根据这些仪表板的刷新频率以及加载这些仪表板的并发用户数量，使用这些预计算最终可以显著降低总体成本。除此之外，仪表板的加载时间还缩短了 10-100 倍。

合并源表和派生表的聚合

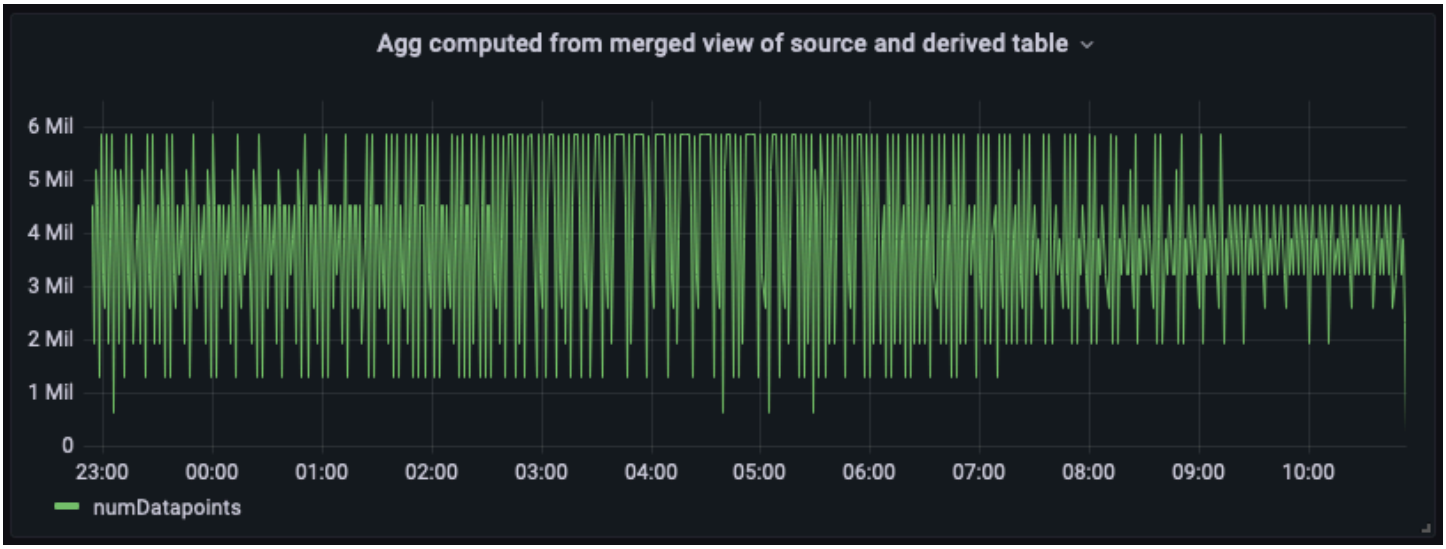
使用派生表创建的仪表板可能会有延迟。如果您的应用场景要求仪表板包含最新数据，则可以利用 Timestream for SQL 支持的强大功能和灵活性，将源表中的最新数据与派生表中的历史聚合数据结合起来，形成合并视图。 LiveAnalytics此合并视图使用源表和派生表的SQL并集语义和非重叠时间范围。在下面的示例中，我们使用的是 “派生”。” per_minute_aggs_pt5m” 派生表。由于该派生表的计划计算每 5 分钟刷新一次（根据计划表达式规范），因此下面的查询使用源表中最近 15 分钟的数据，以及派生表中超过 15 分钟的任何数据，然后合并结果以创建两全其美的合并视图：通过从派生表中读取较旧的预先计算的聚合来实现经济性和低延迟，以及从源表中读取聚合的新鲜度表格可为您的实时分析用例提供支持。

请注意，与仅查询派生表相比，这种联合方法的查询延迟会稍高一些，而且扫描的数据也略高，因为它实时聚合原始数据以填充最近的时间间隔。但是，与从源表中即时聚合相比，这种合并视图仍然会更

快、更便宜，特别是对于呈现数天或数周数据的仪表板而言。您可以调整此示例的时间范围，以适应应用程序的刷新需求和延迟容限。

```
WITH aggregated_source_data AS (
    SELECT bin(time, 1m) as minute, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE
 5 END) as numDatapoints
    FROM "raw_data"."devops"
    WHERE time BETWEEN bin(from_milliseconds(1636743196439), 1m) - 15m AND
 from_milliseconds(1636743196439)
        AND region = 'us-east-1'
    GROUP BY bin(time, 1m)
), aggregated_derived_data AS (
    SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
    FROM "derived"."per_minute_aggs_pt5m"
    WHERE time BETWEEN from_milliseconds(1636699996439) AND
 bin(from_milliseconds(1636743196439), 1m) - 15m
        AND region = 'us-east-1'
    GROUP BY bin(time, 1m)
)
SELECT minute, numDatapoints
FROM (
    (
    SELECT *
    FROM aggregated_derived_data
    )
    UNION
    (
    SELECT *
    FROM aggregated_source_data
    )
)
ORDER BY 1 desc
```

下面是带有此统一合并视图的仪表板面板。如您所见，仪表板看起来与根据派生表计算出的视图几乎相同，唯一的不同是它在最右边的尖端会有最多的 up-to-date聚合。



根据经常刷新的计划计算进行聚合

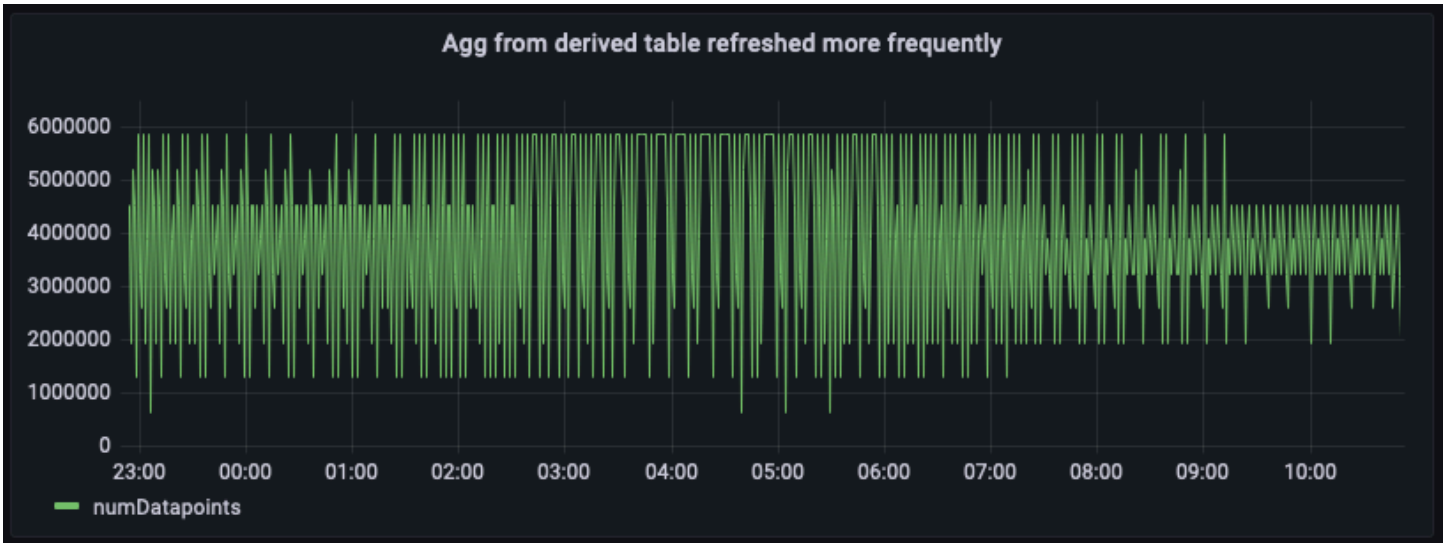
根据仪表板的加载频率和仪表板的延迟时间，还有另一种方法可以在仪表板中获得更新鲜的结果：让计划的计算更频繁地刷新聚合。例如，以下是相同计划计算的配置，只是它每分钟刷新一次（注意调度 express cron (0/1 * * *? *))。使用此设置，与计算指定每 5 分钟刷新一次的场景相比，派生表 per_minute_aggs_pt1m 将具有更新的聚合。

```
{
    "Name": "MultiPT1mPerMinutePerRegionMeasureCount",
    "QueryString": "SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name
 = 'metrics' THEN 20 ELSE 5 END) as numDataPoints FROM raw_data.devops WHERE time
 BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m GROUP BY bin(time, 1m),
 region",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/1 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_minute_aggs_pt1m",
            "TimeColumn": "minute",
            "DimensionMappings": [
                {
```

```
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

```
SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
FROM "derived"."per_minute_aggs_pt1m"
WHERE time BETWEEN from_milliseconds(1636699996446) AND
 from_milliseconds(1636743196446)
    AND region = 'us-east-1'
GROUP BY bin(time, 1m), region
ORDER BY 1 desc
```

由于派生表具有较新的聚合，因此您现在可以直接查询 per_minute_aggs_pt1m 的派生表以获得更新的聚合，从之前的查询和下面的仪表板快照中可以看出。



请注意，以更快的计划（比如 1 分钟而不是 5 分钟）刷新计划计算会增加计划计算的维护成本。每次计算执行的通知消息都会提供有关扫描了多少数据以及向派生表中写入了多少数据的统计信息。同样，如果您使用合并视图合并派生表，则在合并视图上查询成本，与仅查询派生表相比，仪表板加载延迟会更高。因此，您选择的方法将取决于仪表板的刷新频率以及计划查询的维护成本。如果您有数十名用户每分钟左右刷新一次仪表板，那么更频繁地刷新派生表可能会降低总体成本。

每台设备的最后一点

您的应用程序可能会要求您读取设备发出的最后一次测量值。在给定设备之前，可能有更通用的用例来获取设备的最后一次测量值date/time or the first measurement for a device after a given date/time。当您拥有数百万台设备和多年的数据时，此搜索可能需要扫描大量数据。

在下面，您将看到一个示例，说明如何使用计划查询来优化对设备发出的最后一个点的搜索。如果您的应用程序需要，也可以使用相同的模式来优化第一个点查询。

主题

- 根据源表计算
- 要按每日粒度预先计算的派生表
- 根据派生表计算
- 合并源表和派生表

根据源表计算

以下是一个查询示例，用于查找特定部署（例如，给定区域内给定微服务的服务器、单元、筒仓和 availability_zone）中发布的最后一个测量值。在示例应用程序中，此查询将返回数百台服务器的最后一次测量值。另请注意，此查询具有无限的时间谓词，它会查找任何早于给定时间戳的数据。

> **Note**
>
> 有关max和max_by函数的信息，请参见聚合函数。

```
SELECT instance_name, MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
FROM "raw_data"."devops"
WHERE time < from_milliseconds(1636685271872)
    AND measure_name = 'events'
    AND region = 'us-east-1'
    AND cell = 'us-east-1-cell-10'
    AND silo = 'us-east-1-cell-10-silo-3'
    AND availability_zone = 'us-east-1-1'
    AND microservice_name = 'hercules'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
ORDER BY instance_name, time DESC
```

要按每日粒度预先计算的派生表

您可以将前面的用例转换为计划计算。如果您的应用程序要求要求您可能需要跨多个区域、单元、孤岛、可用区和微服务获取整个队列的这些值，则可以使用一次计划计算来预先计算整个队列的值。这就是 Timestream LiveAnalytics 的无服务器定时查询的强大功能，它允许这些查询根据应用程序的扩展要求进行扩展。

以下是一个查询，用于预先计算给定日期所有服务器的最后一个点。请注意，该查询只有时间谓词，没有维度的谓词。时间谓词将查询限制在根据指定的计划表达式触发计算之日起的最后一天。

```
SELECT region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version,
    MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
FROM raw_data.devops
WHERE time BETWEEN bin(@scheduled_runtime, 1d) - 1d AND bin(@scheduled_runtime, 1d)
    AND measure_name = 'events'
GROUP BY region, cell, silo, availability_zone, microservice_name,
```

```
    instance_name, process_name, jdk_version
```

以下是使用前面的查询进行计划计算的配置，该查询在UTC每天 01:00 执行该查询以计算过去一天的聚合。调度表达式 cron (0 1 * *? *) 控制此行为，并在一天结束一小时后运行，以考虑迟到一天的任何数据。

```
{
    "Name": "PT1DPerInstanceLastpoint",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 instance_name, process_name, jdk_version, MAX(time) AS time, MAX_BY(gc_pause, time)
 AS last_measure FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1d) -
 1d AND bin(@scheduled_runtime, 1d) AND measure_name = 'events' GROUP BY region, cell,
 silo, availability_zone, microservice_name, instance_name, process_name, jdk_version",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0 1 * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_timeseries_lastpoint_pt1d",
            "TimeColumn": "time",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
```

```
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "last_measure",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "last_measure",
                        "MeasureValueType": "DOUBLE"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

根据派生表计算

使用上述配置定义派生表，并且至少有一个定时查询实例已将数据实体化到派生表中后，您现在可以查询派生表以获取最新的测量结果。以下是对派生表的查询示例。

```
SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
FROM "derived"."per_timeseries_lastpoint_pt1d"
WHERE time < from_milliseconds(1636746715649)
    AND measure_name = 'last_measure'
    AND region = 'us-east-1'
    AND cell = 'us-east-1-cell-10'
    AND silo = 'us-east-1-cell-10-silo-3'
    AND availability_zone = 'us-east-1-1'
    AND microservice_name = 'hercules'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
ORDER BY instance_name, time DESC
```

合并源表和派生表

与前面的示例类似，派生表中的任何数据都不会有最新的写入数据。因此，您可以再次使用与之前类似的模式来合并来自派生表的数据以获取较旧的数据，并将源数据用于剩余的小费。以下是使用类似UNION方法进行此类查询的示例。由于应用程序要求是在某个时间段之前找到最新的测量值，而这个开始时间可能已经过去，因此编写此查询的方法是使用提供的时间，使用指定时间起最多一天的源数据，然后对较旧的数据使用派生表。从下面的查询示例中可以看出，源数据上的时间谓词是有界限的。这样可以确保对数据量明显更高的源表进行高效处理，然后在派生表上使用无限的时间谓词。

```
WITH last_point_derived AS (
    SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
    FROM "derived"."per_timeseries_lastpoint_pt1d"
    WHERE time < from_milliseconds(1636746715649)
        AND measure_name = 'last_measure'
        AND region = 'us-east-1'
        AND cell = 'us-east-1-cell-10'
        AND silo = 'us-east-1-cell-10-silo-3'
        AND availability_zone = 'us-east-1-1'
        AND microservice_name = 'hercules'
    GROUP BY region, cell, silo, availability_zone, microservice_name,
        instance_name, process_name, jdk_version
), last_point_source AS (
    SELECT instance_name, MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
    FROM "raw_data"."devops"
    WHERE time < from_milliseconds(1636746715649) AND time >
 from_milliseconds(1636746715649) - 26h
        AND measure_name = 'events'
        AND region = 'us-east-1'
        AND cell = 'us-east-1-cell-10'
```

```
        AND silo = 'us-east-1-cell-10-silo-3'
        AND availability_zone = 'us-east-1-1'
        AND microservice_name = 'hercules'
    GROUP BY region, cell, silo, availability_zone, microservice_name,
        instance_name, process_name, jdk_version
)
SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
FROM (
    SELECT * FROM last_point_derived
    UNION
    SELECT * FROM last_point_source
)
GROUP BY instance_name
ORDER BY instance_name, time DESC
```

前面只是如何构造派生表的一个例子。如果您有多年的数据，则可以使用更多级别的聚合。例如，您可以在每日聚合之上设置每月汇总，也可以在每日汇总之前设置每小时汇总。因此，你可以合并最近填写最近一小时，每小时填写最后一天，每天填写最后一个月，每月填写较早的时间。您设置的级别数量与刷新计划之间的差异将取决于您对这些查询出现问题的频率以及同时发出这些查询的用户数量的要求。

唯一的维度值

你可能有一个用例，即你有仪表板，你想使用维度的唯一值作为变量，深入研究与特定数据片段对应的指标。下面的快照是一个示例，其中仪表板预先填充了多个维度的唯一值，例如区域、单元、孤岛、微服务和 availability_zone。在这里，我们展示了一个示例，说明如何使用计划查询来显著加快计算这些变量与所跟踪指标的不同值的速度。

主题

- 在原始数据上
- 预先计算唯一的维度值
- 计算派生表中的变量

在原始数据上

您可以使用计算SELECTDISTINCT从数据中看到的不同值。例如，如果要获取 region 的不同值，则可以使用此表单的查询。

```
SELECT DISTINCT region
FROM "raw_data"."devops"
WHERE time > ago(1h)
```

```
ORDER BY 1
```

您可能正在跟踪数百万台设备和数十亿个时间序列。但是，在大多数情况下，这些有趣的变量适用于较低的基数维度，其中有几到几十个值。DISTINCT从原始数据进行计算可能需要扫描大量数据。

预先计算唯一的维度值

您希望这些变量能够快速加载，以便您的仪表板具有交互性。此外，这些变量通常是在每次仪表板加载时计算的，因此您也希望它们具有成本效益。您可以优化使用定时查询查找这些变量并将其具体化到派生表中。

首先，您需要确定需要计算DISTINCT值或列的维度，计算值时将在谓词中使用这些DISTINCT值或列。

在此示例中，您可以看到仪表板正在为维度区域、单元格、筒仓、availability_zone 和微服务填充不同的值。因此，您可以使用下面的查询来预先计算这些唯一值。

```
SELECT region, cell, silo, availability_zone, microservice_name,
    min(@scheduled_runtime) AS time, COUNT(*) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 15m AND @scheduled_runtime
GROUP BY region, cell, silo, availability_zone, microservice_name
```

这里有一些重要的事情需要注意。

- 您可以使用一个定时计算来预先计算许多不同查询的值。例如，您正在使用前面的查询来预先计算五个不同变量的值。因此，你不需要每个变量都有一个变量。您可以使用相同的模式来识别跨多个面板的共享计算，以优化需要维护的计划查询数量。
- 维度的唯一值本质上并不是时间序列数据。所以你可以使用 @scheduled_runtime 将其转换为时间序列。通过将此数据与 @scheduled_runtime 参数相关联，您还可以跟踪在给定时间点出现了哪些唯一值，从而从中创建时间序列数据。
- 在前面的示例中，您将看到正在跟踪的指标值。此示例使用 COUNT (*)。如果您想为仪表板跟踪其他有意义的聚合，则可以计算出其他有意义的聚合。

以下是使用上一个查询进行计划计算的配置。在此示例中，使用计划表达式 cron (0/15 * * *? 配置为每 15 分钟刷新一次 *)。

```
{
    "Name": "PT15mHighCardPerUniqueDimensions",
```

```
   "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
min(@scheduled_runtime) AS time, COUNT(*) as numDataPoints FROM raw_data.devops WHERE
time BETWEEN @scheduled_runtime - 15m AND @scheduled_runtime GROUP BY region, cell,
silo, availability_zone, microservice_name",
   "ScheduleConfiguration": {
       "ScheduleExpression": "cron(0/15 * * * ? *)"
   },
   "NotificationConfiguration": {
       "SnsConfiguration": {
           "TopicArn": "******"
       }
   },
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName": "derived",
           "TableName": "hc_unique_dimensions_pt15m",
           "TimeColumn": "time",
           "DimensionMappings": [
               {
                   "Name": "region",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "cell",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "silo",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "availability_zone",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "microservice_name",
                   "DimensionValueType": "VARCHAR"
               }
           ],
           "MultiMeasureMappings": {
               "TargetMultiMeasureName": "count_multi",
               "MultiMeasureAttributeMappings": [
                   {
                       "SourceColumn": "numDataPoints",
```

```
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

计算派生表中的变量

一旦定时计算预先实现了派生表 hc_unique_dimensions_pt15m 中的唯一值，您就可以使用派生表有效地计算维度的唯一值。以下是示例查询，说明如何计算唯一值，以及如何在这些唯一值查询中使用其他变量作为谓词。

区域

```
SELECT DISTINCT region
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
ORDER BY 1
```

细胞

```
SELECT DISTINCT cell
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
  AND region = '${region}'
ORDER BY 1
```

筒仓

```
SELECT DISTINCT silo
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
```

```
   AND region = '${region}' AND cell = '${cell}'
ORDER BY 1
```

微服务

```
SELECT DISTINCT microservice_name
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}'
ORDER BY 1
```

可用区

```
SELECT DISTINCT availability_zone
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}' AND silo = '${silo}'
ORDER BY 1
```

处理迟到的数据

在某些情况下，您的数据可能会延迟很长时间到达，例如，与摄取的行关联的时间戳相比，将数据提取到 Tim LiveAnalytics estream 的时间会明显延迟。在前面的示例中，您已经了解了如何使用 @scheduled_runtime 参数定义的时间范围来解释一些延迟到达的数据。但是，如果您的用例中数据可能会延迟数小时或数天，则可能需要不同的模式来确保派生表中的预计算得到适当更新，以反映此类迟到的数据。有关延迟到达的数据的一般信息，请参阅写入数据（插入和向上移动）。

在下文中，您将看到两种不同的方法来处理延迟到达的数据。

- 如果您的数据到达延迟是可预测的，则可以使用另一个 “追赶” 计划计算来更新聚合，以适应延迟到达的数据。
- 如果您有不可预测的延迟或偶尔会有延迟到达，则可以使用手动执行来更新派生表。

本讨论涵盖数据延迟到达的情况。但是，同样的原则也适用于数据校正，即您修改了源表中的数据，并且想要更新派生表中的聚合。

主题

- 定时追赶查询
- 手动执行不可预测的延迟到达数据

定时追赶查询

查询及时到达的聚合数据

以下是一个模式，您将看到在数据到达延迟可预测的情况下，如何使用自动方式更新聚合。以下是前面对实时数据进行定时计算的示例。此计划计算每 30 分钟刷新一次派生表，并且已经考虑了延迟长达一个小时的数据。

```
{
    "Name": "MultiPT30mPerHrPerTimeseriesDPCount",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 instance_type, os_version, instance_name, process_name, jdk_version, bin(time,
 1h) as hour, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as
 numDataPoints FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1h)
 - 1h AND @scheduled_runtime + 1h GROUP BY region, cell, silo, availability_zone,
 microservice_name, instance_type, os_version, instance_name, process_name,
 jdk_version, bin(time, 1h)",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "dp_per_timeseries_per_hr",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
```

```
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "os_version",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
```

```
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

Catch-up 查询为延迟到达的数据更新聚合

现在，如果您考虑一下您的数据可能会延迟大约 12 个小时的情况。以下是同一查询的变体。但是，不同之处在于，与触发计划计算时相比，它会根据延迟最多 12 小时的数据计算聚合。例如，您将在下面的示例中看到查询，此查询的目标时间范围在触发查询之前的 2 小时到 14 小时之间。此外，如果你注意到调度表达式 cron (0 0,12 * *? *)，它将在每天 00:00 UTC 和 12 UTC :00 触发计算。因此，当查询在 2021-12-01 00:00:00:00 触发时，查询会在 2021-11-30 10:00:00:00 到 2021-11-30 22:00:00:00 之间更新聚合。定时查询使用类似于 Timestream 写入 LiveAnalytics的 upsert 语义，如果窗口中有延迟到达的数据或者发现了较新的聚合（例如，此聚合中出现了一个新的分组，而该分组在触发原始计划计算时并不存在），则此追赶查询将使用较新的值更新聚合值，然后将新的聚合插入到派生表中。同样，当下一个实例在 2021-12-01 12:00:00:00 触发时，该实例将更新 2021-11-30 22:00:00 到 2021-12-01 10:00:00:00 范围内的聚合。

```
      {
   "Name": "MultiPT12HPerHrPerTimeseriesDPCountCatchUp",
   "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
instance_type, os_version, instance_name, process_name, jdk_version, bin(time, 1h)
as hour, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND
bin(@scheduled_runtime, 1h) - 2h GROUP BY region, cell, silo, availability_zone,
microservice_name, instance_type, os_version, instance_name, process_name,
jdk_version, bin(time, 1h)",
   "ScheduleConfiguration": {
       "ScheduleExpression": "cron(0 0,12 * * ? *)"
   },
   "NotificationConfiguration": {
       "SnsConfiguration": {
           "TopicArn": "******"
       }
   },
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName": "derived",
           "TableName": "dp_per_timeseries_per_hr",
           "TimeColumn": "hour",
```

```
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "os_version",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
```

```
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

前面的示例假设您的延迟到达时间限制为 12 小时，并且对于晚于实时窗口的数据，可以每 12 小时更新一次派生表。您可以调整此模式，每小时更新一次派生表，这样您的派生表就可以更快地反映延迟到达的数据。同样，您可以将时间范围调整为超过 12 小时，例如一天甚至一周或更长时间，以处理可预测的延迟数据。

手动执行不可预测的延迟到达数据

在某些情况下，您可能会有不可预测的延迟到达，或者您对源数据进行了更改并在事后更新了一些值。在所有这些情况下，您都可以手动触发计划查询以更新派生表。以下是如何实现这一目标的示例。

假设你的用例是将计算写入派生表 dp_per_timeseries_per_per_hr。你在 devops 表中的基础数据已在 2021-11-30 23:00:00-2021-12-01 00:00:00:00 的时间范围内更新。有两种不同的计划查询可用于更新此派生表：Multi PT3 0 mPerHr PerTimeseries DPCount 和 Multi PT12HPerHrPerTimeseriesDPCountCatchUp。您在 Timestream 中为ARN其创建的每个计划计算都 LiveAnalytics 有一个唯一值，您可以在创建计算或执行列表操作时获得该计算结果。你可以使用ARN进行计算，使用查询获取的参数 @scheduled_runtime 的值来执行此操作。

假设多 PT3 0 的计算mPerHrPerTimeseriesDPCount有 ARN arn_1，并且您想使用此计算来更新派生表。由于前面的计划计算会在 @scheduled_runtime 值之前 1 小时和之后 1 小时更新聚合，因此您可以使用 2021-12-01 00:00:00 的 @scheduled_runtime 参数值来覆盖更新的时间范围（2021-11-30 23:00:00-2021-12-01 00:00:00）。您可以使用传递此计算ARN的值，并使用 ExecuteScheduledQuery

API以纪元秒 (inUTC) 为单位的时间参数值来实现此目的。以下是使用的示例 AWSCLI，您可以使用 Timestream SDKs 支持的任何模式来遵循相同的模式。 LiveAnalytics

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638316800 --profile profile --region us-east-1
```

在前面的示例中，配置文件是具有相应权限进行此API调用的AWS配置文件，1638316800 对应于 2021-12-01 00:00:00 的时代秒。假设系统在所需时间段触发了此次调用，则此手动触发器的行为几乎类似于自动触发器。

如果你在更长的时间段内进行了更新，比如说 2021-11-30 23:00:00-2021-12-01 11:00:00:00 的基础数据已更新，那么你可以多次触发前面的查询以覆盖整个时间范围。例如，你可以执行六种不同的执行方式，如下所示。

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638316800 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638324000 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638331200 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638338400 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638345600 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638352800 --profile profile --region us-east-1
```

前六个命令对应于 2021-12-01 00:00:00、2021-12-01 02:00:00:00、2021-12-01 04:0:00、2021-12-01 06:00:00、2021-12-01 06:00:00、2021-12-01 08:00:00 和 2021-12-01 10:00 调用的预定计算：

或者，您可以使用在 2021-12-01 13:00:00 PT12HPerHrPerTimeseriesDPCountCatchUp 触发的计算单次执行来更新整个 12 小时时间范围内的聚合。例如，如果 arn_2 是该计算ARN的，则可以从中执行以下命令。CLI

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_2 --invocation-
time 1638363600 --profile profile --region us-east-1
```

值得注意的是，对于手动触发器，您可以为调用时间参数使用无需与自动触发时间戳对齐的时间戳。例如，在前面的示例中，尽管自动计划仅在时间戳2021-12-01 10:00:00、2021-12-01 12:00:00:00 和 2021-12-01 12:00:00:00 和 2021-12-02 00:00:00 触发计算。Timestream for 使您可以 LiveAnalytics 灵活地根据手动操作的需要使用适当的值来触发它。

以下是使用时的一些重要注意事项 ExecuteScheduledQuery API。

- 如果您要触发多个这样的调用，则需要确保这些调用不会在重叠的时间范围内生成结果。例如，在前面的示例中，有六次调用。每次调用都覆盖 2 小时的时间范围，因此，为了避免更新中出现任何重叠，每个调用时间戳都分散了两个小时。这样可以确保派生表中的数据最终处于与源表中的聚合相匹配的状态。如果您无法确保时间范围不重叠，请确保这些执行依次触发。如果您同时触发多个执行的时间范围重叠，则可以看到触发失败，您可能会在这些执行的错误报告中看到版本冲突。将根据调用的触发时间为定时查询调用生成的结果分配一个版本。因此，较新的调用生成的行具有更高的版本。较高版本的记录可以覆盖较低版本的记录。对于自动触发的计划查询，Timestream fo LiveAnalytics r 会自动管理计划，这样即使后续调用的时间范围重叠，您也不会看到这些问题。
- 前面已经提到，你可以使用 @scheduled_runtime 的任何时间戳值来触发调用。因此，您有责任适当地设置值，以便在派生表中更新相应的时间范围，与源表中数据的更新范围相对应。
- 您也可以将这些手动触发器用于DISABLED处于状态的计划查询。这允许您定义不按自动计划执行的特殊查询，因为它们处于DISABLED状态。相反，您可以使用其上的手动触发器来管理数据更正或延迟到达的用例。

回填历史预计算

创建计划计算时，Timestream for 会 LiveAnalytics 管理向前执行的查询，其中刷新由您提供的计划表达式控制。根据源表的历史数据量，您可能需要使用与历史数据对应的聚合来更新派生表。您可以使用上述手动触发器逻辑来回填历史聚合。

例如，如果我们考虑派生表 per_timeseries_lastpoint_pt1d，那么过去一天的计划计算每天更新一次。如果您的源表包含一年的数据，则可以将ARN用于此预定计算，并在不超过一年的每一天手动触发该计算，以便派生表填充所有历史查询。请注意，手动触发器的所有注意事项都适用于此处。此外，如果派生表的设置方式是历史数据提取将写入派生表上的磁存储，请注意写入磁存储的最佳做法和限制。

定时查询示例

本节包含一些示例，说明如何使用 Timestream for LiveAnalytics 的计划查询来优化成本和仪表板加载时间，同时可视化整个舰队的统计数据，从而有效监控您的设备群。Timestream 中的计划查询 LiveAnalytics 允许您使用 Timestream 的整个SQL表面区域来表达您的查询。 LiveAnalytics您的查询可以包含一个或多个源表，执行聚合或 Timestream SQL 语言允许的任何其他查询，然后将查询结果存储在 Timestream 的另一个目标表中。 LiveAnalytics LiveAnalytics

本节将定时查询的目标表称为派生表。

例如，我们将使用一个 DevOps 应用程序，在该应用程序中，您可以监控部署在多个部署（例如区域、单元和孤岛）、多个微服务中的大量服务器，并使用 Timestream for 跟踪整个队列的统计数据。 LiveAnalytics计划查询示例架构中描述了我们将使用的示例架构。

将描述以下场景。

- 如何转换仪表板，将您采集到 Timestream 的原始数据中的聚合统计数据绘制 LiveAnalytics 成计划查询，然后如何使用预先计算的聚合来创建显示汇总统计数据的新仪表板。
- 如何组合计划查询以获得聚合视图和原始粒度数据，以深入了解细节。这使您可以存储和分析原始数据，同时使用计划查询来优化整个车队的常见操作。
- 如何通过查找在多个仪表板中使用哪些聚合以及让相同的计划查询填充相同或多个仪表板中的多个面板，从而使用计划查询来优化成本。

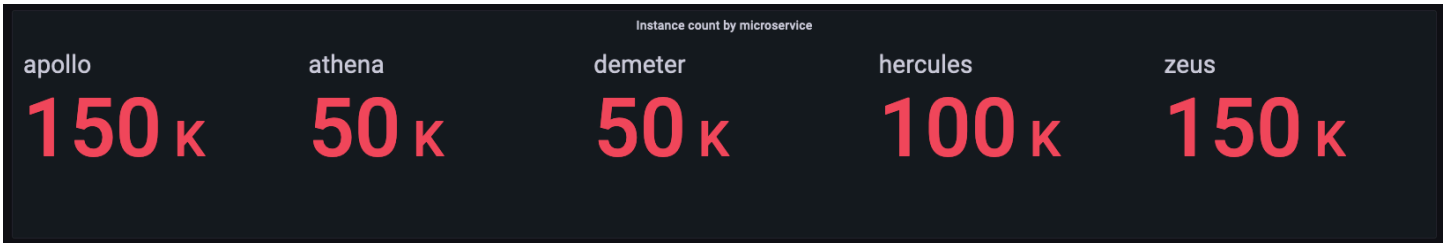
主题

- 将聚合仪表板转换为计划查询
- 使用计划查询和原始数据进行深入分析
- 通过在仪表板之间共享计划查询来优化成本
- 将基表上的查询与计划查询结果的查询进行比较

将聚合仪表板转换为计划查询

假设您正在计算队列范围内的统计信息，例如按五个微服务和部署服务的六个区域计算队列中的主机数量。从下面的快照中，你可以看到有 50 万台服务器发布指标，而一些较大的区域（例如 us-east-1）的服务器超过 20 万。

计算这些聚合，即计算数百 GB 数据的不同实例名称，除了扫描数据的成本外，还会导致数十秒的查询延迟。



原始仪表板查询

仪表板面板中显示的聚合是使用以下查询从原始数据计算出来的。该查询使用多个SQL结构，例如不同的计数和多个聚合函数。

```
SELECT CASE WHEN microservice_name = 'apollo' THEN num_instances ELSE NULL END AS
 apollo,
    CASE WHEN microservice_name = 'athena' THEN num_instances ELSE NULL END AS athena,
    CASE WHEN microservice_name = 'demeter' THEN num_instances ELSE NULL END AS
 demeter,
    CASE WHEN microservice_name = 'hercules' THEN num_instances ELSE NULL END AS
 hercules,
    CASE WHEN microservice_name = 'zeus' THEN num_instances ELSE NULL END AS zeus
FROM (
    SELECT microservice_name, SUM(num_instances) AS num_instances
    FROM (
        SELECT microservice_name, COUNT(DISTINCT instance_name) as num_instances
        FROM "raw_data"."devops"
        WHERE time BETWEEN from_milliseconds(1636526171043) AND
 from_milliseconds(1636612571043)
            AND measure_name = 'metrics'
        GROUP BY region, cell, silo, availability_zone, microservice_name
    )
    GROUP BY microservice_name
)
```

转换为计划查询

上一个查询可以转换为计划查询，如下所示。首先要计算区域、计算单元、孤岛、可用区和微服务中给定部署中的不同主机名。然后，将主机相加，计算出每小时每个微服务主机数。通过使用调度查询支持的@scheduled_runtime参数，您可以重新计算调用查询时过去一小时的参数。内部查询的 bin(@scheduled_runtime, 1h) in the WHERE 子句可确保即使将查询安排在中午的某个时间，您仍然可以获得整整一小时的数据。

尽管查询按小时计算聚合，但正如你将在计划计算配置中看到的那样，它仍设置为每半小时刷新一次，这样你就可以更快地在派生表中获得更新。您可以根据自己的新鲜度要求对其进行调整，例如，每 15 分钟重新计算一次聚合，或者在小时界限重新计算聚合。

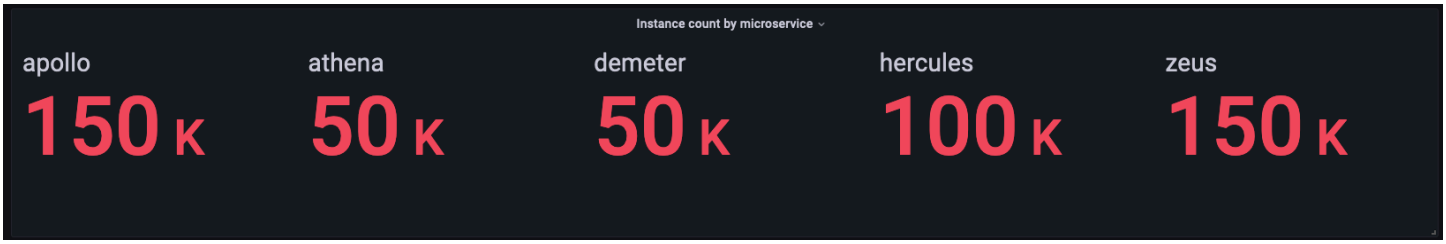
```
SELECT microservice_name, hour, SUM(num_instances) AS num_instances
FROM (
        SELECT microservice_name, bin(time, 1h) AS hour,
            COUNT(DISTINCT instance_name) as num_instances
       FROM raw_data.devops
       WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND @scheduled_runtime

           AND measure_name = 'metrics'
       GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)

     )
GROUP BY microservice_name, hour
```

```
{
    "Name": "MultiPT30mHostCountMicroservicePerHr",
    "QueryString": "SELECT microservice_name, hour, SUM(num_instances) AS num_instances
    FROM (        SELECT microservice_name, bin(time, 1h) AS hour, COUNT(DISTINCT
 instance_name) as num_instances        FROM raw_data.devops        WHERE time BETWEEN
 bin(@scheduled_runtime, 1h) - 1h AND @scheduled_runtime            AND measure_name
 = 'metrics'        GROUP BY region, cell, silo, availability_zone, microservice_name,
 bin(time, 1h)    )    GROUP BY microservice_name, hour",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "host_count_pt1h",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
```

```
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "num_instances",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "num_instances",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

在新仪表板中使用预先计算的结果

现在，您将看到如何使用您创建的计划查询中的派生表来创建聚合视图仪表板。通过仪表板快照，您还可以验证根据派生表和基表计算出的聚合是否也匹配。使用派生表创建仪表板后，您会注意到，与从原始数据计算这些聚合相比，使用派生表的加载时间明显更短，使用派生表的成本也更低。以下是使用预先计算的数据的仪表板快照，以及用于使用存储在 “派生” 表中的预计算数据呈现此面板的查询。” host_count_pt1h”。请注意，查询的结构与仪表板中使用的原始数据查询非常相似，不同之处在于它使用的派生表已经计算了该查询正在聚合的不同计数。




```
SELECT CASE WHEN microservice_name = 'apollo' THEN num_instances ELSE NULL END AS
 apollo,
    CASE WHEN microservice_name = 'athena' THEN num_instances ELSE NULL END AS athena,
    CASE WHEN microservice_name = 'demeter' THEN num_instances ELSE NULL END AS
 demeter,
```

```
    CASE WHEN microservice_name = 'hercules' THEN num_instances ELSE NULL END AS
 hercules,
    CASE WHEN microservice_name = 'zeus' THEN num_instances ELSE NULL END AS zeus
FROM (
    SELECT microservice_name, AVG(num_instances) AS num_instances
    FROM (
        SELECT microservice_name, bin(time, 1h), SUM(num_instances) as num_instances
        FROM "derived"."host_count_pt1h"
        WHERE time BETWEEN from_milliseconds(1636567785421) AND
 from_milliseconds(1636654185421)
            AND measure_name = 'num_instances'
        GROUP BY microservice_name, bin(time, 1h)
    )
    GROUP BY microservice_name
)
```

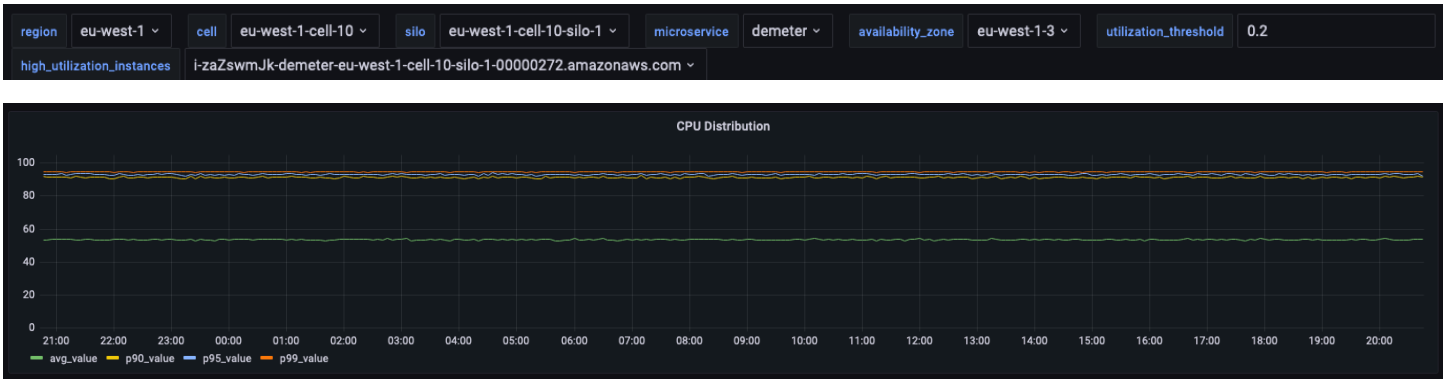
使用计划查询和原始数据进行深入分析

您可以使用整个车队的汇总统计数据来确定需要向下钻取的区域，然后使用原始数据向下钻取精细数据，以获得更深入的见解。

在此示例中，您将看到如何使用聚合仪表板来识别与其他部署相比，CPU利用率似乎更高的任何部署（部署适用于给定区域、单元、孤岛和可用区内的给定微服务）。然后，您可以使用原始数据进行深入研究，以便更好地理解。由于这些向下钻取可能很少见，并且只能访问与部署相关的数据，因此您可以使用原始数据进行此分析，而无需使用计划查询。

每次部署向下钻取

下面的仪表板提供了对给定部署中更精细的服务器级统计数据的深入分析。为了帮助您深入了解队列的不同部分，此仪表板使用了区域、单元、孤岛、微服务和 availability_zone 等变量。然后，它会显示该部署的一些汇总统计信息。



在下面的查询中，您可以看到在变量下拉列表中选择的值被用作查询WHERE子句中的谓词，这使您可以只关注部署的数据。然后，该面板绘制了该部署中实例的汇总CPU指标。您可以使用原始数据在交互式查询延迟的情况下进行深入分析，以获得更深入的见解。

```
SELECT bin(time, 5m) as minute,
    ROUND(AVG(cpu_user), 2) AS avg_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.9), 2) AS p90_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.95), 2) AS p95_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.99), 2) AS p99_value
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099476) AND
 from_milliseconds(1636613499476)
    AND region = 'eu-west-1'
    AND cell = 'eu-west-1-cell-10'
    AND silo = 'eu-west-1-cell-10-silo-1'
    AND microservice_name = 'demeter'
    AND availability_zone = 'eu-west-1-3'
    AND measure_name = 'metrics'
GROUP BY bin(time, 5m)
ORDER BY 1
```

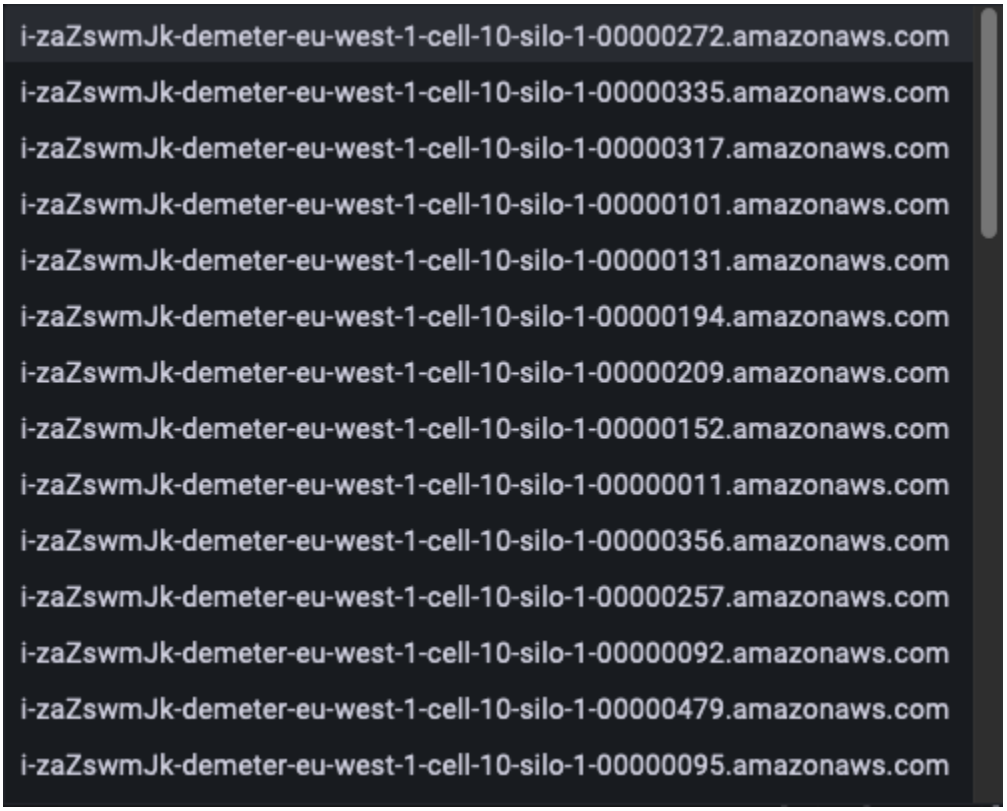
实例级统计信息

此仪表板进一步计算另一个变量，该变量还列出了CPU利用率较高的服务器/实例，按使用率降序排序。用于计算此变量的查询如下所示。

```
WITH microservice_cell_avg AS (
    SELECT AVG(cpu_user) AS microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE $__timeFilter
        AND measure_name = 'metrics'
        AND region = '${region}'
        AND cell = '${cell}'
        AND silo = '${silo}'
        AND availability_zone = '${availability_zone}'
        AND microservice_name = '${microservice}'
), instance_avg AS (
    SELECT instance_name,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE $__timeFilter
        AND measure_name = 'metrics'
```

```
        AND region = '${region}'
        AND cell = '${cell}'
        AND silo = '${silo}'
        AND microservice_name = '${microservice}'
        AND availability_zone = '${availability_zone}'
    GROUP BY availability_zone, instance_name
)
SELECT i.instance_name
FROM instance_avg i CROSS JOIN microservice_cell_avg m
WHERE i.instance_avg_metric > (1 + ${utilization_threshold}) *
 m.microservice_avg_metric
ORDER BY i.instance_avg_metric DESC
```

在前面的查询中，变量是根据为其他变量选择的值动态重新计算的。为部署填充变量后，您可以从列表中选择单个实例，以进一步可视化该实例的指标。您可以从实例名称的下拉列表中选择不同的实例，如下面的快照所示。



前面的面板显示所选实例的统计信息，下面是用于获取这些统计信息的查询。

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(cpu_user) AS avg_cpu,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.99), 2) as p99_cpu
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499477)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(memory_used) AS avg_memory,
    ROUND(APPROX_PERCENTILE(memory_used, 0.99), 2) as p99_memory
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499477)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
```

```
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

```
SELECT COUNT(gc_pause)
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'events'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
```

```
SELECT avg(gc_pause) as avg, round(approx_percentile(gc_pause, 0.99), 2) as p99
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099478) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'events'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
```

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(disk_io_reads) AS avg,
    ROUND(APPROX_PERCENTILE(disk_io_reads, 0.99), 2) as p99
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099478) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

通过在仪表板之间共享计划查询来优化成本

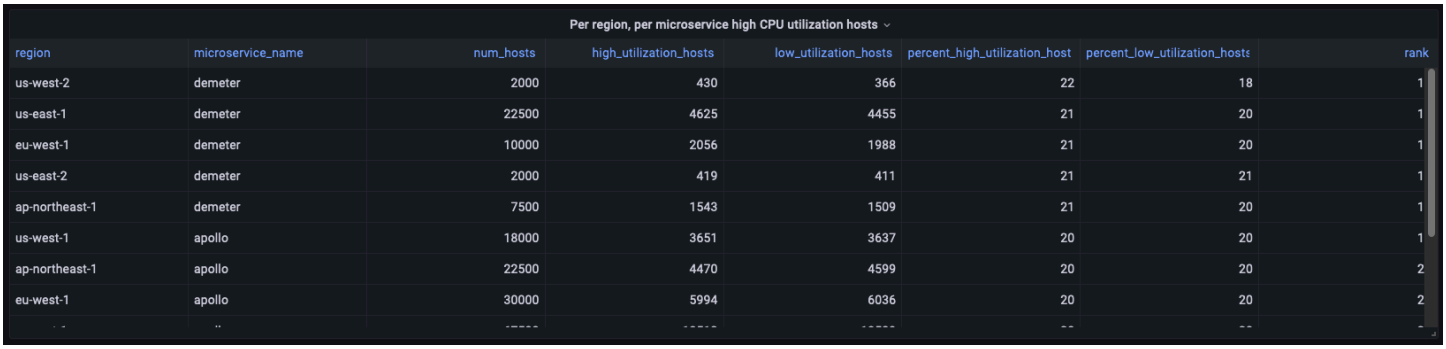
在此示例中，我们将看到一个场景，即多个仪表板面板显示相似信息的变体（查找高CPU主机和机群中CPU利用率高的部分），以及如何使用相同的计划查询来预先计算结果，然后使用这些结果填充多个面板。这种重复使用进一步优化了您的成本，即您不使用不同的计划查询，每个面板各一个，而只使用所有者。

包含原始数据的仪表板面板

CPU每个微服务的每个区域的利用率

第一个面板计算在区域、计算单元、孤岛、可用区和微服务中给定部署中，其平均CPUCPU利用率低于或高于上述利用率的阈值的实例。然后，它会对利用率最高的主机占比最高的区域和微服务进行排序。它有助于确定特定部署的服务器的运行热度，然后深入研究以更好地了解问题。

面板的查询展示了 Timestream f LiveAnalytics or SQL 支持使用常用表表达式、窗口函数、联接等执行复杂分析任务的灵活性。



Per region, per microservice high CPU utilization hosts

region	microservice_name	num_hosts	high_utilization_hosts	low_utilization_hosts	percent_high_utilization_host	percent_low_utilization_hosts	rank
us-west-2	demeter	2000	430	366	22	18	1
us-east-1	demeter	22500	4625	4455	21	20	1
eu-west-1	demeter	10000	2056	1988	21	20	1
us-east-2	demeter	2000	419	411	21	21	1
ap-northeast-1	demeter	7500	1543	1509	21	20	1
us-west-1	apollo	18000	3651	3637	20	20	1
ap-northeast-1	apollo	22500	4470	4599	20	20	2
eu-west-1	apollo	30000	5994	6036	20	20	2

查询：

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, AVG(cpu_user) AS
 microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526593876) AND
 from_milliseconds(1636612993876)
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name
), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526593876) AND
 from_milliseconds(1636612993876)
```

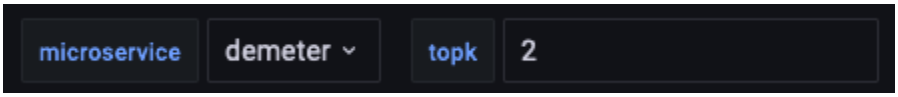
```
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name
), instances_above_threshold AS (
  SELECT i.*,
    CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS high_utilization,
    CASE WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS low_utilization
  FROM instance_avg i INNER JOIN microservice_cell_avg m
    ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
      AND m.microservice_name = i.microservice_name
), per_deployment_high AS (
SELECT region, microservice_name, COUNT(*) AS num_hosts, SUM(high_utilization) AS
 high_utilization_hosts, SUM(low_utilization) AS low_utilization_hosts,
    ROUND(SUM(high_utilization) * 100.0 / COUNT(*), 0) AS
 percent_high_utilization_hosts,
    ROUND(SUM(low_utilization) * 100.0 / COUNT(*), 0) AS percent_low_utilization_hosts
FROM instances_above_threshold
GROUP BY region, microservice_name
), per_region_ranked AS (
    SELECT *,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY percent_high_utilization_hosts
 DESC, high_utilization_hosts DESC) AS rank
    FROM per_deployment_high
)
SELECT *
FROM per_region_ranked
WHERE rank <= 2
ORDER BY percent_high_utilization_hosts desc, rank asc
```

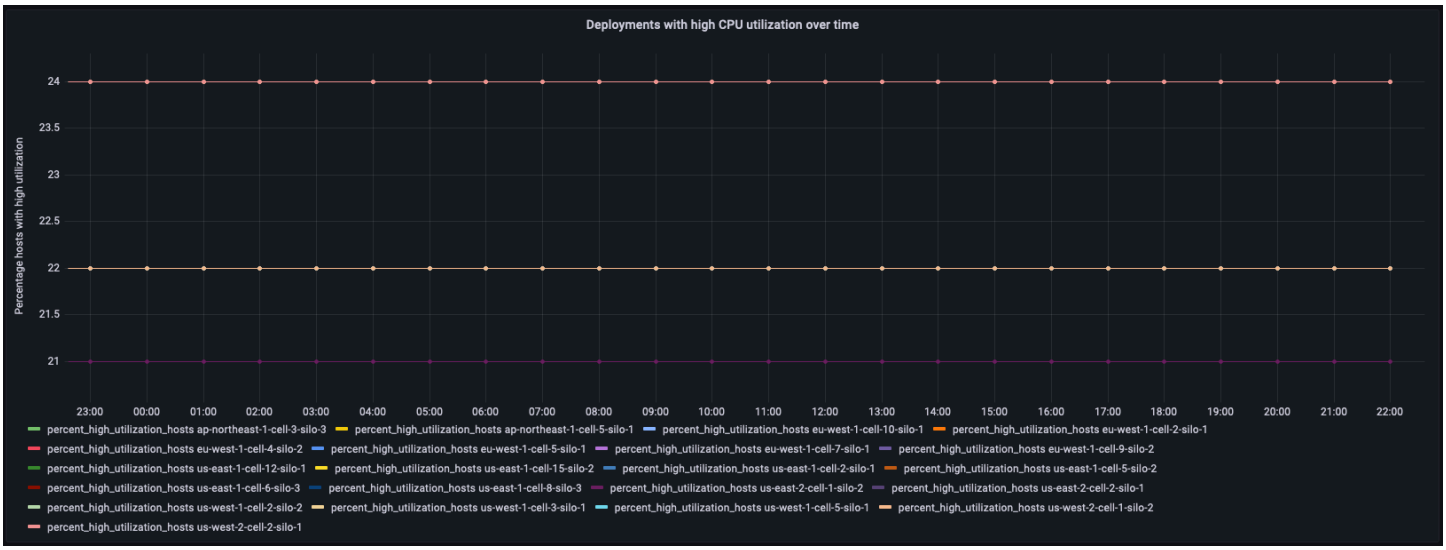
深入研究微服务以查找热点

下一个仪表板允许您更深入地研究其中一个微服务，以找出该微服务的特定区域、单元和孤岛，该微服务正在以更高的利用率运行其队列中的一小部分。CPU例如，在舰队范围的仪表板中，你看到微服务 demeter 出现在排名前几个的位置，所以在这个仪表板中，你想更深入地研究该微服务。

此仪表板使用变量来选择要深入研究的微服务，变量的值使用该维度的唯一值填充。选择微服务后，仪表板的其余部分将刷新。

如下所示，第一个面板绘制了一段时间内部署（微服务的区域、单元和孤岛）中主机的百分比，以及用于绘制仪表板的相应查询。此图本身标识了具有较高比例的主机占比高的特定部署CPU。





查询：

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, bin(time, 1h) as
 hour, AVG(cpu_user) AS microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526898831) AND
 from_milliseconds(1636613298831)
        AND measure_name = 'metrics'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)
), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h) as hour,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526898831) AND
 from_milliseconds(1636613298831)
        AND measure_name = 'metrics'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h)
), instances_above_threshold AS (
  SELECT i.*,
    CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS high_utilization
```

```
  FROM instance_avg i INNER JOIN microservice_cell_avg m
    ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
      AND m.microservice_name = i.microservice_name AND m.hour = i.hour
), high_utilization_percent AS (
    SELECT region, cell, silo, microservice_name, hour, COUNT(*) AS num_hosts,
 SUM(high_utilization) AS high_utilization_hosts,
        ROUND(SUM(high_utilization) * 100.0 / COUNT(*), 0) AS
 percent_high_utilization_hosts
    FROM instances_above_threshold
    GROUP BY region, cell, silo, microservice_name, hour
), high_utilization_ranked AS (
    SELECT region, cell, silo, microservice_name,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY
 AVG(percent_high_utilization_hosts) desc, AVG(high_utilization_hosts) desc) AS rank
    FROM high_utilization_percent
    GROUP BY region, cell, silo, microservice_name
)
SELECT hup.silo, CREATE_TIME_SERIES(hour, hup.percent_high_utilization_hosts) AS
 percent_high_utilization_hosts
FROM high_utilization_percent hup INNER JOIN high_utilization_ranked hur
    ON hup.region = hur.region AND hup.cell = hur.cell AND hup.silo = hur.silo AND
 hup.microservice_name = hur.microservice_name
WHERE rank <= 2
GROUP BY hup.region, hup.cell, hup.silo
ORDER BY hup.silo
```

转换为单个计划查询以实现重复使用

值得注意的是，在两个仪表板的不同面板上进行了类似的计算。您可以为每个面板定义单独的计划查询。在这里，您将看到如何通过定义一个预定查询来进一步优化成本，该查询的结果可用于呈现所有三个面板。

以下是捕获计算并用于所有不同面板的聚合的查询。您将在此计划查询的定义中观察到几个重要方面。

- 定时查询支持的灵活性和强大功能，您可以在其中使用公用表表达式、联接、案例语句等。SQL
- 您可以使用一个计划查询来计算比特定仪表板可能需要的更精细的粒度以及仪表板可能用于不同变量的所有值的统计数据。例如，您将看到聚合是跨区域、单元、筒仓和微服务计算的。因此，您可以将它们组合起来创建区域级聚合、区域聚合和微服务级聚合。同样，同一个查询会计算所有区域、单元、孤岛和微服务的聚合。它允许您对这些列应用筛选器，以获取值子集的聚合。例如，你可以计算任何一个区域（比如 us-east-1）的聚合，也可以计算任何一个微服务（比如 demeter）的聚合，也

可以深入研究一个区域、单元、孤岛和微服务中的特定部署。这种方法进一步优化了维护预先计算的聚合的成本。

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, bin(time, 1h) as
 hour, AVG(cpu_user) AS microservice_avg_metric
    FROM raw_data.devops
    WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h)
 + 1h
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)

), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h) as hour,
        AVG(cpu_user) AS instance_avg_metric
   FROM raw_data.devops
   WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h)
 + 1h
       AND measure_name = 'metrics'
   GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h)
), instances_above_threshold AS (
    SELECT i.*,
        CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1
 ELSE 0 END AS high_utilization,
        CASE WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1
 ELSE 0 END AS low_utilization
   FROM instance_avg i INNER JOIN microservice_cell_avg m
       ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
           AND m.microservice_name = i.microservice_name AND m.hour = i.hour
)
SELECT region, cell, silo, microservice_name, hour,
     COUNT(*) AS num_hosts, SUM(high_utilization) AS high_utilization_hosts,
 SUM(low_utilization) AS low_utilization_hosts
FROM instances_above_threshold GROUP BY region, cell, silo, microservice_name, hour
```

以下是上一个查询的计划查询定义。调度表达式配置为每 30 分钟刷新一次，并将数据刷新最多一个小时，再次使用 bin (@scheduled_runtime, 1h) 构造来获取整整一小时的事件。根据应用程序的新鲜度要求，您可以将其配置为更高或更少的刷新频率。通过使用 t WHERE ime BETWEEN bin

(@scheduled_runtime，1h)-1h AND bin (@scheduled_runtime，1h) + 1h，我们可以确保即使你每 15 分钟刷新一次，你也能获得当前小时和前一小时的整小时数据。

稍后，您将看到这三个面板如何使用写入表 deployment_cpu_stats_per_hr 的这些聚合来可视化与面板相关的指标。

```
{
    "Name": "MultiPT30mHighCpuDeploymentsPerHr",
    "QueryString": "WITH microservice_cell_avg AS (    SELECT region, cell,
 silo, availability_zone, microservice_name, bin(time, 1h) as hour, AVG(cpu_user)
 AS microservice_avg_metric    FROM raw_data.devops    WHERE time BETWEEN
 bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h) + 1h    AND
 measure_name = 'metrics'    GROUP BY region, cell, silo, availability_zone,
 microservice_name, bin(time, 1h)    ), instance_avg AS (    SELECT region,
 cell, silo, availability_zone, microservice_name, instance_name, bin(time, 1h)
 as hour,    AVG(cpu_user) AS instance_avg_metric    FROM raw_data.devops
 WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime,
 1h) + 1h    AND measure_name = 'metrics'    GROUP BY region, cell, silo,
 availability_zone, microservice_name, instance_name, bin(time, 1h)    ),
 instances_above_threshold AS (    SELECT i.*,    CASE WHEN i.instance_avg_metric >
 (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE 0 END AS high_utilization,    CASE
 WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1 ELSE 0 END
 AS low_utilization    FROM instance_avg i INNER JOIN microservice_cell_avg m    ON
 i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND i.availability_zone
 = m.availability_zone    AND m.microservice_name = i.microservice_name AND m.hour =
 i.hour    )    SELECT region, cell, silo, microservice_name, hour,         COUNT(*)
 AS num_hosts, SUM(high_utilization) AS high_utilization_hosts, SUM(low_utilization) AS
 low_utilization_hosts    FROM instances_above_threshold GROUP BY region, cell, silo,
 microservice_name, hour",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "deployment_cpu_stats_per_hr",
            "TimeColumn": "hour",
            "DimensionMappings": [
```

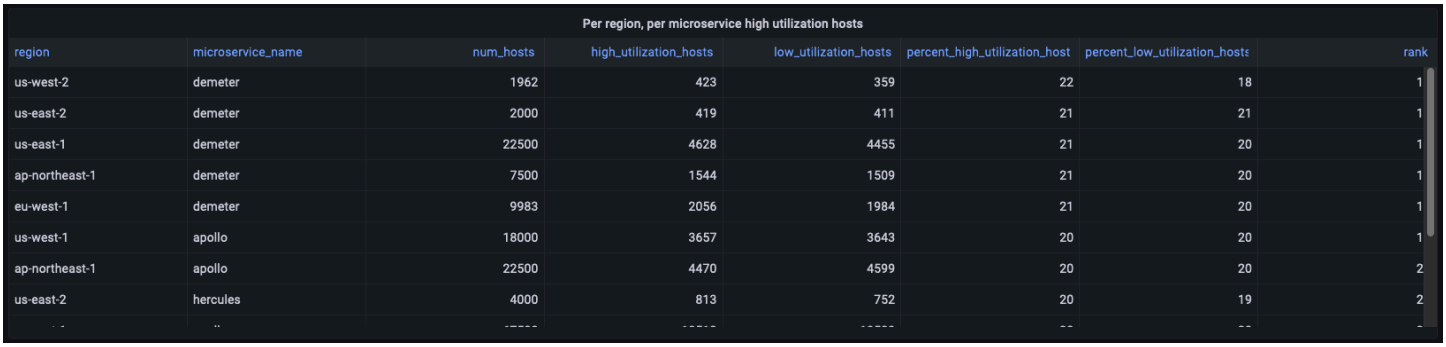
```
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "cpu_user",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "num_hosts",
                        "MeasureValueType": "BIGINT"
                    },
                    {
                        "SourceColumn": "high_utilization_hosts",
                        "MeasureValueType": "BIGINT"
                    },
                    {
                        "SourceColumn": "low_utilization_hosts",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
```

```
}
```

来自预先计算结果的仪表板

高CPU利用率主机

对于高利用率主机，您将看到不同的面板如何使用来自 deployment_cpu_stats_per_hr 的数据来计算面板所需的不同聚合。例如，该面板提供区域级信息，因此它报告按区域和微服务分组的聚合，而不筛选任何区域或微服务。



Per region, per microservice high utilization hosts

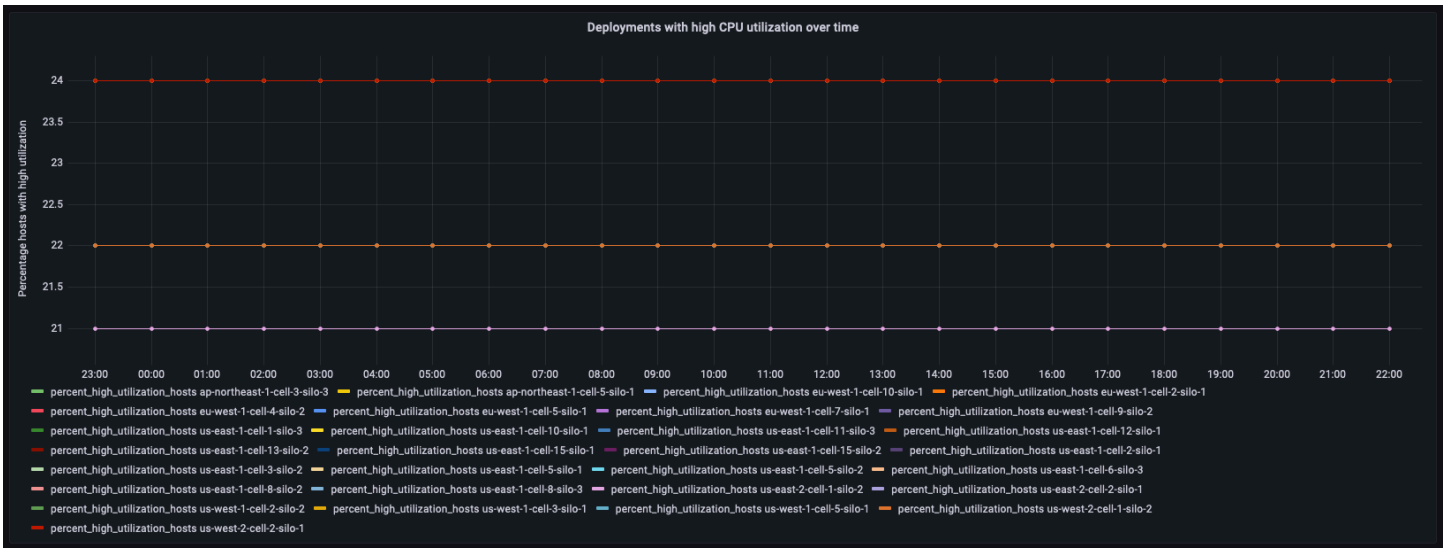
region	microservice_name	num_hosts	high_utilization_hosts	low_utilization_hosts	percent_high_utilization_host	percent_low_utilization_hosts	rank
us-west-2	demeter	1962	423	359	22	18	1
us-east-2	demeter	2000	419	411	21	21	1
us-east-1	demeter	22500	4628	4455	21	20	1
ap-northeast-1	demeter	7500	1544	1509	21	20	1
eu-west-1	demeter	9983	2056	1984	21	20	1
us-west-1	apollo	18000	3657	3643	20	20	1
ap-northeast-1	apollo	22500	4470	4599	20	20	2
us-east-2	hercules	4000	813	752	20	19	2

```
WITH per_deployment_hosts AS (
    SELECT region, cell, silo, microservice_name,
        AVG(num_hosts) AS num_hosts,
        AVG(high_utilization_hosts) AS high_utilization_hosts,
        AVG(low_utilization_hosts) AS low_utilization_hosts
    FROM "derived"."deployment_cpu_stats_per_hr"
    WHERE time BETWEEN from_milliseconds(1636567785437) AND
 from_milliseconds(1636654185437)
        AND measure_name = 'cpu_user'
    GROUP BY region, cell, silo, microservice_name
), per_deployment_high AS (
    SELECT region, microservice_name,
        SUM(num_hosts) AS num_hosts,
        ROUND(SUM(high_utilization_hosts), 0) AS high_utilization_hosts,
        ROUND(SUM(low_utilization_hosts),0) AS low_utilization_hosts,
        ROUND(SUM(high_utilization_hosts) * 100.0 / SUM(num_hosts)) AS
 percent_high_utilization_hosts,
        ROUND(SUM(low_utilization_hosts) * 100.0 / SUM(num_hosts)) AS
 percent_low_utilization_hosts
    FROM per_deployment_hosts
    GROUP BY region, microservice_name
),
per_region_ranked AS (
    SELECT *,
```

```
        DENSE_RANK() OVER (PARTITION BY region ORDER BY percent_high_utilization_hosts
 DESC, high_utilization_hosts DESC) AS rank
    FROM per_deployment_high
)
SELECT *
FROM per_region_ranked
WHERE rank <= 2
ORDER BY percent_high_utilization_hosts desc, rank asc
```

深入研究微服务以查找高CPU使用率部署

下一个示例再次使用 deployment_cpu_stats_per_hr 派生表，但现在对特定的微服务应用了筛选器（在本例中为 demeter，因为它在聚合仪表板中报告了高利用率主机）。此面板跟踪一段时间内高CPU利用率主机的百分比。




```
WITH high_utilization_percent AS (
    SELECT region, cell, silo, microservice_name, bin(time, 1h) AS hour, MAX(num_hosts)
 AS num_hosts,
        MAX(high_utilization_hosts) AS high_utilization_hosts,
        ROUND(MAX(high_utilization_hosts) * 100.0 / MAX(num_hosts)) AS
 percent_high_utilization_hosts
    FROM "derived"."deployment_cpu_stats_per_hr"
    WHERE time BETWEEN from_milliseconds(1636525800000) AND
 from_milliseconds(1636612200000)
        AND measure_name = 'cpu_user'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, microservice_name, bin(time, 1h)
), high_utilization_ranked AS (
    SELECT region, cell, silo, microservice_name,
```

```
        DENSE_RANK() OVER (PARTITION BY region ORDER BY
 AVG(percent_high_utilization_hosts) desc, AVG(high_utilization_hosts) desc) AS rank
    FROM high_utilization_percent
    GROUP BY region, cell, silo, microservice_name
)
SELECT hup.silo, CREATE_TIME_SERIES(hour, hup.percent_high_utilization_hosts) AS
 percent_high_utilization_hosts
FROM high_utilization_percent hup INNER JOIN high_utilization_ranked hur
    ON hup.region = hur.region AND hup.cell = hur.cell AND hup.silo = hur.silo AND
 hup.microservice_name = hur.microservice_name
WHERE rank <= 2
GROUP BY hup.region, hup.cell, hup.silo
ORDER BY hup.silo
```

将基表上的查询与计划查询结果的查询进行比较

在这个 Timestream 查询示例中，我们使用以下架构、示例查询和输出将基表上的查询与对计划查询结果派生表的查询进行比较。通过精心策划的计划查询，您可以获得行数更少且具有其他特征的派生表，这些特征可以使查询速度比原始基表更快。

有关描述此场景的视频，请参阅[在 Amazon Timestream 中使用计划查询提高查询性能并降低成本](#)。LiveAnalytics

在本示例中，我们使用以下场景：

- 区域 — us-east-1
- 基本表 — `"clickstream"."shopping"`
- 派生表 — `"clickstream"."aggregate"`

基表

下面描述了基表的架构。

列	类型	LiveAnalytics 属性类型的时间流
channel	varchar	MULTI
description	varchar	MULTI

列	类型	LiveAnalytics 属性类型的时间流
事件	varchar	DIMENSION
ip_address	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
product	varchar	MULTI
product_id	varchar	MULTI
quantity	double	MULTI
查询	varchar	MULTI
session_id	varchar	DIMENSION
用户组	varchar	DIMENSION
user_id	varchar	DIMENSION

下面描述了基表的度量。基表是指在 Timestream 中运行定时查询的表。

- 测量名称 — `metrics`
- 数据-多
- 尺寸：

```
[ ( user_group, varchar ),( user_id, varchar ),( session_id, varchar ),( ip_address,
 varchar ),( event, varchar ) ]
```

在基表上查询

以下是一个临时查询，它按给定时间范围内按照 5 分钟的聚合时间收集计数。

```
SELECT BIN(time, 5m) as time,
channel,
product_id,
```

```
SUM(quantity) as product_quantity
FROM "clickstream"."shopping"
WHERE BIN(time, 5m) BETWEEN '2023-05-11 10:10:00.000000000' AND '2023-05-11
 10:30:00.000000000'
AND channel = 'Social media'
and product_id = '431412'
GROUP BY BIN(time, 5m),channel,product_id
```

输出：

```
duration:1.745 sec
Bytes scanned: 29.89 MB
Query Id: AEBQEANMHG7MHHBHCKJ3BSOE3QUGIDBGWCCP5I6J6YUW5CVJZ2M3JCJ27QRMM7A
Row count:5
```

定时查询

以下是每 5 分钟运行一次的定时查询。

```
SELECT BIN(time, 5m) as time, channel as measure_name, product_id, product,
SUM(quantity) as product_quantity
FROM "clickstream"."shopping"
WHERE time BETWEEN BIN(@scheduled_runtime, 5m) - 10m AND BIN(@scheduled_runtime, 5m) -
 5m
AND channel = 'Social media'
GROUP BY BIN(time, 5m), channel, product_id, product
```

对派生表进行查询

以下是对派生表的临时查询。派生表是指包含定时查询结果的时间流表。

```
SELECT time, measure_name, product_id,product_quantity
FROM "clickstream"."aggregate"
WHERE time BETWEEN '2023-05-11 10:10:00.000000000' AND '2023-05-11 10:30:00.000000000'
AND measure_name = 'Social media'
and product_id = '431412'
```

输出：

```
duration: 0.2960 sec
Bytes scanned: 235.00 B
QueryID: AEBQEANMHHAAQU4FFTT6CFM6UYXTL4SMLZV22MFP4KV2Z7IRVOPLOMLDD6BR33Q
```

```
Row count: 5
```

比较

以下是对基表的查询结果与对派生表的查询结果的比较。对通过计划查询完成的聚合结果的派生表的相同查询完成速度更快，扫描的字节更少。

这些结果显示了使用计划查询聚合数据以加快查询速度的价值。

	在基表上查询	对派生表进行查询
持续时间	1.745 秒	0.2960 秒
扫描的字节数	29.89 MB	235 个字节
行数	5	5

UNLOAD用于将查询结果从 Timestream 导出到 S3 LiveAnalytics

LiveAnalytics 目前，Amazon Timestream 允许您使用该语句以经济实惠且安全的方式将查询结果导出到 Amazon S3。UNLOAD使用该UNLOAD语句，您现在可以以 Apache Parquet 或逗号分隔值 (CSV) 格式将时序数据导出到选定的 S3 存储桶，从而可以灵活地与其他服务一起存储、合并和分析您的时间序列数据。该UNLOAD语句允许您以压缩方式导出数据，从而减少传输的数据和所需的存储空间。UNLOAD还支持在导出数据时根据所选属性进行分区，从而提高性能并缩短下游服务访问数据的处理时间。此外，您可以使用 Amazon S3 托管密钥 (SSE-S3) 或AWS密钥管理服务 (AWS KMS) 托管密钥 (SSE-KMS) 来加密导出的数据。

UNLOAD来自 Timestream 的好处 LiveAnalytics

使用该UNLOAD语句的主要好处如下。

- 操作简便 — 使用该UNLOAD语句，您可以在单个查询请求中以 Apache Parque CSV t 或格式导出千兆字节的数据，从而可以灵活地选择最适合下游处理需求的格式，并使构建数据湖变得更加容易。
- 安全且具有成本效益 — UNLOAD 声明提供了以压缩方式将数据导出到 S3 存储桶以及使用客户托管密钥加密（SSE-KMS 或 SSE _S3）数据的功能，从而降低数据存储成本并防止未经授权的访问。
- 性能-使用UNLOAD语句，您可以在导出到 S3 存储桶时对数据进行分区。对数据进行分区使下游服务能够并行处理数据，从而缩短其处理时间。此外，下游服务只能处理他们需要的数据，从而减少了所需的处理资源，从而减少了相关成本。

来UNLOAD自 Timestream 的用例 LiveAnalytics

您可以使用UNLOAD语句将数据写入您的 S3 存储桶到以下内容。

- 构建数据仓库 — 您可以将千兆字节的查询结果导出到 S3 存储桶中，更轻松地将时序数据添加到数据湖中。您可以使用诸如 Amazon Athena 和 Amazon Redshift 之类的服务将您的时间序列数据与其他相关数据相结合，从而得出复杂的业务见解。
- 构建 AI 和 ML 数据管道 — 该UNLOAD语句使您能够轻松地为访问时序数据的机器学习模型构建数据管道，从而更轻松地在 Amazon SageMaker 和 Amazon 等服务中使用时序数据EMR。
- 简化ETL处理 — 将数据导出到 S3 存储桶可以简化对数据执行提取、转换、加载 (ETL) 操作的过程，使您能够无缝使用第三方工具或 AWS 服务（例如 AWS Glue）来处理和转换数据。

UNLOAD概念

语法

```
UNLOAD (SELECT statement)
 TO 's3://bucket-name/folder'
 WITH ( option = expression [, ...] )
```

在option哪里

```
{ partitioned_by = ARRAY[ col_name[,…] ]
 | format = [ '{ CSV | PARQUET }' ]
 | compression = [ '{ GZIP | NONE }' ]
 | encryption = [ '{ SSE_KMS | SSE_S3 }' ]
 | kms_key = '<string>'
 | field_delimiter ='<character>'
 | escaped_by = '<character>'
 | include_header = ['{true, false}']
 | max_file_size = '<value>'
 | }
```

参数

SELECT声明

用于从一个或多个 Timestream 中为 LiveAnalytics 表选择和检索数据的查询语句。

```
(SELECT column 1, column 2, column 3 from database.table
      where measure_name = "ABC" and timestamp between ago (1d) and now() )
```

收件人条款

```
TO 's3://bucket-name/folder'
```

或者

```
TO 's3://access-point-alias/folder'
```

语TO句中的子UNLOAD句指定查询结果输出的目的地。您需要提供完整路径，包括 Amazon S3 存储桶名称或 Amazon S3，以及在 Amazon S3 access-point-alias 上 LiveAnalytics写入输出文件对象的 Timestream 上的文件夹位置。S3 存储桶应由同一账户所有且位于同一区域。除了查询结果集之外，Timestream 还会将清单和元数据文件 LiveAnalytics 写入指定的目标文件夹。

PARTITIONED_BY 子句

```
partitioned_by = ARRAY [col_name[,…] , (default: none)
```

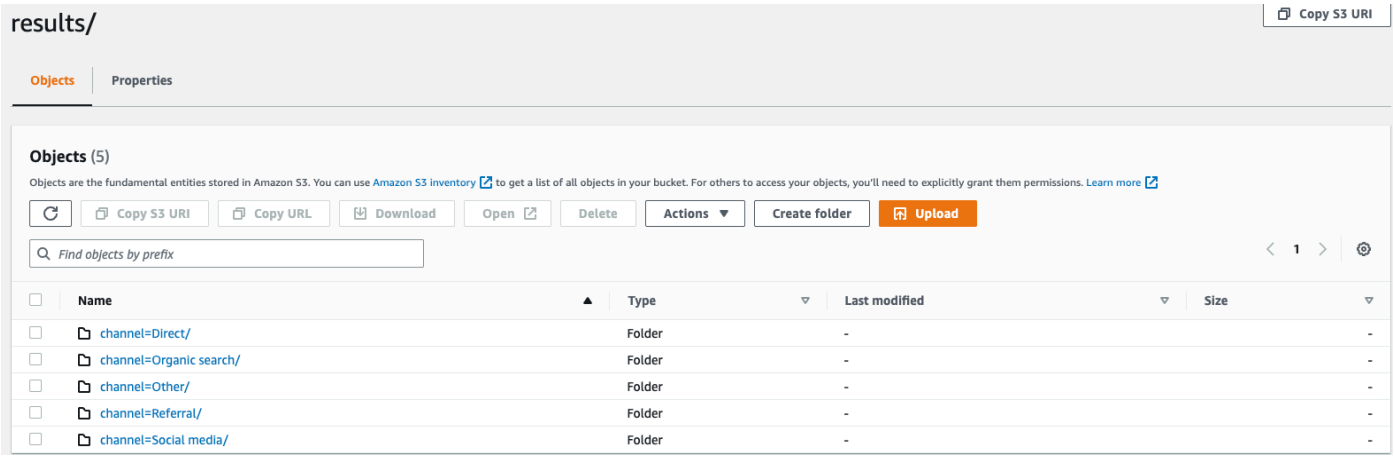
查询中使用该partitioned_by子句对数据进行精细分组和分析。将查询结果导出到 S3 存储桶时，您可以选择根据选择查询中的一列或多列对数据进行分区。对数据进行分区时，根据分区列将导出的数据分成子集，每个子集存储在单独的文件夹中。在包含导出数据的结果文件夹中，将自动创建一个folder/results/partition column = partition value/子文件夹。但是，请注意，分区列不包含在输出文件中。

partitioned_by不是语法中的必备子句。如果您选择在不进行任何分区的情况下导出数据，则可以在语法中排除该子句。

Example

假设您正在监控网站的点击流数据，并且有 5 个流量渠道direct，即Social Media、Organic SearchOther、和Referral。导出数据时，您可以选择使用列对数据进行分区Channel。在您的数据文件夹中s3://bucketname/results，您将有五个文件夹，每个文件夹都有各自的频道名称，例如，s3://bucketname/results/channel=Social Media/.在此文件夹中，您将找到通过该Social Media渠道登陆您网站的所有客户的数据。同样，剩下的频道还有其他文件夹。

按频道列分区的导出数据



FORMAT

```
format = [ '{ CSV | PARQUET }' , default: CSV
```

用于指定写入 S3 存储桶的查询结果格式的关键字。您可以使用逗号 (,) 作为默认分隔符将数据导出为逗号分隔值 (CSV)，也可以以 Apache Parquet 格式（一种用于分析的高效开放列式存储格式）导出。

COMPRESSION

```
compression = [ '{ GZIP | NONE }' ], default: GZIP
```

您可以使用压缩算法压缩导出的数据，GZIP也可以通过指定NONE选项将其解压缩。

ENCRYPTION

```
encryption = [ '{ SSE_KMS | SSE_S3 }' ], default: SSE_S3
```

Amazon S3 上的输出文件使用您选择的加密选项进行加密。除了您的数据外，还会根据您选择的加密选项对清单文件和元数据文件进行加密。我们目前支持 SSE _S3 和 SSE _ KMS 加密。SSE_S3 是一种服务器端加密，Amazon S3 使用 256 位高级加密标准 () 加密对数据进行加密。AESSSE_ KMS 是一种服务器端加密，用于使用客户管理的密钥对数据进行加密。

KMS_KEY

```
kms_key = '<string>'
```

KMS密钥是客户定义的密钥，用于加密导出的查询结果。KMS密钥由密 AWS 钥管理服务 (AWS KMS) 安全管理，用于加密 Amazon S3 上的数据文件。

FIELD_DELIMITER

```
field_delimiter ='<character>' , default: (,)
```

以CSV格式导出数据时，此字段指定一个ASCII字符用于分隔输出文件中的字段，例如竖线字符 (|)、逗号 (,) 或制表符 (/t)。CSV文件的默认分隔符是逗号字符。如果数据中的值包含选定的分隔符，则分隔符将用引号字符引用。例如，如果您的数据中的值包含Time,stream，则该值将像在导出数据"Time,stream"中一样被引用。Timestream 使用的引号字符 LiveAnalytics 是双引号 (“)。

FIELD_DELIMITER如果要在中包含标题，请避免将回车符（ASCII130D，十六进制，文本 '\ r'）或换行符（ASCII10，十六进制 0A，文本'\ n'）指定为CSV，因为这将使许多解析器无法在生成的输出中正确解析标题。CSV

ESCAPED_by

```
escaped_by = '<character>', default: (\)
```

以CSV格式导出数据时，此字段指定写入 S3 存储桶的数据文件中应被视为转义字符的字符。逃跑发生在以下场景中：

1. 如果值本身包含引号字符 (“)，则将使用转义字符对其进行转义。例如，如果值为Time"stream，其中 (\) 是配置的转义字符，则将其转义为Time\"stream。
2. 如果该值包含配置的转义字符，则会对其进行转义。例如，如果值为Time\stream，则将其转义为Time\\stream。

> **Note**
>
> 如果导出的输出包含诸如数组、行或时间序列之类的复杂数据类型，则会将其序列化为JSON字符串。以下为示例。

数据类型	实际价值	如何CSV以 [序列化JSON字符串] 格式对值进行转义
数组	[23,24,25]	"[23,24,25]"
行	(x=23.0, y=hello)	"{\"x\":23.0,\"y\":\"hello\"}"

数据类型	实际价值	如何CSV以 [序列化JSON字符串] 格式对值进行转义
时间序列	`[ ( time=1970-01-01 00:00:00.000000010 , value=100.0 ), ( time=1970-01-01 00:00:00.000000012, value=120.0 ) ]`	`"[{\"time\":\"1970-01-01 00:00:00.000000010Z\",\"value\":100.0},{\"time\":\"1970-01-01 00:00:00.000000012Z\",\"value\":120.0}]"`

INCLUDE_HEADER

```
include_header = 'true' , default: 'false'
```

以CSV格式导出数据时，此字段允许您将列名作为导出CSV数据文件的第一行。

接受的值为 “真” 和 “假”，默认值为 “假”。诸如`escaped_by`和之类的文本转换选项也`field_delimiter`适用于标题。

> **Note**
>
> 包括标题时，请务必不要选择回车符（ASCII13，十六进制 0D，文本 '\r'）或换行符（ASCII10，十六进制 0A，文本'\n'）作为标题`FIELD_DELIMITER`，因为这将使许多解析器无法在生成的输出中正确解析标题。CSV

MAX_FILE_SIZE

```
max_file_size = 'X[MB|GB]' , default: '78GB'
```

此字段指定该UNLOAD语句在 Amazon S3 中创建的最大文件大小。该UNLOAD语句可以创建多个文件，但是写入 Amazon S3 的每个文件的最大大小将接近该字段中指定的大小。

该字段的值必须介于 16 MB 到 78 GB 之间（含）。你可以用整数（例如）或小数（如`0.5GB`或`24.7MB`）来指定。`12GB`默认值为 78 GB。

实际文件大小是写入文件时的近似值，因此实际最大大小可能不完全等于您指定的数字。

我的 S3 存储桶中写入了什么？

对于每个成功执行的UNLOAD查询，Timestream for 都会将您的查询结果、元数据文件和清单文件 LiveAnalytics 写入 S3 存储桶。如果您已对数据进行分区，则结果文件夹中包含所有分区文件夹。清单文件包含UNLOAD命令写入的文件列表。元数据文件包含描述写入数据的特征、属性和属性的信息。

导出的文件名是什么？

导出的文件名包含两个部分，第一个组成部分是 queryID，第二个组成部分是唯一标识符。

CSV文件

```
S3://bucket_name/results/<queryid>_<UUID>.csv
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.csv
```

压缩CSV文件

```
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.gz
```

实木复合文件

```
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.parquet
```

元数据和清单文件

```
S3://bucket_name/<queryid>_<UUID>_manifest.json
S3://bucket_name/<queryid>_<UUID>_metadata.json
```

由于CSV格式中的数据存储在文件级别，因此在导出到 S3 时压缩数据时，该文件的扩展名将为 “.gz”。但是，Parquet 中的数据是在列级别压缩的，因此即使在导出时压缩数据，文件仍将具有.parquet 扩展名。

每个文件包含什么信息？

清单文件

清单文件提供有关UNLOAD执行时导出的文件列表的信息。清单文件可在提供的 S3 存储桶中找到，文件名为:`s3://<bucket_name>/<queryid>_<UUID>_manifest.json`。清单文件将包含结果文件

夹中文件的 URL、相应文件的记录数和大小以及查询元数据（即为查询导出到 S3 的总字节数和总行数）。

```
{
  "result_files": [
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 32295,
                "row_count": 10
            }
    },
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 62295,
                "row_count": 20
            }
    },
  ],
  "query_metadata":
    {
      "content_length_in_bytes": 94590,
      "total_row_count": 30,
      "result_format": "CSV",
      "result_version": "Amazon Timestream version 1.0.0"
    },
  "author": {
        "name": "Amazon Timestream",
        "manifest_file_version": "1.0"
  }
}
```

元数据

元数据文件提供有关数据集的其他信息，例如列名、列类型和架构。<queryid>元数据文件可在提供的 S3 存储桶中找到，文件名为：S3: //bucket_name/ _< >_metadata.json UUID

以下是元数据文件的示例。

```
{
    "ColumnInfo": [
        {
            "Name": "hostname",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "region",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "measure_name",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "cpu_utilization",
            "Type": {
                "TimeSeriesMeasureValueColumnInfo": {
                    "Type": {
                        "ScalarType": "DOUBLE"
                    }
                }
            }
        }
  ],
  "Author": {
        "Name": "Amazon Timestream",
        "MetadataFileVersion": "1.0"
  }
}
```

元数据文件中共享的列信息与查询API响应中ColumnInfo发送的列信息的结构相同。SELECT

结果

结果文件夹包含以 Apache Parque CSV t 或格式导出的数据。

示例

当你通过 Query 提交如下UNLOAD查询时API，

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel
                    FROM sample_clickstream.sample_shopping WHERE time BETWEEN ago(2d)
 AND now())
                TO 's3://my_timestream_unloads/withoutpartition/' WITH ( format='CSV',
 compression='GZIP')
```

UNLOAD查询响应将有 1 行 * 3 列。这 3 列是：

- 行类型 BIGINT-表示导出的行数
- metadataFile of 类型 VARCHAR-这是导出的元数据文件的 S3 URI
- manifestFile of 类型 VARCHAR-这是导出的清单文件URI的 S3

您将从 Query 中得到以下响应API：

```
{
    "Rows": [
        {
            "Data": [
                {
                    "ScalarValue": "20" # No of rows in output across all files
                },
                {
                    "ScalarValue": "s3://my_timestream_unloads/withoutpartition/
AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY_<UUID>_metadata.json"
 #Metadata file
                },
                {
                    "ScalarValue": "s3://my_timestream_unloads/withoutpartition/
AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY_<UUID>_manifest.json"
 #Manifest file
                }
            ]
        }
    ],
    "ColumnInfo": [
        {
```

```
            "Name": "rows",
            "Type": {
                "ScalarType": "BIGINT"
            }
        },
        {
            "Name": "metadataFile",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "manifestFile",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        }
    ],
    "QueryId": "AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY",
    "QueryStatus": {
        "ProgressPercentage": 100.0,
        "CumulativeBytesScanned": 1000,
        "CumulativeBytesMetered": 10000000
    }
}
```

数据类型

该UNLOAD语句支持支持的数据类型除和之外所述 LiveAnalytics的 Timestream 查询语言的所有数据类型time。unknown

来UNLOAD自 Timestream 的先决条件 LiveAnalytics

以下是使用 UNLOAD Timestream for 将数据写入 S3 的 LiveAnalytics先决条件。

- 您必须有权从 Timestream 读取数据，才能在UNLOAD命令中使用 LiveAnalytics 表。
- 您的 Amazon S3 存储桶必须与您的 Timestream 位于同一 AWS 区域才能存放 LiveAnalytics资源。
- 对于选定的 S3 存储桶，请确保 S3 存储桶策略还具有允许 Timestream 导 LiveAnalytics 出数据的权限。
- 用于执行UNLOAD查询的凭证必须具有必要的 AWS 身份和访问管理 (IAM) 权限，允许 Timestream LiveAnalytics 将数据写入 S3。示例策略如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:ListMeasures",
                "timestream:WriteRecords",
                "timestream:Unload"
            ],
            "Resource": "arn:aws:timestream:<region>:<account_id>:database/
<database_name>/table/<table_name>"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
                "s3:PutObject",
                "s3:GetObjectMetadata",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3:::<S3_Bucket_Created>",
                "arn:aws:s3:::<S3_Bucket_Created>/*"
            ]
        }
    ]
}
```

有关这些 S3 写入权限的更多背景信息，请参阅 Amazon 简单存储服务指南。如果您使用KMS密钥对导出的数据进行加密，请参阅以下内容，了解所需的其他IAM策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
            "kms:DescribeKey",
            "kms:Decrypt",
            "kms:GenerateDataKey*"
        ],
        "Resource": "<account_id>-arn:aws:kms:<region>:<account_id>:key/*",
```

```
        "Condition": {
            "ForAnyValue:StringLike": {
                "kms:ResourceAliases": "alias/<Alias_For_Generated_Key>"
            }
        }
    }, {
        "Effect": "Allow",
        "Action": [
            "kms:CreateGrant"
        ],
        "Resource": "<account_id>-arn:aws:kms:<region>:<account_id>:key/*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "kms:EncryptionContextKeys": "aws:timestream:<database_name>"
            },
            "Bool": {
                "kms:GrantIsForAWSResource": true
            },
            "StringLike": {
                "kms:ViaService": "timestream.<region>.amazonaws.com"
            },
            "ForAnyValue:StringLike": {
                "kms:ResourceAliases": "alias/<Alias_For_Generated_Key>"
            }
        }
    }
]
}
```

来UNLOAD自 Timestream 的最佳实践 LiveAnalytics

以下是与该UNLOAD命令相关的最佳实践。

- 使用该UNLOAD命令可以导出到 S3 存储桶的数据量没有限制。但是，查询会在 60 分钟后超时，我们建议在单个查询中导出不超过 60GB 的数据。如果您需要导出超过 60GB 的数据，请将任务分成多个查询。
- 虽然您可以向 S3 发送数千个上传数据的请求，但建议将写入操作并行化到多个 S3 前缀。请参阅此处的文档。当多个读取器/写入器访问同一个文件夹时，S3 API 调用率可能会受到限制。
- 考虑到定义前缀的 S3 密钥长度限制，我们建议将存储桶和文件夹名称设置在 10-15 个字符以内，尤其是在使用`partitioned_by`子句时。

- 当您收到包含该UNLOAD语句的查询的 4XX 或 5XX 时，可能会将部分结果写入 S3 存储桶。的 Timestream LiveAnalytics 不会从您的存储桶中删除任何数据。在执行具有相同 S3 目标的另一个UNLOAD查询之前，我们建议手动删除由失败的查询创建的文件。您可以将失败的查询写入的文件与相应的文件进行识别`QueryExecutionId`。对于失败的查询，Timestream LiveAnalytics 不会将清单文件导出到 S3 存储桶。
- Timestream LiveAnalytics 使用分段上传将查询结果导出到 S3。当你收到来自 Timestream 的包含UNLOAD语句的查询的 4XX 或 5XX 时，Timestream for 会尽力中止多部分上传，但可能会留下一些 LiveAnalytics 不完整的部分。 LiveAnalytics 因此，我们建议按照此处的指导设置自动清理您的 S3 存储桶中未完成的分段上传。

使用CSV解析器访问CSV格式化数据的建议

- CSV解析器不允许你在分隔符、转义符和引号字符中使用相同的字符。
- 有些解CSV析器无法解释复杂的数据类型，例如数组，我们建议通过JSON反序列化器来解释这些类型。

访问 Parquet 格式的数据的建议

1. 如果您的用例需要在架构（也就是列名）中支持 UTF -8 个字符，我们建议使用 Parq uet-MR 库。
2. 结果中的时间戳以 12 字节的整数表示 () INT96
3. 时间序列将表示为`array<row<time, value>>`，其他嵌套结构将使用 Parquet 格式支持的相应数据类型

使用 partition_by 子句

- 该`partitioned_by`字段中使用的列应是选择查询中的最后一列。如果该`partitioned_by`字段中使用了多列，则这些列应是选择查询中的最后一列，并且顺序应与`partition_by`字段中使用的顺序相同。
- 用于对数据（字`partitioned_by`段）进行分区的列值只能包含ASCII字符。虽然 Timestream LiveAnalytics 允许值中包含 UTF -8 个字符，但 S3 仅支持ASCII字符作为对象键。

来UNLOAD自 Timestream 的示例用例 LiveAnalytics

假设您正在监控电子商务网站的用户会话指标、流量来源和产品购买情况。您正在使用Timestream LiveAnalytics 来获取有关用户行为、产品销售的实时见解，并对吸引客户访问网站的流量渠道（自然搜索、社交媒体、直接流量、付费广告系列等）进行营销分析。

主题

导出不带任何分区的数据

您想以CSV格式导出最近两天的数据。

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time,
query, quantity, product_id, channel
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/withoutpartition'
WITH (  format='CSV',
compression='GZIP')
```

按通道对数据进行分区

您想以CSV格式导出最近两天的数据，但希望将来自每个流量渠道的数据放在单独的文件夹中。为此，您需要使用channel列对数据进行分区，如下所示。

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time,
query, quantity, product_id, channel
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbychannel/'
WITH (
partitioned_by = ARRAY ['channel'],
```

```
format='CSV',
compression='GZIP')
```

按事件对数据进行分区

您想以CSV格式导出最近两天的数据，但希望将每个事件的数据放在单独的文件夹中。为此，您需要使用event列对数据进行分区，如下所示。

```
UNLOAD(SELECT user_id, ip_address, channel, session_id, measure_name, time,
query, quantity, product_id, event
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbyevent/'
WITH (
partitioned_by = ARRAY ['event'],
format='CSV',
compression='GZIP')
```

按通道和事件对数据进行分区

您想以CSV格式导出最近两天的数据，但希望将每个频道和频道内的数据存储在单独的文件夹中。为此，您需要使用channel和event列对数据进行分区，如下所示。

```
UNLOAD(SELECT user_id, ip_address, session_id, measure_name, time,
query, quantity, product_id, channel,event
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbychannelevent/'
WITH (
partitioned_by = ARRAY ['channel','event'],
format='CSV',
compression='GZIP')
```

清单和元数据文件

清单文件

清单文件提供有关UNLOAD执行时导出的文件列表的信息。清单文件可在提供的 S3 存储桶中找到，文件名为:S3://bucket_name/<queryid>_<UUID>_manifest.json。清单文件将包含结果文件夹中文件的 URL、相应文件的记录数和大小以及查询元数据（即为查询导出到 S3 的总字节数和总行数）。

```
{
  "result_files": [
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 32295,
                "row_count": 10
            }
    },
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 62295,
                "row_count": 20
            }
    },
  ],
  "query_metadata":
    {
      "content_length_in_bytes": 94590,
      "total_row_count": 30,
      "result_format": "CSV",
      "result_version": "Amazon Timestream version 1.0.0"
    },
  "author": {
        "name": "Amazon Timestream",
        "manifest_file_version": "1.0"
  }
}
```

元数据

元数据文件提供有关数据集的其他信息，例如列名、列类型和架构。<queryid>元数据文件可在提供的 S3 存储桶中找到，文件名为：S3: //bucket_name/ _< >_metadata.json UUID

以下是元数据文件的示例。

```
{
    "ColumnInfo": [
```

```
        {
            "Name": "hostname",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "region",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "measure_name",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "cpu_utilization",
            "Type": {
                "TimeSeriesMeasureValueColumnInfo": {
                    "Type": {
                        "ScalarType": "DOUBLE"
                    }
                }
            }
        }
  ],
  "Author": {
        "Name": "Amazon Timestream",
        "MetadataFileVersion": "1.0"
  }
}
```

元数据文件中共享的列信息与查询API响应中ColumnInfo发送的列信息的结构相同。SELECT

使用 Glue 爬虫构建 Glue 数据目录

1. 使用管理员凭据登录您的账户进行以下验证。
2. 使用此处提供的指南为 Glue 数据库创建爬虫。请注意，要在数据源中提供的 S3 文件夹应为UNLOAD结果文件夹，例如。s3://my_timestream_unloads/results

3. 按照此处的指南运行爬虫。
4. 查看 Glue 表。
   - 前往 G AWS lue → T ables。
   - 在创建爬网程序时，您将看到一个使用表格前缀创建的新表。
   - 您可以通过单击表详细信息视图来查看架构和分区信息。

以下是使用 AWS Glue 数据目录的其他 AWS 服务和开源项目。

- Amazon Athena — 有关更多信息，请参阅亚马逊 Athena 用户指南中的了解表、数据库和数据目录。
- 亚马逊 Redshift Spectrum — 有关更多信息，请参阅亚马逊 Redshift 数据库开发者指南中的使用亚马逊 Redshift Spectrum 查询外部数据。
- 亚马逊 EMR — 有关更多信息，请参阅《亚马逊EMR管理指南》中的使用基于资源的策略 AWS 让亚马逊EMR访问 Glue 数据目录。
- AWS 适用于 Apache Hive 元数据仓的 Glue 数据目录客户端 — 有关此 GitHub项目的更多信息，请参阅 Apache Hive Metastor e 的AWS Glue 数据目录客户端。

来UNLOAD自 Timestream 的限制 LiveAnalytics

以下是与该UNLOAD命令相关的限制。

- 使用该UNLOAD语句的查询的并发度为每秒 1 个查询 (QPS)。超过查询速率可能会导致限制。
- 包含UNLOAD语句的查询每次查询最多可以导出 100 个分区。我们建议在使用所选列对导出的数据进行分区之前，先检查该列的不同计数。
- 包含UNLOAD语句的查询在 60 分钟后超时。
- 该UNLOAD语句在 Amazon S3 中创建的最大文件大小为 78 GB。

有关 Timestream 的其他限制 LiveAnalytics，请参阅 配额

使用查询见解来优化 Amazon Timestream 中的查询

Query insights 是一项性能调整功能，可帮助您优化查询、提高查询性能并降低成本。借助查询见解，您可以评估查询的时间、基于时间和空间分区键的修剪效率。使用查询见解，您还可以确定需要改进的

领域，以提高查询性能。此外，借助查询见解，您可以评估您的查询使用基于时间的索引和基于分区键的索引来优化数据检索的有效性。要优化查询性能，必须对控制查询执行的时间和空间参数进行微调。

主题

查询见解的好处

以下是使用查询见解的主要好处：

- 识别效率低下的查询-Query insights 提供有关对查询访问的表进行基于时间和基于属性的修剪的信息。此信息可帮助您识别访问效果不佳的表。
- 优化数据模型和分区-您可以使用查询见解信息来访问和微调您的数据模型和分区策略。
- 调整查询 — 查询见解突显了更有效地使用索引的机会。

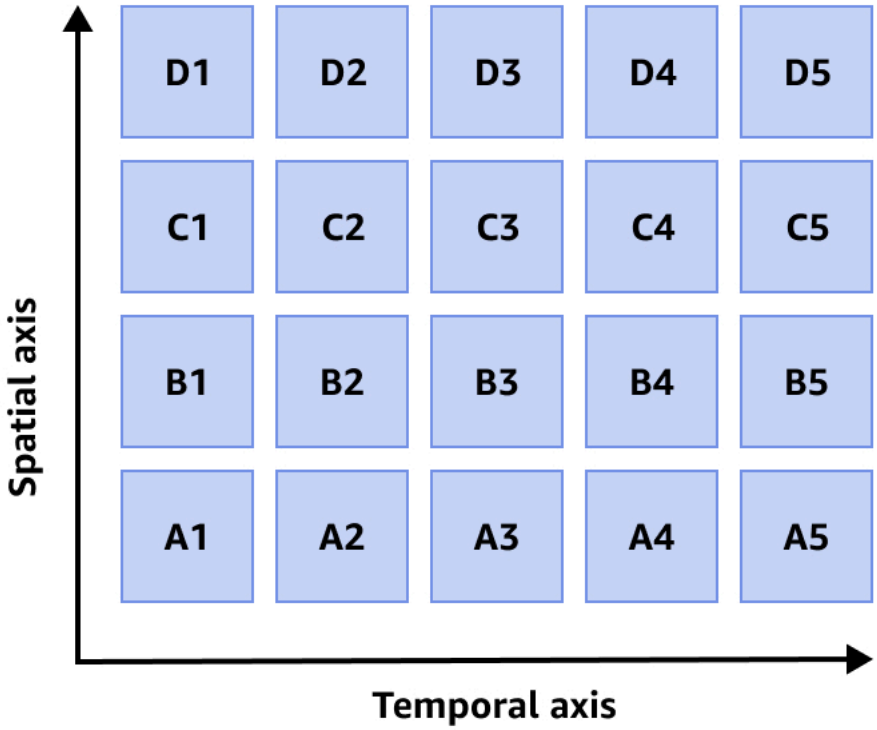
优化 Amazon Timestream 中的数据访问

您可以使用时间流分区方案或数据组织技术优化 Amazon Timestream 中的数据访问模式。

主题

时间流分区方案

Amazon Timestream 使用高度可扩展的分区方案，其中每个 Timestream 表可以有数百、数千甚至数百万个独立分区。高度可用的分区跟踪和索引服务可以管理分区，从而最大限度地减少故障的影响，并提高系统的弹性。



数据组织

Timestream 将其摄取的每个数据点存储在单个分区中。当您将数据提取到 Timestream 表时，Timestream 会根据数据中的时间戳、分区键和其他上下文属性自动创建分区。除了按时对数据进行分区（时间分区）外，Timestream 还根据选定的分区键和其他维度（空间分区）对数据进行分区。这种方法旨在分发写入流量，并允许对查询的数据进行有效修剪。

查询见解功能为查询的删减效率提供了宝贵的见解，包括查询空间覆盖范围和查询时间覆盖范围。

主题

- QuerySpatialCoverage
- QueryTemporalCoverage

QuerySpatialCoverage

该QuerySpatialCoverage指标提供了对已执行查询的空间覆盖范围的见解，以及空间修剪效率最低的表。这些信息可以帮助您确定分区策略中需要改进的领域，以增强空间修

剪。QuerySpatialCoverage指标的值介于 0 和 1 之间。指标值越低，查询在空间轴上的修剪效果越好。例如，值为 0.1 表示查询扫描空间轴的 10%。值为 1 表示查询扫描了 100% 的空间轴。

Example 使用查询见解来分析查询的空间覆盖范围

假设你有一个存储天气数据的 Timestream 数据库。假设位于美国不同州的气象站每小时记录一次温度。想象一下，您选择State作为客户定义的分区密钥 (CDPK) 来按状态对数据进行分区。

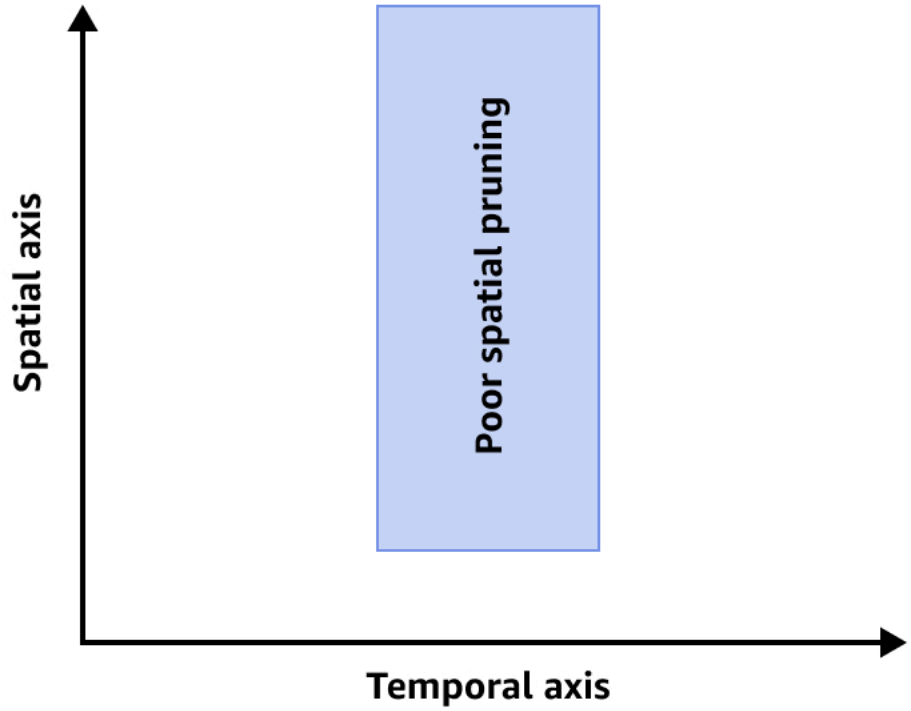
假设您执行查询以检索特定日期下午 2 点至下午 4 点之间加利福尼亚所有气象站的平均温度。以下示例显示了针对此场景的查询。

```
SELECT AVG(temperature)
FROM "weather_data"."hourly_weather"
WHERE time >= '2024-10-01 14:00:00' AND time < '2024-10-01 16:00:00'
  AND state = 'CA';
```

使用查询见解功能，您可以分析查询的空间覆盖范围。想象一下，该QuerySpatialCoverage指标返回的值为 0.02。这意味着查询仅扫描了空间轴的 2%，这很高效。在本例中，该查询能够有效地修剪空间范围，只从加利福尼亚州检索数据，而忽略来自其他州的数据。

相反，如果QuerySpatialCoverage指标返回的值为 0.8，则表示查询扫描了 80% 的空间轴，效率较低。这可能表明需要完善分区策略以改善空间修剪。例如，您可以将分区键选择为城市或区域，而不是州。通过分析QuerySpatialCoverage指标，您可以发现优化分区策略和提高查询性能的机会。

下图显示了空间修剪效果不佳。



要提高空间修剪效率，可以执行以下一项或两项操作：

- 添加measure_name、默认分区键或在查询中使用CDPK谓词。
- 如果您已经添加了前面提到的属性，请移除围绕这些属性或子句的函数，例如LIKE。

QueryTemporalCoverage

该QueryTemporalCoverage指标提供对已执行查询所扫描的时间范围的见解，包括扫描时间范围最大的表。该QueryTemporalCoverage指标的值是以纳秒表示的时间范围。该指标的值越低，查询在时间范围上的修剪越优化。例如，扫描最后几分钟数据的查询比扫描表的整个时间范围的查询性能更高。

Example

假设你有一个 Timestream 数据库，用于存储物联网传感器数据，每分钟从位于制造工厂的设备上进行一次测量。假设您已将数据分区为。device_ID

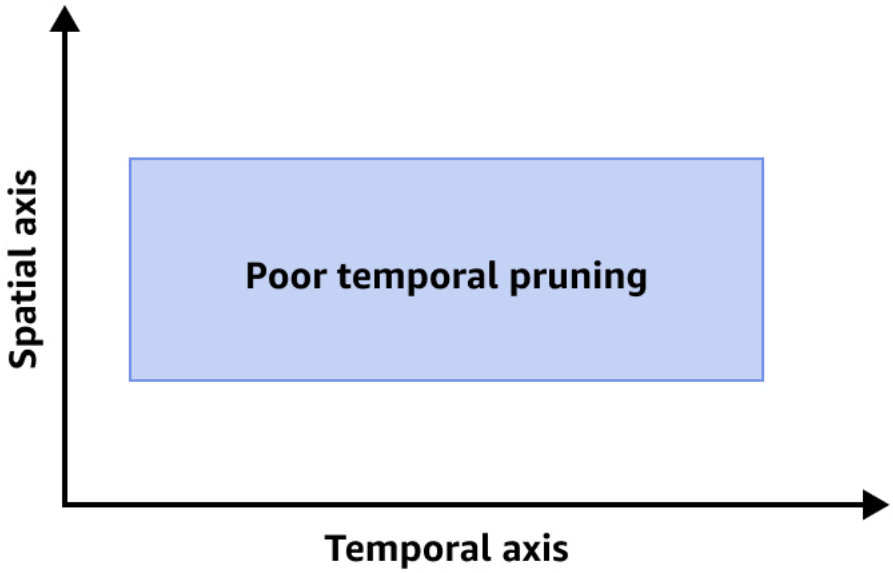
假设您执行查询以检索过去 30 分钟内特定设备的平均传感器读数。以下示例显示了针对此场景的查询。

```
SELECT AVG(sensor_reading)
FROM "sensor_data"."factory_1"
WHERE device_id = 'DEV_123'
  AND time >= NOW() - INTERVAL 30 MINUTE and time < NOW();
```

使用查询见解功能，您可以分析查询所扫描的时间范围。想象一下，该QueryTemporalCoverage指标返回的值为 1800000000000 纳秒（30 分钟）。这意味着该查询仅扫描最近 30 分钟的数据，这是一个相对狭窄的时间范围。这是一个好兆头，因为它表明查询能够有效地修剪时间分区，并且只检索了请求的数据。

相反，如果该QueryTemporalCoverage指标返回的值为 1 年（以纳秒为单位），则表示查询扫描了表中一年的时间范围，这会降低效率。这可能表明该查询未针对时间修剪进行优化，您可以通过添加时间过滤器来对其进行改进。

下图显示临时修剪效果不佳。



为了改善时间修剪，我们建议您执行以下一项或全部操作：

- 在查询中添加缺少的时间谓词，并确保时间谓词正在修剪所需的时间窗口。
- 移除函数MAX()，例如在时间谓词前后。
- 向所有子查询添加时间谓词。如果您的子查询要联接大型表或执行复杂的操作，则这一点很重要。

在 Amazon Timestream 中启用查询见解

通过直接通过查询响应提供的见解，您可以为查询启用查询见解。启用查询见解不需要额外的基础架构，也不会产生任何额外成本。启用查询见解后，除了作为查询响应一部分的查询结果外，它还会返回与查询性能相关的元数据字段。您可以使用这些信息来调整查询，以提高查询性能并降低查询成本。

有关启用查询见解的信息，请参阅运行查询。

要查看通过启用查询见解返回的响应示例，请参阅计划查询示例。

> **Note**
>
> - 启用查询见解时，它将查询速率限制为每秒 1 次查询 (QPS)。为避免影响性能，我们强烈建议您仅在查询的评估阶段启用查询见解，然后再将其部署到生产环境。
> - 查询见解中提供的见解最终是一致的，这意味着随着新数据不断被摄入表中，它们可能会发生变化。

使用查询见解响应优化查询

假设你正在使用 Amazon Timestream LiveAnalytics 来监控不同地点的能耗。想象一下，你的数据库中有两个名为`raw-metrics`和的表`aggregate-metrics`。

该`raw-metrics`表存储了设备级别的详细能耗数据，并包含以下各列：

- Timestamp
- 例如，华盛顿州
- 设备 ID
- 能源消耗

该表的数据是按 minute-by-minute粒度收集和存储的。该表用`State`作CDPK.

该`aggregate-metrics`表存储计划查询的结果，以每小时汇总所有设备的能耗数据。此表包含以下各列：

- Timestamp
- 例如，华盛顿州
- 总能耗

该aggregate-metrics表以每小时的粒度存储这些数据。该表用State作CDPK.

主题

查询最近 24 小时的能耗

假设你想提取华盛顿在过去 24 小时内消耗的总能量。要查找这些数据，您可以利用这两个表的优势：raw-metrics和aggregate-metrics。该aggregate-metrics表提供了过去 23 小时的每小时能耗数据，而该raw-metrics表提供过去 1 小时的分钟粒度数据。通过对两个表格进行查询，您可以完整而准确地了解华盛顿在过去24小时内的能源消耗情况。

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
 "metrics"."aggregate-metrics" am
 LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE rm.time >= ago(1h) and rm.time < now()
```

此示例查询仅用于说明目的，可能无法按原样运行。它旨在演示该概念，但您可能需要对其进行修改以适应您的特定用例或环境。

执行此查询后，您可能会注意到查询响应时间比预期的要慢。要确定此性能问题的根本原因，您可以使用查询见解功能来分析查询的性能并优化其执行。

以下示例显示了查询见解的响应。

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 1.0,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/raw-metrics,
                        PartitionKey: [State]
                    }
                },
                QueryTemporalRange: {
```

```
                    Max: {
                        Value:31540000000000000 //365 days,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

查询见解响应提供以下信息：

- 时间范围：查询扫描的表的时间范围超出了 365 天。aggregate-metrics这表明临时过滤的使用效率低下。
- 空间覆盖率：查询扫描了raw-metrics表格的整个空间范围 (100%)。这表明空间过滤没有得到有效利用。

如果您的查询访问了多个表，则查询见解会为访问模式最差的表提供指标。

针对时间范围优化查询

根据查询见解响应，您可以针对时间范围优化查询，如以下示例所示。

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
  "metrics"."aggregate-metrics" am
  LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE
  am.time >=  ago(23h) and am.time < now()
  AND rm.time >=  ago(1h) and rm.time < now()
  AND rm.state = 'Washington'
```

如果您再次运行该QueryInsights命令，它将返回以下响应。

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 1.0,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics,
```

```
                        PartitionKey: [State]
                    }
                },
                QueryTemporalRange: {
                    Max: {
                        Value: 82800000000000 //23 hours,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

此响应显示aggregate-metrics表格的空间覆盖率仍为 100%，效率低下。下一节说明如何优化空间覆盖率查询。

优化空间覆盖范围查询

根据查询见解响应，您可以优化查询的空间覆盖范围，如以下示例所示。

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
  "metrics"."aggregate-metrics" am
  LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE
  am.time >=  ago(23h) and am.time < now()
  AND am.state ='Washington'
  AND rm.time >=  ago(1h) and rm.time < now()
  AND rm.state = 'Washington'
```

如果您再次运行该QueryInsights命令，它将返回以下响应。

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 0.02,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics,
                        PartitionKey: [State]
                    }
                },
```

```
                QueryTemporalRange: {
                    Max: {
                        Value: 82800000000000 //23 hours,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

提高了查询性能

优化查询后，查询见解将提供以下信息：

- aggregate-metrics表格的临时修剪时间为 23 小时。这表示只扫描了时间范围的 23 小时。
- aggregate-metrics表格的空间修剪为 0.02。这表明仅扫描了表的空间范围数据的 2%。该查询只扫描了极小一部分表，从而提高了性能并降低了资源利用率。修剪效率的提高表明查询现在已针对性能进行了优化。

与... 合作 AWS Backup

Amazon Timestream 中的数据保护功能 LiveAnalytics 是一种完全托管的解决方案，可帮助您满足监管合规和业务连续性要求。该功能是通过与统一备份服务的本机集成实现的 AWS Backup，该服务旨在简化备份的创建、迁移、恢复和删除，同时提供改进的报告和审计。通过与集成 AWS Backup，您可以使用完全托管、策略驱动的集中式数据保护解决方案来创建不可变的备份，并集中管理跨越Timestream和其他AWS支持的服务的应用程序数据的数据保护。 AWS Backup

要使用该功能，您必须选择允许 AWS Backup 保护您的 Timestream 资源。选择加入选项适用于特定的账户和 AWS 区域，因此您可能需要使用同一个账户选择加入多个区域。有关 AWS Backup 的更多信息，请参阅《AWS Backup 开发人员指南》。

通过提供的数据保护功能 AWS Backup 包括以下内容。

定时备份-您可以使用备份计划为 LiveAnalytics 表设置 Timestream 的定期定时备份。

跨账户和跨区域复制-您可以自动将备份复制到不同 AWS 地区或账户中的另一个备份保管库，这样您就可以满足您的数据保护要求。

冷存储分层-您可以将备份配置为实施生命周期规则，以删除备份或将备份过渡到较冷的存储。这可以帮助您优化备份成本。

标签-您可以自动为备份添加标签，以用于计费和成本分配。

加密-您的备份数据存储在保 AWS Backup 管库中。这允许您使用独立于 Timestream LiveAnalytics 表加密 AWS KMS密钥的密钥来加密和保护备份。

使用WORM模型保护备份-您可以使用 AWS Backup Vault Lock 为备份启用 write-once-read-many (WORM) 设置。借 AWS Backup 助 Vault Lock，您可以添加额外的防御层，保护备份免受无意或恶意删除操作、备份保留期更改以及生命周期设置更新的影响。要了解更多信息，请参阅 AWS Backup 文件库锁定。

所有地区均提供数据保护功能。要了解有关该功能的更多信息，请参阅AWS Backup 开发人员指南。

备份和恢复 Timestream 表：工作原理

您可以为 Amazon Timestream 表创建备份。此部分概述了备份和还原过程中发生的情况。

主题

备份

您可以使用按需备份功能为表创建 Amazon Timestream 的完整备份。 LiveAnalytics 此部分概述了备份和还原过程中发生的情况。

您可以按表格粒度创建 Timestream 数据的备份。您可以使用 Timestream 控制台、控制台、或 AWS Backup 启动选定表的备份。SDK CLI备份是异步创建的，备份启动时间之前表中的所有数据都包含在备份中。但是，备份过程中摄入到表中的某些数据也可能包含在备份中。为了保护您的数据，您可以创建一次性按需备份，也可以安排对表进行定期备份。

在备份过程中，您无法执行以下操作。

- 暂停或取消备份操作。
- 删除备份的源表。
- 禁用表的备份 (如果正在备份该表)。

配置完成后， AWS Backup 可提供自动备份计划、保留期管理和生命周期管理，无需自定义脚本和手动流程。有关更多信息，请参阅《AWS Backup 开发人员指南》

所有用于 LiveAnalytics 备份的 Timestream 本质上都是增量备份，这意味着表的第一次备份是完整备份，而同一表的后续每一次备份都是增量备份，仅复制自上次备份以来对数据的更改。由 LiveAnalytics 于 Timestream 中的数据存储在一组分区中，因此在后续备份过程中，所有由于接收新数据或更新自上次备份以来现有数据而发生更改的分区都将在后续备份中复制。

如果您将 Timestream 用于 LiveAnalytics 控制台，则为账户中的所有资源创建的备份将列在 “备份” 选项卡中。此外，备份也列在表格详细信息中。

还原

您可以从 LiveAnalytics 控制台、控制台、或 AWS Backup Timestream 中恢复表。SDK AWS CLI您可以从备份中恢复全部数据，也可以配置表保留设置以恢复所选数据。启动还原时，可以配置下表设置。

- Database Name
- 表名称
- 内存存储保留
- 磁性存储器保留
- 启用磁存储写入
- S3 错误日志位置（可选）
- IAM恢复备份时 AWS Backup 将扮演的角色

上述配置与源表无关。要恢复备份中的所有数据，我们建议您配置新的表设置，使内存存储保留期和磁性存储保留期的总和大于最旧的时间戳与现在的时间戳之间的差值。当您选择要恢复的增量备份时，所有数据（增量+底层完整数据）都将恢复。成功恢复后，该表将处于活动状态，您可以对恢复的表执行摄取和/或查询操作。但是，在恢复过程中，您无法执行这些操作。恢复后，该表将与您账户中的任何其他表格类似。

Example 恢复备份中的所有数据

此示例具有以下假设。

最早的时间戳 — `August 1, 2021 0:00:00`

- 现在 — `November 9, 2022 0:00:00`

要从备份中恢复所有数据，请按如下方式输入并比较值。

1. 输入 “内存存储保留期” 和 “磁性存储保留期”。例如，假设这些值。
   - 内存存储保留时间 — 12 小时
   - 磁性存储保存 —500 天
2. 找出内存存储保留和磁性存储保留的总和。

```
12 hours + (500 * 24 hours) =
12 hours + 12,000 hours =
12,012 hours
```

3. 找出最旧的时间戳和现在的时间戳之间的区别。

```
November 9, 2022 0:00:00 - August 1, 2021 0:00:00 =
465 days =
465 * 24 hours =
11,160 hours
```

4. 确保第二步中的留存值之和大于第三步中的时间差。如有必要，调整保留时间。

```
12,012 > 11,160
true
```

Example 从备份中恢复所选数据

此示例具有以下假设。

- 现在 — November 9, 2022 0:00:00

要仅从备份中恢复选定的数据，请按如下方式输入并比较值。

1. 确定所需的最早时间戳。例如，假设December 4, 2021 0:00:00。
2. 找出所需的最早时间戳和现在的时间戳之间的区别。

```
November 9, 2022 0:00:00 - December 4, 2021 0:00:00 =
340 days =
340 * 24 hours =
8,160 hours
```

3. 输入所需的内存存储保留期值。例如，输入 12 小时。
4. 从第二步的差值中减去该值。

```
8,160 hours - 12 hours =
8148 hours
```

5. 为 Magnetic 存储留存率输入该值。

您可以将 LiveAnalytics 表数据的 Timestream 备份复制到其他 AWS 区域，然后在该新区域中将其恢复。您可以在 AWS 商业区域和 AWS GovCloud （美国）区域之间复制备份，然后恢复备份。只需为从源区域复制的数据以及在目标区域中还原到新表的数据付费。

恢复表后，您必须在恢复后的表上手动设置以下内容。

- AWS Identity and Access Managemen IAM t () 策略
- 标签
- 计划查询

恢复时间与表的配置直接相关。其中包括表的大小、底层分区的数量、恢复到内存存储的数据量以及其他变量。规划灾难恢复的最佳做法是定期记录平均恢复完成时间，并确定这些时间如何影响您的总体恢复时间目标（RTO）。

所有备份和还原控制台及API操作都会被捕获并记录在中， AWS CloudTrail以便记录、持续监控和审计。

创建 Amazon Timestream 表的备份

本节介绍如何为 Amazon Timestream 启用 AWS Backup 和创建按需备份和定时备份。

主题

- 启用 AWS Backup 保护数据的时间 LiveAnalytics 流
- 创建按需备份
- 计划备份

启用 AWS Backup 保护数据的时间 LiveAnalytics 流

必须启用它 AWS Backup 才能将其与 Timestream 配合使用。 LiveAnalytics

要 AWS Backup 在 LiveAnalytics 控制台的 Timestream 中启用，请执行以下步骤。

1. 登录 AWS 管理控制台。
2. 在 Timestream for LiveAnalytics 仪表板页面的顶部会显示一个弹出式横幅，用于支持数据的 Timestream。 AWS Backup LiveAnalytics 否则，请从导航窗格中选择 “备份”。
3. 在 Backup 窗口中，您将看到要启用的横幅 AWS Backup。请选择 启用。

   现在，您 AWS Backup 的 Timestream 提供了 LiveAnalytics 表格的数据保护。

要通过启用 AWS Backup，请参阅 AWS Backup 文档以通过控制台和编程方式启用。

如果您在启用 Timestream LiveAnalytics 数据保护后选择禁 AWS Backup 用 Timestream 数据保护，请通过 AWS Backup 控制台登录并将切换开关向左移动。

如果您无法启用或禁用这些 AWS Backup 功能，则您的 AWS 管理员可能需要执行这些操作。

创建按需备份

要为 LiveAnalytics 表创建 Timestream 的按需备份，请执行以下步骤。

1. 登录 AWS 管理控制台。
2. 在控制台左侧的导航窗格中，选择备份。
3. 选择创建按需备份。
4. 继续在备份窗口中选择设置。
5. 您可以立即创建备份，立即启动备份，也可以选择备份窗口开始备份。
6. 选择备份的生命周期管理策略。您可以将备份数据转移到冷存储中，在那里您必须将备份保留至少 90 天。您可以为备份设置所需的保留期。您可以选择现有保管库，也可以选择创建新的备份保管库，导航到 AWS Backup 控制台并创建新的备份保管库 <documentation link on creating a new backup vault here>
7. 选择适当的IAM角色。
8. 如果您要将一个或多个标签分配到按需备份，请输入键和可选值，然后选择添加标签。
9. 选择创建按需备份。这会将您带到 “Backup（备份）” 页面，您将在其中看到作业列表。
10为您选择备份的资源选择备份作业 ID 以查看该作业的详细信息。

计划备份

要安排备份，请参阅创建定时备份。

恢复 Amazon Timestream 表的备份

本节介绍如何恢复 Amazon Timestream 表的备份。

主题

从中恢复 LiveAnalytics 表的时间流 AWS Backup

要 AWS Backup 使用 LiveAnalytics 控制台版 Timestream 恢复 LiveAnalytics 表格的 Timestream，请按照以下步骤操作。

1. 登录 AWS 管理控制台。
2. 在控制台左侧的导航窗格中，选择备份。
3. 要恢复资源，请选择资源恢复点 ID 旁边的单选按钮。在窗格的右上角，选择还原。
4. 输入表配置设置，即数据库名称和表名。请注意，还原后的表名应与原始源表名称不同。
5. 配置内存和磁性存储器保留设置。
6. 在 “还原角色” 中，选择 AWS Backup 将担任此还原的IAM角色。
7. 选择还原备份。页面顶部的消息提供了有关还原作业的信息。

> **Note**
>
> 无论配置的内存和磁存储保留期长短，您都需要为恢复整个备份付费。但是，恢复完成后，还原后的表将仅包含配置的保留期内的数据。

将 LiveAnalytics 表格的时间流恢复到其他地区或账户

要将 LiveAnalytics 表的 Timestream 恢复到另一个区域或账户，您首先需要将备份复制到该新区域或账户。为了复制到另一个账户，该账户必须首先向您授予权限。将用于 LiveAnalytics 备份的 Timestream 复制到新区域或账户后，可以通过上一节中的流程将其恢复。

正在复制 Amazon Timestream 表的备份

您可以创建当前备份的副本。您可以按需将备份复制到多个 AWS 账户或 AWS 区域，也可以作为定时备份计划的一部分自动复制备份。如果您需要将备份存储在最接近生产数据的位置以满足业务连续性或合规性要求，则跨区域复制会特别有用。

跨账户备份可用于将备份安全地复制到组织中的一个或多个 AWS 账户，以实现运营或安全。如果原始备份被无意中删除，则可以将备份从目标账户复制到其源账户，然后开始还原。在执行此操作之前，您必须在 Organizations 服务中拥有两个属于同一组织的账户，并且必须拥有这些账户所需的权限。当您将增量备份复制到另一个账户或区域时，关联的完整备份也会被复制。

除非您另行指定，否则副本将继承源备份的配置。有一个例外。如果您将新副本指定为 “永不”，则会过期。使用此设置，新副本仍会继承其源复本的到期日期。如果您希望新备份副本是永久性的，请将源备份设置为永不过期，或者将新副本指定为在创建后 100 年过期。

要从 Timestream 控制台复制备份，请按照以下步骤操作。

1. 登录 AWS 管理控制台。
2. 在控制台左侧的导航窗格中，选择备份。
3. 选择资源恢复点 ID 旁边的单选按钮。在窗格的右上角，选择 “操作”，然后选择 “复制”。
4. 选择 “继续 AWS 备份”，然后按照跨账户备份的步骤进行操作。

LiveAnalytics 控制台版 Timestream 目前不支持跨账户和地区复制按需备份和定时备份，您必须导航 AWS Backup 到才能执行该操作。

删除备份

本节介绍如何删除 LiveAnalytics 表的 Timestream 备份。

要从 Timestream 控制台中删除备份，请按照以下步骤操作。

1. 登录 AWS 管理控制台。
2. 在控制台左侧的导航窗格中，选择备份。
3. 选择资源恢复点 ID 旁边的单选按钮。在窗格的右上角，选择 “操作”，然后选择 “删除”。
4. 选择 C ontin AWS u e to Backup，然后按照删除备份中的删除备份步骤进行操作。

> **Note**
>
> 删除增量备份时，仅删除增量备份，而不会删除底层完整备份。

配额和限制

AWS Backup 将备份限制为每个资源只能进行一次并发备份。因此，资源的其他定时或按需备份请求会被排队，并且只有在现有备份任务完成后才会启动。如果备份任务未在备份窗口内启动或完成，则请求将失败。有关 AWS Backup 限制的更多信息，请参阅《Backup 开发者指南》中的AWSAWS 备份限制。

创建备份时，每个账户最多可以同时执行四个备份。同样，您可以为每个账户执行一次并发还原。当您同时启动四个以上的备份任务时，只会启动四个备份作业，其余的任务将定期重试。启动后，如果备份作业未在配置的备份窗口持续时间内完成，则备份作业将失败。如果失败的备份任务是按需备份，则可以重试备份，对于定时备份，按以下计划尝试该作业。

客户定义的分区键

用于 LiveAnalytics 客户定义的分区键的 Amazon Timestream 是 Timestream 中的一项功能 LiveAnalytics ，它使客户能够为自己的表定义自己的分区键。分区是一种用于在多个物理存储单元之间分配数据的技术，可以更快、更高效地检索数据。使用客户定义的分区键，客户可以创建更符合其查询模式和用例的分区架构。

使用 LiveAnalytics 客户定义的分区键的 Timestream，客户可以选择一个维度名称作为其表的分区键。这样可以更灵活地为其数据定义分区架构。通过选择正确的分区键，客户可以优化其数据模型，提高查询性能并减少查询延迟。

主题

- 为现有表创建分区键
- 使用自定义复合分区键进行 LiveAnalytics 架构验证的时间流

使用客户定义的分区键

如果您的查询模式定义明确、基数维度高，并且需要低查询延迟，那么用于 LiveAnalytics 客户定义分区键的 Timestream 可以成为增强数据模型的有用工具。例如，如果您是一家跟踪您网站上的客户互动的零售公司，则主要的访问模式可能是通过客户 ID 和时间戳进行的。通过将客户 ID 定义为分区键，您的数据可以均匀分布，从而减少延迟，最终改善用户体验。

另一个例子是在医疗保健行业，可穿戴设备收集传感器数据以跟踪患者的生命体征。主要的访问模式将是通过设备 ID 和时间戳进行的，两个维度的基数都很高。通过将 Device ID 定义为分区键，可以优化查询执行并确保持续的长期查询性能。

总而言之，当您的查询模式清晰、基数维度高，并且查询需要低延迟时，用于 LiveAnalytics 客户定义的分区键的 Timestream 最有用。通过定义与您的查询模式一致的分区键，您可以优化查询执行并确保持续的长期性能查询性能。

开始使用客户定义的分区键

在控制台中，选择 “表”，然后创建一个新表。您也可以使用访问该`CreateTable`操作来创建可包含客户定义分区键的新表。SDK

使用维度类型分区键创建表

您可以使用以下代码片段创建具有维度类型分区键的表。

Java

```
public void createTableWithDimensionTypePartitionKeyExample() {
        System.out.println("Creating table");
        CreateTableRequest createTableRequest = new CreateTableRequest();
        createTableRequest.setDatabaseName(DATABASE_NAME);
        createTableRequest.setTableName(TABLE_NAME);
        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
        createTableRequest.setRetentionProperties(retentionProperties);
```

```
        // Can specify enforcement level with OPTIONAL or REQUIRED
        final List<PartitionKey> partitionKeyWithDimensionAndOptionalEnforcement =
 Collections.singletonList(new PartitionKey()
             .withName(COMPOSITE_PARTITION_KEY_DIM_NAME)
             .withType(PartitionKeyType.DIMENSION)
             .withEnforcementInRecord(PartitionKeyEnforcementLevel.OPTIONAL));
        Schema schema = new Schema();

 schema.setCompositePartitionKey(partitionKeyWithDimensionAndOptionalEnforcement);
        createTableRequest.setSchema(schema);

        try {
            writeClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
 DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Java v2

```
public void createTableWithDimensionTypePartitionKeyExample() {
        System.out.println("Creating table");
        final RetentionProperties retentionProperties =
 RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS)
                .build();
        // Can specify enforcement level with OPTIONAL or REQUIRED
        final List<PartitionKey> partitionKeyWithDimensionAndOptionalEnforcement =
 Collections.singletonList(PartitionKey
                .builder()
                .name(COMPOSITE_PARTITION_KEY_DIM_NAME)
                .type(PartitionKeyType.DIMENSION)
                .enforcementInRecord(PartitionKeyEnforcementLevel.OPTIONAL)
                .build());
        final Schema schema = Schema.builder()

 .compositePartitionKey(partitionKeyWithDimensionAndOptionalEnforcement).build();
        final CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .databaseName(DATABASE_NAME)
                .tableName(TABLE_NAME)
```

```
                .retentionProperties(retentionProperties)
                .schema(schema)
                .build();

        try {
            writeClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Go v1

```
func createTableWithDimensionTypePartitionKeyExample(){
        // Can specify enforcement level with OPTIONAL or REQUIRED
        partitionKeyWithDimensionAndOptionalEnforcement :=
 []*timestreamwrite.PartitionKey{
                {
                    Name:                aws.String(CompositePartitionKeyDimName),
                    EnforcementInRecord: aws.String("OPTIONAL"),
                    Type:                aws.String("DIMENSION"),
                },
        }
        createTableInput := &timestreamwrite.CreateTableInput{
             DatabaseName: aws.String(*databaseName),
             TableName:    aws.String(*tableName),
             // Enable MagneticStoreWrite for Table
             MagneticStoreWriteProperties:
 &timestreamwrite.MagneticStoreWriteProperties{
                 EnableMagneticStoreWrites: aws.Bool(true),
                 // Persist MagneticStoreWrite rejected records in S3
                 MagneticStoreRejectedDataLocation:
 &timestreamwrite.MagneticStoreRejectedDataLocation{
                     S3Configuration: &timestreamwrite.S3Configuration{
                         BucketName:       aws.String("timestream-sample-bucket"),
                         ObjectKeyPrefix:  aws.String("TimeStreamCustomerSampleGo"),
                         EncryptionOption: aws.String("SSE_S3"),
                     },
                 },
             },
             Schema: &timestreamwrite.Schema{
```

```
                CompositePartitionKey:
partitionKeyWithDimensionAndOptionalEnforcement,
            }
        }
        _, err := writeSvc.CreateTable(createTableInput)
    }
```

Go v2

```
 func (timestreamBuilder TimestreamBuilder)
CreateTableWithDimensionTypePartitionKeyExample() error {
        partitionKeyWithDimensionAndOptionalEnforcement := []types.PartitionKey{
            {
                Name:                aws.String(CompositePartitionKeyDimName),
                EnforcementInRecord: types.PartitionKeyEnforcementLevelOptional,
                Type:                types.PartitionKeyTypeDimension,
            },
        }
        _, err := timestreamBuilder.WriteSvc.CreateTable(context.TODO(),
&timestreamwrite.CreateTableInput{
            DatabaseName: aws.String(databaseName),
            TableName:    aws.String(tableName),
            MagneticStoreWriteProperties: &types.MagneticStoreWriteProperties{
                EnableMagneticStoreWrites: aws.Bool(true),
                // Persist MagneticStoreWrite rejected records in S3
                MagneticStoreRejectedDataLocation:
&types.MagneticStoreRejectedDataLocation{
                    S3Configuration: &types.S3Configuration{
                        BucketName:       aws.String(s3BucketName),
                        EncryptionOption: "SSE_S3",
                    },
                },
            },
            Schema: &types.Schema{
                CompositePartitionKey:
partitionKeyWithDimensionAndOptionalEnforcement,
            },
        })

        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
```

```
            fmt.Println("Create table is successful")
        }
        return err
    }
```

Python

```
def create_table_with_measure_name_type_partition_key(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': CT_TTL_DAYS
        }
        partitionKey_with_measure_name = [
            {'Type': 'MEASURE'}
        ]
        schema = {
            'CompositePartitionKey': partitionKey_with_measure_name
        }
        try:
            self.client.create_table(DatabaseName=DATABASE_NAME,
 TableName=TABLE_NAME,
                                     RetentionProperties=retention_properties,
 Schema=schema)
            print("Table [%s] successfully created." % TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                TABLE_NAME, DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

检查分区架构配置

您可以通过几种方式检查分区架构的表配置情况。在控制台中，选择 “数据库”，然后选择要检查的表。您也可以使用SDK访问该DescribeTable操作。

用分区键描述表

您可以使用以下代码片段来描述带有分区键的表。

Java

```
   public void describeTable() {
       System.out.println("Describing table");
       final DescribeTableRequest describeTableRequest = new
DescribeTableRequest();
       describeTableRequest.setDatabaseName(DATABASE_NAME);
       describeTableRequest.setTableName(TABLE_NAME);
       try {
           DescribeTableResult result =
amazonTimestreamWrite.describeTable(describeTableRequest);
           String tableId = result.getTable().getArn();
           System.out.println("Table " + TABLE_NAME + " has id " + tableId);
           // If table is created with composite partition key, it can be described
with
           //
System.out.println(result.getTable().getSchema().getCompositePartitionKey());
       } catch (final Exception e) {
           System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
           throw e;
       }
   }
```

下面是一个示例输出。

1. 表具有维度类型分区键

```
[{Type: DIMENSION,Name: hostId,EnforcementInRecord: OPTIONAL}]
```

2. 表具有度量名称类型分区键

```
[{Type: MEASURE,}]
```

3. 从未指定复合分区键创建的表中获取复合分区键

```
[{Type: MEASURE,}]
```

Java v2

```
 public void describeTable() {
       System.out.println("Describing table");
```

```
        final DescribeTableRequest describeTableRequest =
DescribeTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DescribeTableResponse response =
writeClient.describeTable(describeTableRequest);
            String tableId = response.table().arn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
            // If table is created with composite partition key, it can be described
with
            //
System.out.println(response.table().schema().compositePartitionKey());
        } catch (final Exception e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
    }
```

下面是一个示例输出。

1. 表具有维度类型分区键

```
[PartitionKey(Type=DIMENSION, Name=hostId, EnforcementInRecord=OPTIONAL)]
```

2. 表具有度量名称类型分区键

```
[PartitionKey(Type=MEASURE)]
```

3. 从未指定复合分区键创建的表中获取复合分区键将返回

```
[PartitionKey(Type=MEASURE)]
```

Go v1

```
    <tablistentry>
     <tabname> Go </tabname>
     <tabcontent>
      <programlisting language="go"></programlisting>
     </tabcontent>
    </tablistentry>
```

下面是一个示例输出。

```
{
  Table: {
    Arn: "arn:aws:timestream:us-west-2:533139590831:database/devops/table/
host_metrics_dim_pk_1",
    CreationTime: 2023-05-31 01:52:00.511 +0000 UTC,
    DatabaseName: "devops",
    LastUpdatedTime: 2023-05-31 01:52:00.511 +0000 UTC,
    MagneticStoreWriteProperties: {
      EnableMagneticStoreWrites: true,
      MagneticStoreRejectedDataLocation: {
        S3Configuration: {
          BucketName: "timestream-sample-bucket-west",
          EncryptionOption: "SSE_S3",
          ObjectKeyPrefix: "TimeStreamCustomerSampleGo"
        }
      }
    },
    RetentionProperties: {
      MagneticStoreRetentionPeriodInDays: 73000,
      MemoryStoreRetentionPeriodInHours: 6
    },
    Schema: {
      CompositePartitionKey: [{
          EnforcementInRecord: "OPTIONAL",
          Name: "hostId",
          Type: "DIMENSION"
        }]
    },
    TableName: "host_metrics_dim_pk_1",
    TableStatus: "ACTIVE"
  }
}
```

Go v2

```
 func (timestreamBuilder TimestreamBuilder) DescribeTable()
 (*timestreamwrite.DescribeTableOutput, error) {
         describeTableInput := &timestreamwrite.DescribeTableInput{
             DatabaseName: aws.String(databaseName),
             TableName:    aws.String(tableName),
         }
```

```
        describeTableOutput, err :=
 timestreamBuilder.WriteSvc.DescribeTable(context.TODO(), describeTableInput)

        if err != nil {
            fmt.Printf("Failed to describe table with Error: %s", err.Error())
        } else {
            fmt.Printf("Describe table is successful : %s\n",
 JsonMarshalIgnoreError(*describeTableOutput))
            // If table is created with composite partition key, it will be included
 in the output
        }

        return describeTableOutput, err
    }
```

下面是一个示例输出。

```
{
  "Table": {
    "Arn":"arn:aws:timestream:us-east-1:351861611069:database/cdpk-wr-db/table/
host_metrics_dim_pk",
    "CreationTime":"2023-05-31T22:36:10.66Z",
    "DatabaseName":"cdpk-wr-db",
    "LastUpdatedTime":"2023-05-31T22:36:10.66Z",
    "MagneticStoreWriteProperties":{
      "EnableMagneticStoreWrites":true,
      "MagneticStoreRejectedDataLocation":{
        "S3Configuration":{
          "BucketName":"error-configuration-sample-s3-bucket-cq8my",
          "EncryptionOption":"SSE_S3",
          "KmsKeyId":null,"ObjectKeyPrefix":null
        }
      }
    },
    "RetentionProperties":{
      "MagneticStoreRetentionPeriodInDays":73000,
      "MemoryStoreRetentionPeriodInHours":6
    },
    "Schema":{
      "CompositePartitionKey":[{
        "Type":"DIMENSION",
        "EnforcementInRecord":"OPTIONAL",
        "Name":"hostId"
```

```
      }]
    },
    "TableName":"host_metrics_dim_pk",
    "TableStatus":"ACTIVE"
  },
  "ResultMetadata":{}
}
```

Python

```
 def describe_table(self):
       print('Describing table')
       try:
           result = self.client.describe_table(DatabaseName=DATABASE_NAME,
TableName=TABLE_NAME)
           print("Table [%s] has id [%s]" % (TABLE_NAME, result['Table']['Arn']))
           # If table is created with composite partition key, it can be described
with
           # print(result['Table']['Schema'])
       except self.client.exceptions.ResourceNotFoundException:
           print("Table doesn't exist")
       except Exception as err:
           print("Describe table failed:", err)
```

下面是一个示例输出。

1. 表具有维度类型分区键

```
[{'CompositePartitionKey': [{'Type': 'DIMENSION', 'Name': 'hostId',
 'EnforcementInRecord': 'OPTIONAL'}]}]
```

2. 表具有度量名称类型分区键

```
[{'CompositePartitionKey': [{'Type': 'MEASURE'}]}]
```

3. 从未指定复合分区键创建的表中获取复合分区键

```
[{'CompositePartitionKey': [{'Type': 'MEASURE'}]}]
```

更新分区架构配置

您可以使用 with access th SDK e access `UpdateTable` 操作来更新分区架构的表配置。

使用分区键更新表

您可以使用以下代码片段使用分区键更新表。

Java

```
   public void updateTableCompositePartitionKeyEnforcement() {
       System.out.println("Updating table");

       UpdateTableRequest updateTableRequest = new UpdateTableRequest();
       updateTableRequest.setDatabaseName(DATABASE_NAME);
       updateTableRequest.setTableName(TABLE_NAME);

       // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
       final List<PartitionKey> partitionKeyWithDimensionAndRequiredEnforcement =
Collections.singletonList(new PartitionKey()
           .withName(COMPOSITE_PARTITION_KEY_DIM_NAME)
           .withType(PartitionKeyType.DIMENSION)
           .withEnforcementInRecord(PartitionKeyEnforcementLevel.REQUIRED));
       Schema schema = new Schema();

schema.setCompositePartitionKey(partitionKeyWithDimensionAndRequiredEnforcement);
       updateTableRequest.withSchema(schema);

       writeClient.updateTable(updateTableRequest);
       System.out.println("Table updated");
```

Java v2

```
   public void updateTableCompositePartitionKeyEnforcement() {
       System.out.println("Updating table");
       // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
       final List<PartitionKey> partitionKeyWithDimensionAndRequiredEnforcement =
Collections.singletonList(PartitionKey
           .builder()
           .name(COMPOSITE_PARTITION_KEY_DIM_NAME)
           .type(PartitionKeyType.DIMENSION)
```

```
            .enforcementInRecord(PartitionKeyEnforcementLevel.REQUIRED)
            .build());
        final Schema schema = Schema.builder()

.compositePartitionKey(partitionKeyWithDimensionAndRequiredEnforcement).build();
        final UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()

.databaseName(DATABASE_NAME).tableName(TABLE_NAME).schema(schema).build();

        writeClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
```

Go v1

```
// Update table partition key enforcement attribute
   updateTableInput := &timestreamwrite.UpdateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
        // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
        Schema: &timestreamwrite.Schema{
            CompositePartitionKey: []*timestreamwrite.PartitionKey{
                {
                        Name:
aws.String(CompositePartitionKeyDimName),
                        EnforcementInRecord: aws.String("REQUIRED"),
                        Type:                aws.String("DIMENSION"),
                },
            }},
    }
    updateTableOutput, err := writeSvc.UpdateTable(updateTableInput)
        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update table is successful, below is the output:")
            fmt.Println(updateTableOutput)
        }
```

Go v2

```
// Update table partition key enforcement attribute
        updateTableInput := &timestreamwrite.UpdateTableInput{
```

```
            DatabaseName: aws.String(*databaseName),
            TableName:    aws.String(*tableName),
            // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
            Schema: &types.Schema{
                CompositePartitionKey: []types.PartitionKey{
                    {
                        Name:
aws.String(CompositePartitionKeyDimName),
                        EnforcementInRecord:
types.PartitionKeyEnforcementLevelRequired,
                        Type:               types.PartitionKeyTypeDimension,
                    },
                }},
        }
        updateTableOutput, err :=
timestreamBuilder.WriteSvc.UpdateTable(context.TODO(), updateTableInput)
        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update table is successful, below is the output:")
            fmt.Println(updateTableOutput)
        }
```

Python

```
   def update_table(self):
       print('Updating table')
       try:
           # Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
           partition_key_with_dimension_and_required_enforcement = [
               {
                   'Type': 'DIMENSION',
                   'Name': COMPOSITE_PARTITION_KEY_DIM_NAME,
                   'EnforcementInRecord': 'REQUIRED'
               }
           ]
           schema = {
               'CompositePartitionKey':
partition_key_with_dimension_and_required_enforcement
           }
```

```
            self.client.update_table(DatabaseName=DATABASE_NAME,
TableName=TABLE_NAME,
                                     Schema=schema)
            print('Table updated.')
        except Exception as err:
            print('Update table failed:', err)
```

客户定义的分区键的优点

增强的查询性能：客户定义的分区键使您能够优化查询执行并提高整体查询性能。通过定义与查询模式一致的分区键，您可以最大限度地减少数据扫描并优化数据修剪，从而降低查询延迟。

更好的长期性能可预测性：客户定义的分区键允许客户在分区之间均匀分配数据，从而提高数据管理的效率。这将确保您的查询性能保持稳定，因为您的数据存储会随着时间的推移而扩展。

客户定义的分区键的局限性

作为 LiveAnalytics 用户的 Timestream，重要的是要记住客户分区密钥的限制。首先，它需要对您的工作负载和查询模式有很好的了解。这意味着您应该清楚地知道哪些维度最常用作查询中的主要筛选条件，并具有较高的基数，以便最有效地使用分区键。

其次，分区键需要在创建表时定义，不能添加到现有表中。这意味着在创建表之前，应仔细考虑分区策略，以确保其符合您的业务需求。

最后，需要注意的是，一旦创建了表，之后就无法更改分区键。这意味着在实施分区策略之前，您应该对其进行彻底的测试和评估。考虑到这些限制，Timestream的客户定义分区键可以极大地提高查询性能和长期满意度。

客户定义的分区键和低基数维度

如果您决定使用基数非常低的分区键，例如特定的区域或州，请务必注意，其他实体（例如`customerIDProductCategory`、和其他实体）的数据最终可能会分布在太多的分区中，有时数据很少或根本没有。这可能导致查询执行效率低下和性能降低。

为避免这种情况，我们建议您选择的维度不仅是密钥筛选条件的一部分，而且基数更高。这将有助于确保数据在分区之间均匀分布，并提高查询性能。

为现有表创建分区键

如果您在 Timestream 中已经有表， LiveAnalytics 并且想要使用客户定义的分区键，则需要将数据迁移到具有所需分区架构定义的新表中。这可以通过导出到 S3 和批量加载一起来完成，包括将数据从现有表导出到 S3，修改数据以包含分区键（如有必要），并在CSV文件中添加标头，然后将数据导入到定义了所需分区架构的新表中。请记住，这种方法可能既耗时又昂贵，特别是对于大型表格。

或者，您可以使用计划查询将数据迁移到具有所需分区架构的新表。此方法涉及创建一个定时查询，该查询从现有表中读取数据并写入新表。可以将计划查询设置为定期运行，直到所有数据都迁移完毕。请记住，在迁移过程中，您需要支付读取和写入数据的费用。

使用自定义复合分区键进行 LiveAnalytics 架构验证的时间流

Timestream 中的架构验证 LiveAnalytics 有助于确保输入到数据库中的数据符合指定的架构，从而最大限度地减少摄取错误并提高数据质量。特别是，当采用客户定义的分区键以优化查询性能时，架构验证特别有用。

使用客户定义的分区 LiveAnalytics 键进行架构验证的时间流是什么？

用于 LiveAnalytics 架构验证的时间流是一项功能，它可以根据预定义的架构验证输入到 LiveAnalytics 表的时间流中的数据。此架构定义了数据模型，包括要插入的记录的分区键、数据类型和约束。

使用客户定义的分区键时，架构验证变得更加重要。分区键允许您指定分区键，该分区键决定了您的数据在 Timestream 中的存储方式。 LiveAnalytics通过使用自定义分区键对照架构验证传入的数据，您可以强制执行数据一致性，尽早发现错误，并提高存储在 Timestream 中的数据的整体质量。 LiveAnalytics

如何使用 Timestream 通过自定义复合分区键进行 LiveAnalytics 架构验证

要使用 Timestream 对自定义复合分区键进行 LiveAnalytics 架构验证，请执行以下步骤：

想一想你的查询模式会是什么样子：要正确选择和定义 Timestream LiveAnalytics 表的架构，你应该从查询要求开始。

指定自定义复合分区键：创建表时，请指定自定义分区键。此键决定将用于对表数据进行分区的属性。您可以在维度键和度量键之间进行选择以进行分区。维度键根据维度名称对数据进行分区，而度量键则根据度量名称对数据进行分区。

设置强制级别：为了确保正确的数据分区以及随之而来的好处，Amazon Timestream for 允许您 LiveAnalytics 为架构中的每个分区键设置强制级别。强制级别决定了载入记录时分区键维度是必填的还是可选的。你可以在两个选项之间进行选择：`REQUIRED`，这意味着分区键必须存在于摄取的记录

中；和OPTIONAL，这意味着分区键不必存在。建议您在使用客户定义的分区时使用REQUIRED强制级别，以确保对数据进行正确分区，并充分利用此功能。此外，您可以在架构创建后随时更改强制级别配置，以适应您的数据摄取要求。

摄取数据：将数据提取到 Timestream for LiveAnalytics 表时，架构验证过程将使用自定义复合分区键对照定义的架构检查记录。如果记录不符合架构，则 Timestream LiveAnalytics 将返回验证错误。

处理验证错误：如果出现验证错误，Timestream for LiveAnalytics 将返回 a `ValidationException` 或 a`RejectedRecordsException`，具体取决于错误的类型。请务必在应用程序中处理这些异常并采取适当的措施，例如修复错误的记录并重试摄取。

更新强制级别：如有必要，您可以在创建表后使用UpdateTable操作更新分区键的强制级别。但是，需要注意的是，分区键配置的某些方面（例如名称和类型）在创建表后无法更改。如果将强制级别从更改REQUIRED为OPTIONAL，则无论是否存在选择为客户定义的分区键的属性，所有记录都将被接受。相反，如果将强制级别从OPTIONAL更改为REQUIRED，则对于不符合此条件的记录，可能会开始看到 4xx 写入错误。因此，在创建表时，必须根据数据的分区要求为您的用例选择适当的实施级别。

何时使用 Timestream 对自定义复合分区键进行 LiveAnalytics 架构验证

在数据一致性、质量和优化分区至关重要的场景中，应使用使用自定义复合分区键进行 LiveAnalytics 架构验证的时间流。通过在数据摄取期间强制使用架构，您可以防止可能导致错误分析或丢失宝贵见解的错误和不一致。

与批量加载任务的交互

在设置批量加载任务以使用客户定义的分区键将数据导入表时，有几种情况可能会影响该过程：

1. 如果将强制级别设置为OPTIONAL，则如果在配置作业期间未映射分区键，则在创建流程期间，控制台上将显示警报。使用API或时不会出现此警报CLI。
2. 如果将强制级别设置为REQUIRED，则除非将分区键映射到源数据列，否则作业创建将被拒绝。
3. 如果在创建任务REQUIRED后将强制级别更改为，则该作业将继续执行，但是任何没有正确映射分区键的记录都将被拒绝，并出现 4xx 错误。

与计划查询的交互

在设置计划查询作业以计算聚合、汇总和其他形式的预处理数据并将其存储到具有客户定义分区键的表中时，有几种情况可能会影响该过程：

1. 如果将强制级别设置为OPTIONAL，则如果在配置作业期间未映射分区键，则会显示警报。使用API或时不会出现此警报CLI。

2. 如果将强制级别设置为REQUIRED，则除非将分区键映射到源数据列，否则作业创建将被拒绝。
3. 如果在任务创建REQUIRED后将强制级别更改为，并且计划查询结果不包含分区键维度，则该作业的所有后续迭代都将失败。

向资源添加标签和标注

您可以使用标签为 LiveAnalytics 资源标记 Amazon Timestream。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。

亚马逊弹性计算云（亚马逊）、亚马逊简单存储 AWS 服务（Amazon S3EC2）、Timestream LiveAnalytics for 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. 了解标签限制。
2. 使用标记操作创建标签。

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。

添加标签限制

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：

- LiveAnalytics 表的每个 Timestream 只能有一个具有相同密钥的标签。如果您尝试添加现有标签，则现有标签值将更新为新值。
- 值在标签类别中充当描述符。在 Timestream 中 LiveAnalytics ，该值不能为空或为空。
- 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-assigned 的标签名称和值会自动分配aws:前缀，但您无法分配前缀。AWS-分配的标签名称不计入 50 的标签限制。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。

标记操作

您可以使用 LiveAnalytics控制台版 Amazon Timestream、查询语言或 AWS Command Line Interface ()AWS CLI来添加、列出、编辑或删除数据库和表的标签。

主题

- 使用控制台向新的或现有的数据库和表添加标签

使用控制台向新的或现有的数据库和表添加标签

在创建新数据库、表和计划查询时，您可以使用 LiveAnalytics 控制台的 Timestream 向它们添加标签。您还可以添加、编辑或删除现有表的标签。

在创建数据库时对其进行标记（控制台）

1. 在 https://console.aws.amazon.com/timestream 上打开 Timestream 控制台。
2. 在导航窗格中，选择数据库，然后选择创建数据库。
3. 在创建数据库页面上，提供数据库的名称。输入标签的键和值，然后选择 Add new tag (添加新标签)。
4. 选择创建数据库。

在创建表时对表进行标记（控制台）

1. 在 https://console.aws.amazon.com/timestream 上打开 Timestream 控制台。
2. 在导航窗格中，选择表，然后选择创建表。
3. 在 “为 LiveAnalytics 表创建时间流” 页面上，提供表的名称。输入标签的键和值，选择 Add new tag (添加新标签)。
4. 选择创建表。

在创建计划查询时对其进行标记（控制台）

1. 在 https://console.aws.amazon.com/timestream 上打开 Timestream 控制台。

2. 在导航窗格中，选择 “计划查询”，然后选择 “创建计划查询”。
3. 在步骤 3 上。配置查询设置页面，选择添加新标签。输入标签的键和值。要添加其他标签，请选择添加新标签。
4. 选择下一步。

标记现有资源（控制台）

1. 在 https://console.aws.amazon.com/timestream 上打开 Timestream 控制台。
2. 在导航窗格中，选择 “数据库”、“表” 或 “计划查询”。
3. 在列表中选择一个数据库或表。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 添加标签限制。

Timestream 中的安全性 LiveAnalytics

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。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Timestream的合规计划 LiveAnalytics，请参阅按合规计划划分的范围内的AWS服务。
- 云端安全-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。

本文档将帮助您了解在使用 Timestream 时如何应用分担责任模型。 LiveAnalytics以下主题向您介绍如何配置 Timestream LiveAnalytics 以满足您的安全和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 Timestream 中的 LiveAnalytics 资源。

主题

- Timestream 中的数据保护 LiveAnalytics
- 适用于 Amazon Timestream 的身份和访问管理 LiveAnalytics
- 在 Timestream 中记录和监控 LiveAnalytics

- 亚马逊 Timestream 实时分析中的弹性
- 亚马逊 Timestream 实时分析中的基础设施安全
- Timestream 中的配置和漏洞分析
- Timestream 中的事件响应 LiveAnalytics
- VPC端点 (AWS PrivateLink)
- 适用于 Amazon Timestream 的安全最佳实践 LiveAnalytics

Timestream 中的数据保护 LiveAnalytics

分担责任模式 AWS 分担责任模型适用于亚马逊 Timestream Live Analytics 中的数据保护。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础架构上的内容的控制。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FAQ。 有关欧洲数据保护的信息，请参阅责任AWS 共担模型和AWS安全GDPR博客上的博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭据并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。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 资源通信。我们需要 TLS 1.2，建议使用 TLS 1.3。
- 使用API进行设置和用户活动记录 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 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 AWS 通过命令行界面或访问时需要 FIPS 140-3 经过验证的加密模块API，请使用端点。FIPS有关可用FIPS端点的更多信息，请参阅联邦信息处理标准 (FIPS) 140-3。

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）。这包括当你使用控制台、、API或 AWS 服务 使用 Timestream Live Analytics 或其他方式 AWS CLI时。 AWS SDKs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URL向外部服务器提供，我们强烈建议您不要在中包含凭据信息，URL以验证您对该服务器的请求。

有关静态加密和密钥管理等 LiveAnalytics 数据保护主题的 Timestream 的更多详细信息，请选择以下任何可用主题。

主题

静态加密

用于静 LiveAnalytics 态加密的 Timestream 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，从而增强安全性。此功能减少保护敏感数据时涉及的操作负担和复杂性。利用静态加密，可以构建符合严格加密合规性和法规要求的安全敏感型应用程序。

- 默认情况下，Timestream 上的 LiveAnalytics 数据库加密处于开启状态，并且无法关闭。行业标准的 AES -256 加密算法是使用的默认加密算法。
- AWS KMS 是在 Timestream 中进行静态加密所必需的 LiveAnalytics。
- 无法仅加密表的项目子集。
- 您无需修改数据库客户端应用程序来使用加密。

如果您不提供密钥，则 Timestream LiveAnalytics 会创建并使用`alias/aws/timestream`在您的账户中命名的 AWS KMS 密钥。

您可以使用自己的客户管理密钥KMS来加密您的 Timestream 中的 LiveAnalytics 数据。有关 Timestream 中的密钥的更多信息 LiveAnalytics，请参阅密钥管理。

Timestream 用于将您的数据 LiveAnalytics 存储在两个存储层，即内存存储和磁性存储。内存存储数据使用 LiveAnalytics 服务密钥的时间流进行加密。磁性存储数据使用您的 AWS KMS密钥进行加密。

Timestream 查询服务需要凭据才能访问您的数据。这些凭证使用您的KMS密钥进行加密。

> **Note**
>
> Timestream f AWS KMS or LiveAnalytics 并不要求每个解密操作。取而代之的是，它在流量活跃的情况下将密钥的本地缓存保存 5 分钟。任何权限更改都将通过 LiveAnalytics 系统的 Timestream 传播，并在最多 5 分钟内实现最终一致性。

传输中加密

您的所有 Timestream 实时分析数据在传输过程中都经过加密。默认情况下，与 Timestream 的所有通信 LiveAnalytics 都使用传输层安全 (TLS) 加密进行保护。

密钥管理

您可以使用密钥管理服务 ()AWS KMS 管理亚马逊 Timestream Live Analytics 的AWS 密钥。Timestream Live Analytics 需要使用KMS来加密您的数据。根据需要对密钥的控制程度，您可以选择以下密钥管理选项：

数据库和表资源

- Timestream Live Analytics 管理的密钥：如果你不提供密钥，Timestream Live Analytics 将使用创建密钥。`alias/aws/timestream` KMS
- 客户管理的密钥：支持KMS客户管理的密钥。如果您需要更好地控制密钥的权限和生命周期，包括每年自动轮换密钥，请选择此选项。

定时查询资源

- Timestream Live Analytics 拥有的密钥：如果你不提供密钥，Timestream Live Analytics 将使用自己的KMS密钥来加密查询资源，该密钥存在于 timestream 账户中。有关更多详细信息，请参阅KMS开发者指南中的AWS 自有密钥。
- 客户管理的密钥：支持KMS客户管理的密钥。如果您需要更好地控制密钥的权限和生命周期，包括每年自动轮换密钥，请选择此选项。

KMS不支持外部密钥存储库 (XKS) 中的密钥。

适用于 Amazon Timestream 的身份和访问管理 LiveAnalytics

AWS Identity and Access Management (IAM) AWS 服务 可以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（拥有权限）使用 Timestream 获取 LiveAnalytics 资源。IAM无需支付额外费用即可使用。 AWS 服务

主题

- 受众
- 使用身份进行身份验证

- 使用策略管理访问
- 亚马逊 Timestream 的使用 LiveAnalytics 方式 IAM
- AWS 亚马逊 Timestream Live Analytics 的托管政策
- Amazon Timestream 查看 LiveAnalytics 基于身份的策略示例
- 对 Amazon Timestream 的 LiveAnalytics 身份和访问权限进行故障排除

受众

你使用 AWS Identity and Access Management (IAM) 的方式会有所不同，具体取决于你在 Timestream 中所做的工作。 LiveAnalytics

服务用户 — 如果您使用 Timestream for LiveAnalytics 服务来完成工作，则您的管理员会为您提供所需的凭据和权限。当你使用更多的 Timestream LiveAnalytics 功能来完成工作时，你可能需要额外的权限。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 Timestream 中的某项功能 LiveAnalytics，请参阅对 Amazon Timestream 的 LiveAnalytics 身份和访问权限进行故障排除。

服务管理员 — 如果你负责公司 LiveAnalytics 资源的 Timestream，那么你可能拥有对 Timestream 的完全访问权限。 LiveAnalytics你的工作是确定你的服务用户应该访问哪个 Timestream 的 LiveAnalytics 功能和资源。然后，您必须向IAM管理员提交更改服务用户权限的请求。查看此页面上的信息以了解的基本概念IAM。要详细了解贵公司如何IAM与 Timestream 配合使用 LiveAnalytics，请参阅亚马逊 Timestream 的使用 LiveAnalytics 方式 IAM。

IAM管理员 — 如果您是IAM管理员，则可能需要详细了解如何编写策略来管理 Timestream 的 LiveAnalytics访问权限。要查看可在中使用的 LiveAnalytics 基于身份的策略的 Timestream 示例，请参阅。IAM Amazon Timestream 查看 LiveAnalytics 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 AWS 账户根用户、IAM用户身份或通过担任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身份中心）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，您的管理员之前使用IAM角色设置了联合身份。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API请求AWS 签名版本 4。

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，请参阅用户指南中的多因素身份验证和AWS IAM Identity Center 用户指南IAM中的AWS 多因素身份验证。IAM

IAM 用户和组

IAM用户是您内部 AWS 账户 对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时证书，而不是创建拥有密码和访问密钥等长期凭证的IAM用户。但是，如果您有需要IAM用户长期凭证的特定用例，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》中的针对需要长期凭证的用例定期轮换访问密钥。

IAM群组是指定IAM用户集合的身份。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，您可以拥有一个名为的群组，IAMAdmins并授予该群组管理IAM资源的权限。

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，请参阅用户指南中的IAMIAM用户用例。

IAM角色

IAM角色是您内部具有特定权限 AWS 账户 的身份。它与IAM用户类似，但与特定人员无关。要在中临时扮IAM演角色 AWS Management Console，可以从用户切换到IAM角色（控制台）。您可以通过调用 AWS CLI 或 AWS API操作或使用自定义操作来代入角色URL。有关使用角色的方法的更多信息，请参阅《IAM用户指南》中的代入角色的方法。

IAM具有临时证书的角色在以下情况下很有用：

- 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。有关用于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。为了控制您的身份在进行身份验证后可以访问的内容，Ident IAM ity Center 会将权限集关联到中的IAM角色。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 用户或角色可以代入一个IAM角色，为特定任务临时获得不同的权限。
- 跨账户访问-您可以使用IAM角色允许其他账户中的某人（受信任的委托人）访问您账户中的资源。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）。要了解角色和基于资源的跨账户访问策略之间的区别，请参阅IAM用户指南IAM中的跨账户资源访问权限。

- 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，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序EC2或在 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，然后此操作在其他服务中启动另一个操作。FAS使用调用委托人的权限 AWS 服务以及 AWS 服务 向下游服务发出请求的请求。FAS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。有关提出FAS请求时的政策详情，请参阅转发访问会话。
  - 服务角色-服务IAM角色是服务代替您执行操作的角色。IAM管理员可以在内部创建、修改和删除服务角色IAM。有关更多信息，请参阅《IAM用户指南》 AWS 服务中的创建角色以向委派权限。
  - 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 Amazon 上运行的应用程序 EC2 — 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。这比在EC2实例中存储访问密钥更可取。要为EC2实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含该角色，并允许在EC2实例上运行的程序获得临时证书。有关更多信息，请参阅IAM用户指南中的使用IAM角色向在 Amazon EC2 实例上运行的应用程序授予权限。

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策略文档结构和内容的更多信息，请参阅《IAM用户指南》中的JSON策略概述。

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。要授予用户对其所需资源执行操作的权限，IAM管理员可以创建IAM策略。然后，管理员可以将IAM策略添加到角色中，用户可以代入角色。

IAM无论您使用何种方法执行操作，策略都会定义该操作的权限。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、或获取角色信息 AWS API。

基于身份的策略

基于身份的策略是可以附加到身份（例如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用户指南中的使用客户托管策略定义自定义IAM权限。

基于身份的策略可以进一步归类为内联策略或托管策略。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。要了解如何在托管策略或内联策略之间进行选择，请参阅《IAM用户指南》中的在托管策略和内联策略之间进行选择。

基于资源的策略

基于资源的JSON策略是您附加到资源的策略文档。基于资源的策略的示例包括IAM角色信任策略和 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IAM中使用 AWS 托管策略。

访问控制列表 (ACLs)

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。ACLs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 AWS WAF、和亚马逊VPC就是支持的服务示例ACLs。要了解更多信息ACLs，请参阅《亚马逊简单存储服务开发者指南》中的访问控制列表 (ACL) 概述。

其他策略类型

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-权限边界是一项高级功能，您可以在其中设置基于身份的策略可以向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 (SCPs)-SCPs 是为中的组织或组织单位 (OU) 指定最大权限的JSON策略 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。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。对成员账户中的实体（包括每个实体）的权限进行了SCP限制 AWS 账户根用户。有关 Organization SCPs s 和的更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 会话策略 – 会话策略是当您以编程方式为角色或联合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。

亚马逊 Timestream 的使用 LiveAnalytics 方式 IAM

在使用IAM管理 Timestream 的访问权限之前 LiveAnalytics，您应该了解 Timestream 可以用于哪些IAM功能。 LiveAnalytics要全面了解 Timestream LiveAnalytics 和其他 AWS 服务的使用方式IAM，请参阅《IAM用户指南》IAM中的 “与之配合使用的AWS 服务”。

主题

- LiveAnalytics基于身份的策略的时间流
- 基于 LiveAnalytics资源的策略的时间流
- 基于时间流对标签进行 LiveAnalytics 授权
- 角色的时间 LiveAnalytics IAM流

LiveAnalytics基于身份的策略的时间流

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。Timestream LiveAnalytics 支持特定的操作和资源以及条件键。要了解您在JSON策略中使用的所有元素，请参阅IAM用户指南中的IAMJSON策略元素参考。

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`Action`元素描述了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。也有一些例外，例如没有匹配API操作的仅限权限的操作。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。

您可以在IAM策略声明的 “操作” 元素中指定以下操作。使用策略授予在中执行操作的权限AWS。当您在策略中使用操作时，通常会允许或拒绝访问具有相同名称的API操作、CLISQL命令或命令。

在某些情况下，单个操作可以控制对API操作和SQL命令的访问权限。还有某些操作需要多种不同的动作。

有关支持的 Timestream LiveAnalytics `Action` 的列表，请参阅下表：

> **Note**
>
> 对于所有特定于数据库的`Actions`内容，您可以指定一个数据库，ARN将操作限制在特定的数据库上。

操作	描述	访问级别	资源类型（* 为必需）
DescribeEndpoints	返回必须向其发出后续请求的 Timestream 端点。	全部	*
Select	在 Timestream 上运行查询，从一个或多个表中选择数据。有关详细说明，请参阅此说明	读取	桌子*
CancelQuery	取消查询。	读取	*
ListTables	获取表格列表。	列出	数据库*
ListDatabases	获取数据库列表。	列出	*
ListMeasures	获取措施清单。	读取	桌子*

操作	描述	访问级别	资源类型（* 为必需）
DescribeTable	获取表格描述。	读取	桌子*
DescribeDatabase	获取数据库描述。	读取	数据库*
SelectValues	运行不需要指定特定资源的查询。有关@@ 详细说明，请参阅此说明。	读取	*
WriteRecords	将数据插入时间流。	写入	桌子*
CreateTable	创建表。	写入	数据库*
CreateDatabase	创建数据库。	写入	*
DeleteDatabase	删除数据库。	写入	*
UpdateDatabase	更新数据库。	写入	*
DeleteTable	删除表。	写入	数据库*
UpdateTable	更新表。	写入	数据库*

SelectValues 与选择：

`SelectValuesAction`是用于不需要资源的查询。不需要资源的查询示例如下：

```
SELECT 1
```

请注意，此查询并未引用特定的时间流来获取 LiveAnalytics 资源。再举一个例子：

```
SELECT now()
```

此查询使用`now()`函数返回当前时间戳，但不需要指定资源。 `SelectValues`通常用于测试，因此 Timestream for LiveAnalytics 可以在没有资源的情况下运行查询。现在，考虑一个`Select`查询：

```
SELECT * FROM database.table
```

这种类型的查询需要资源，特别是用于的 Timestream LiveAnalytics `table`，以便可以从表中获取指定的数据。

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策略元素指定要应用操作的一个或多个对象。语句必须包含 `Resource` 或 `NotResource` 元素。最佳做法是，使用资源的 Amazon 资源名称 (ARN) 来指定资源。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 Timestream 中，可以在IAM权限`Resource`元素中使用 LiveAnalytics 数据库和表。

LiveAnalytics 数据库资源的时间流具有以下内容：ARN

```
arn:${Partition}:timestream:${Region}:${Account}:database/${DatabaseName}
```

LiveAnalytics 表格资源的时间流具有以下内容：ARN

```
arn:${Partition}:timestream:${Region}:${Account}:database/${DatabaseName}/table/
${TableName}
```

有关格式的更多信息ARNs，请参阅 Amazon 资源名称 (ARNs) 和 AWS 服务命名空间。

例如，要在语句中指定`database`密钥空间，请使用以下命令：ARN

```
"Resource": "arn:aws:timestream:us-east-1:123456789012:database/mydatabase"
```

要指定属于特定账户的所有数据库，请使用通配符 (*)：

```
"Resource": "arn:aws:timestream:us-east-1:123456789012:database/*"
```

某些用于 LiveAnalytics 操作的时间流（例如创建资源的操作）无法在特定资源上执行。在这些情况下，您必须使用通配符 （*)。

```
"Resource": "*"
```

条件键

Timestream for LiveAnalytics 不提供任何特定于服务的条件密钥，但它确实支持使用一些全局条件密钥。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。

示例

要查看 LiveAnalytics 基于身份的策略的 Timestream 示例，请参阅。Amazon Timestream 查看 LiveAnalytics 基于身份的策略示例

基于 LiveAnalytics资源的策略的时间流

的 Timestream LiveAnalytics 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。

基于时间流对标签进行 LiveAnalytics 授权

您可以使用标签管理对您的 Timestream LiveAnalytics 资源的访问权限。要管理基于标签的资源访问，您可以使用 `timestream: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。有关为 LiveAnalytics 资源标记 Timestream 的更多信息，请参阅。the section called “标记资源”

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 LiveAnalytics 资源访问时间流。

角色的时间 LiveAnalytics IAM流

IAM角色是您的 AWS 账户中具有特定权限的实体。

将临时凭证与 Timestream 配合使用 LiveAnalytics

您可以使用临时证书通过联合身份登录、代入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。

服务相关角色

的 Timestream LiveAnalytics 不支持服务相关角色。

服务角色

的时间流 LiveAnalytics 不支持服务角色。

AWS 亚马逊 Timestream Live Analytics 的托管政策

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管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 托管策略可能不会为您的特定用例授予最低权限权限，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托管式策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 当新服务启动或现有服务 AWS 服务 有新API操作可用时，最有可能更新 AWS 托管策略。

有关更多信息，请参阅《IAM用户指南》中的AWS 托管策略。

主题

- AWS 托管策略： AmazonTimestreamReadOnlyAccess
- AWS 托管策略： AmazonTimestreamConsoleFullAccess
- AWS 托管策略： AmazonTimestreamFullAccess
- Timestream 实时分析对 AWS 托管策略的更新

AWS 托管策略： AmazonTimestreamReadOnlyAccess

您可以将 AmazonTimestreamReadOnlyAccess 附加到您的用户、组和角色。该政策提供对亚马逊 Timestream 的只读访问权限。

权限详细信息

此策略包含以下权限：

- Amazon Timestream— 提供对亚马逊 Timestream 的只读访问权限。此策略还授予取消任何正在运行的查询的权限。

要以JSON格式查看此政策，请参阅AmazonTimestreamReadOnlyAccess。

AWS 托管策略： AmazonTimestreamConsoleFullAccess

您可以将 AmazonTimestreamConsoleFullAccess 附加到您的用户、组和角色。

该政策提供了使用管理亚马逊 Timestream 的完全访问权限。 AWS Management Console此策略还授予某些 AWS KMS 操作和操作的权限，以管理您保存的查询。

权限详细信息

该策略包含以下权限：

- Amazon Timestream— 授予委托人访问亚马逊 Timestream 的完全访问权限。
- AWS KMS— 允许委托人列出别名和描述密钥。
- Amazon S3— 允许委托人列出所有 Amazon S3 存储桶。
- Amazon SNS— 允许委托人列出 Amazon SNS 话题。
- IAM— 允许委托人列出IAM角色。
- DBQMS – 允许主体访问、创建、删除、描述和更新查询。数据库查询元数据服务 (dbqms) 是一项仅限内部使用的服务。它为包括Amazon Timestream在内的多个 AWS 服务（包括Amazon Timestrea AWS Management Console m）上的查询编辑器提供了你最近和保存的查询。

要以JSON格式查看此政策，请参阅AmazonTimestreamConsoleFullAccess。

AWS 托管策略： AmazonTimestreamFullAccess

您可以将 AmazonTimestreamFullAccess 附加到您的用户、组和角色。

该政策提供对亚马逊 Timestream 的完全访问权限。此策略还授予某些 AWS KMS 操作的权限。

权限详细信息

该策略包含以下权限：

- Amazon Timestream— 授予委托人访问亚马逊 Timestream 的完全访问权限。
- AWS KMS— 允许委托人列出别名和描述密钥。
- Amazon S3— 允许委托人列出所有 Amazon S3 存储桶。

要以JSON格式查看此政策，请参阅AmazonTimestreamFullAccess。

Timestream 实时分析对 AWS 托管策略的更新

查看有关Timestream Live Analytics AWS 托管策略自该服务开始跟踪这些更改以来这些更新的详细信息。要获得有关此页面变更的自动提醒，请订阅 Tim RSS estream Live Analytics 文档历史记录页面上的提要。

更改	描述	日期
AmazonTimestreamReadOnlyAccess – 对现有策略的更新	已将timestream:DescribeAccountSettings 操作添加到现有AmazonTimestreamReadOnlyAccess 托管策略中。此操作用于描述 AWS 账户 设置。 Timestream Live Analytics 还通过添加一个Sid字段来更新此托管政策。 策略更新不会影响AmazonTimestreamReadOnlyAccess 托管策略的使用。	2024年6月3日
AmazonTimestreamReadOnlyAccess – 更新到现有策略	在现有AmazonTimestreamReadOnlyAccess 托管策略中添加了timestream:DescribeBatchLoadTask 和timestream:ListBatchLoadTasks 操作。这些操作用于列出和描述批量加载任务。 策略更新不会影响AmazonTimestreamReadOnlyAccess 托管策略的使用。	2023 年 2 月 24 日
AmazonTimestreamReadOnlyAccess – 更新到现有策略	在现有AmazonTimestreamReadOnlyAccess 托管策略中添	2021 年 11 月 29 日

更改	描述	日期
	加了timestrea m:DescribeSchedule dQuery 和timestrea m:ListScheduledQue ries 操作。这些操作用于列出和描述现有的计划查询。 策略更新不会影响AmazonTim estreamReadOnlyAcc ess 托管策略的使用。	
AmazonTimestreamCo nsoleFullAccess – 更新到现有策略	已将s3:ListAllMyBucket s 操作添加到现有AmazonTim estreamConsoleFull Access 托管策略中。当您为 Timestream 指定一个 Amazon S3 存储桶以记录磁性存储写入错误时，将使用此操作。 策略更新不会影响AmazonTim estreamConsoleFull Access 托管策略的使用。	2021 年 11 月 29 日
AmazonTimestreamFu llAccess – 更新到现有策略	已将s3:ListAllMyBucket s 操作添加到现有AmazonTim estreamFullAccess 托管策略中。当您为 Timestream 指定一个 Amazon S3 存储桶以记录磁性存储写入错误时，将使用此操作。 策略更新不会影响AmazonTim estreamFullAccess 托管策略的使用。	2021 年 11 月 29 日

更改	描述	日期
AmazonTimestreamConsoleFullAccess – 更新到现有策略	从现有AmazonTimestreamConsoleFullAccess 托管策略中删除了多余的操作。以前，该策略包括一项冗余操作dbqms:DescribeQueryHistory 。更新的策略删除了多余的操作。 策略更新不会影响AmazonTimestreamConsoleFullAccess 托管策略的使用。	2021 年 4 月 23 日
Timestream 实时分析开始跟踪更改	Timestream Live Analytics 开始跟踪其 AWS 托管策略的变化。	2021 年 4 月 21 日

Amazon Timestream 查看 LiveAnalytics 基于身份的策略示例

默认情况下，IAM用户和角色无权为 LiveAnalytics 资源创建或修改 Timestream。他们也无法使用 AWS Management Console、CQLSH AWS CLI、或执行任务 AWS API。IAM管理员必须创建IAM策略，授予用户和角色对其所需的指定资源执行特定API操作的权限。然后，管理员必须将这些策略附加到需要这些权限的IAM用户或群组。

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》JSON中的“在选项卡上创建策略”。

主题

- 策略最佳实践
- 使用控制台的 Timestrea LiveAnalytics m
- 允许用户查看他们自己的权限
- Timestream 中的常见操作 LiveAnalytics
- 基于标签的 LiveAnalytics 资源访问时间流
- 计划查询

策略最佳实践

基于身份的策略决定了某人是否可以创建、访问或删除您账户中的 LiveAnalytics 资源的 Timestream。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，请遵循以下指南和建议：

- 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AWS 管策略或工作职能托管策略。
- 应用最低权限权限-使用IAM策略设置权限时，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。有关使用应用权限IAM的更多信息，请参阅《IAM用户指南》IAM中的策略和权限。
- 使用IAM策略中的条件进一步限制访问权限-您可以在策略中添加条件以限制对操作和资源的访问权限。例如，您可以编写一个策略条件来指定所有请求都必须使用发送SSL。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IAMJSON策略元素：条件。
- 使用 A IAM ccess Analyzer 验证您的IAM策略以确保权限的安全性和功能性 — A IAM ccess Analyzer 会验证新的和现有的策略，以便策略符合IAM策略语言 (JSON) 和IAM最佳实践。IAMAccess Analyzer 提供了 100 多项策略检查和可行的建议，可帮助您制定安全和实用的策略。有关更多信息，请参阅《IAM用户指南》中的使用 A IAM ccess Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果您的场景需要IAM用户或 root 用户 AWS 账户，请打开MFA以提高安全性。要要求MFA何时调用API操作，请在策略中添加MFA条件。有关更多信息，请参阅《IAM用户指南》MFA中的使用进行安全API访问。

有关中最佳做法的更多信息IAM，请参阅《IAM用户指南》IAM中的安全最佳实践。

使用控制台的 Timestrea LiveAnalytics m

Timestream LiveAnalytics 版不需要特定权限即可访问适用于控制台的 Amazon Timestream LiveAnalytics 。您至少需要具有只读权限才能列出和查看有关您 AWS 账户中 LiveAnalytics 资源的时间流的详细信息。如果您创建的基于身份的策略比所需的最低权限更严格，则控制台将无法按预期运行，适用于使用该策略的实体（IAM用户或角色）。

允许用户查看他们自己的权限

此示例说明如何创建允许IAM用户查看附加到其用户身份的内联和托管策略的策略。此策略包括在控制台上或使用或以编程方式完成此操作的 AWS CLI 权限。 AWS API

```
{
    "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": "*"
        }
    ]
}
```

Timestream 中的常见操作 LiveAnalytics

以下是允许在 Timestream 中进行常见操作的示例IAM策略 LiveAnalytics。

主题

- 允许所有操作
- 允许SELECT操作
- 允许对多个资源进行SELECT操作

- 允许元数据操作
- 允许INSERT操作
- 允许CRUD操作
- 取消查询并在不指定资源的情况下选择数据
- 创建、描述、删除和描述数据库
- 按标签限制列出的数据库 {"Owner": "${username}"}
- 列出数据库中的所有表
- 在表格上创建、描述、删除、更新和选择
- 按表限制查询

允许所有操作

以下是允许在 Timestream 中进行所有操作的示例策略。 LiveAnalytics

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:*"
            ],
            "Resource": "*"
        }
    ]
}
```

允许SELECT操作

以下示例策略允许对特定资源进行样SELECT式查询。

> **Note**
>
> <account_ID>用您的亚马逊账户编号替换。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:DescribeTable",
                "timestream:ListMeasures"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        },
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

允许对多个资源进行SELECT操作

以下示例策略允许对多个资源进行样SELECT式查询。

> **Note**
>
> <account_ID>用您的亚马逊账户编号替换。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:DescribeTable",
                "timestream:ListMeasures"
            ],
            "Resource": [
```

```
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps",
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps1",
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

允许元数据操作

以下示例策略允许用户执行元数据查询，但不允许用户执行在 Timestream 中读取或写入实际数据的操作。 LiveAnalytics

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:DescribeTable",
                "timestream:ListMeasures",
                "timestream:SelectValues",
                "timestream:ListTables",
                "timestream:ListDatabases",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

允许INSERT操作

以下示例策略允许用户对账户database/sampleDB/table/DevOps内执行INSERT操作<account_id>。

> **Note**
>
> <account_ID>用您的亚马逊账户编号替换。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "timestream:WriteRecords"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:<account_id>:database/sampleDB/table/
DevOps"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

允许CRUD操作

以下示例策略允许用户在 Timestream 中为 LiveAnalytics执行CRUD操作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:CreateTable",
                "timestream:DescribeTable",
                "timestream:CreateDatabase",
                "timestream:DescribeDatabase",
                "timestream:ListTables",
                "timestream:ListDatabases",
                "timestream:DeleteTable",
                "timestream:DeleteDatabase",
                "timestream:UpdateTable",
                "timestream:UpdateDatabase"
            ],
            "Resource": "*"
        }
    ]
}
```

取消查询并在不指定资源的情况下选择数据

以下示例策略允许用户取消查询并对不需要指定资源的数据执行Select查询：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

创建、描述、删除和描述数据库

以下示例策略允许用户创建、描述、删除和描述数据库sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "timestream:CreateDatabase",
                "timestream:DescribeDatabase",
                "timestream:DeleteDatabase",
                "timestream:UpdateDatabase"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB"
        }
    ]
}
```

按标签限制列出的数据库 **{"Owner": "${username}"}**

以下示例策略允许用户列出所有使用键值对标记的数据库{"Owner": "${username}"}：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:ListDatabases"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

列出数据库中的所有表

以下示例策略用于列出数据库中的所有表sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "timestream:ListTables"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/"
        }
    ]
}
```

在表格上创建、描述、删除、更新和选择

以下示例策略允许用户创建表、描述表、删除表、更新表以及对数据库DevOps中的表执行Select查询sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:CreateTable",
                "timestream:DescribeTable",
                "timestream:DeleteTable",
                "timestream:UpdateTable",
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        }
    ]
}
```

按表限制查询

以下示例策略允许用户查询除DevOps数据库之外的所有表sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select"
```

```
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        }
    ]
}
```

基于标签的 LiveAnalytics 资源访问时间流

您可以使用基于身份的策略中的条件来控制基于标签的 LiveAnalytics 资源对 Timestream 的访问权限。本章节提供了一些示例。

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）。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAccessTaggedTables",
            "Effect": "Allow",
            "Action": "timestream:Select",
            "Resource": "arn:aws:timestream:us-east-2:111122223333:database/mydatabase/
table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

您可以将此政策附加到您账户中的IAM用户。如果名为的用户`richard-roe`尝试查看 LiveAnalytics 表的时间流，则必须对该表进行标记`Owner=richard-roe`或`owner=richard-roe`。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 “IAMJSON策略元素：条件”。

如果请求中传递的标签具有键Owner和值，则以下策略向用户授予创建带有标签的表的权限username：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTagTableUser",
            "Effect": "Allow",
            "Action": [
                "timestream:Create",
                "timestream:TagResource"
            ],
            "Resource": "arn:aws:timestream:us-east-2:111122223333:database/mydatabase/
table/*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:RequestTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

以下策略允许DescribeDatabaseAPI在任何将env标签设置为dev或的数据库上使用test：

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDescribeEndpoints",
      "Effect": "Allow",
      "Action": [
        "timestream:DescribeEndpoints"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowDevTestAccess",
      "Effect": "Allow",
      "Action": [
```

```
        "timestream:DescribeDatabase"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "timestream:tag/env": [
              "dev",
              "test"
          ]
        }
      }
    }
  ]
}
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagAccessForDevResources",
      "Effect": "Allow",
      "Action": [
        "timestream:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": [
            "test",
            "dev"
          ]
        }
      }
    }
  ]
}
```

此策略使用Condition密钥来允许将密钥env且值为testqa、或dev的标签添加到资源中。

计划查询

列出、删除、更新、执行 ScheduledQuery

以下示例策略允许用户列出、删除、更新和执行计划查询。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DeleteScheduledQuery",
                "timestream:ExecuteScheduledQuery",
                "timestream:UpdateScheduledQuery",
                "timestream:ListScheduledQueries",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*"
        }
    ]
}
```

CreateScheduledQuery 使用客户管理的KMS密钥

以下示例策略允许用户创建使用客户托管KMS密钥加密的计划查询；*<keyid for ScheduledQuery>*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/ScheduledQueryExecutionRole"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:CreateScheduledQuery",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
```

```
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        }
    ]
}
```

DescribeScheduledQuery 使用客户管理的KMS密钥

以下示例策略允许用户描述使用客户托管KMS密钥创建的计划查询；*<keyid for ScheduledQuery>*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "timestream:DescribeScheduledQuery",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        }
    ]
}
```

执行角色权限（使用客户托管KMS密钥进行计划查询，SSE-KMS 用于错误报告）

将以下示例策略附加到ScheduledQueryExecutionRoleArn参数中指定的IAM角色，该角色使用客户托管KMS密钥进行计划查询SSE-KMS加密和错误报告加密。CreateScheduledQuery API

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "kms:GenerateDataKey",
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for database-1>",
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for database-n>",
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for ScheduledQuery>"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-west-2:123456789012:scheduled-query-notification-topic-
*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:Select",
                "timestream:SelectValues",
                "timestream:WriteRecords"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject",
                "s3:GetBucketAcl"
```

```
            ],
            "Resource": [
                "arn:aws:s3:::scheduled-query-error-bucket",
                "arn:aws:s3:::scheduled-query-error-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

执行角色信任关系

以下是在的ScheduledQueryExecutionRoleArn参数中指定的IAM角色的信任关系CreateScheduledQueryAPI。

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

允许访问在账户中创建的所有计划查询

将以下示例策略附加到ScheduledQueryExecutionRoleArn参数中指定的IAM角色CreateScheduledQueryAPI，以允许访问在账户中创建的所有计划查询 *Account_ID*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "Account_ID"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:timestream:us-
west-2:Account_ID:scheduled-query/*"
                }
            }
        }
    ]
}
```

允许访问所有具有特定名称的计划查询

将以下示例策略附加到ScheduledQueryExecutionRoleArn参数中指定的IAM角色CreateScheduledQueryAPI，以允许访问名称以开头的所有定时查询 *Scheduled_Query_Name*，在账户内 *Account_ID*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "Account_ID"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:timestream:us-
west-2:Account_ID:scheduled-query/Scheduled_Query_Name*"
                }
            }
        }
    ]
}
```

对 Amazon Timestream 的 LiveAnalytics 身份和访问权限进行故障排除

使用以下信息来帮助您诊断和修复在使用和的 Timestream 时可能遇到的 LiveAnalytics 常见问题。IAM

主题

- 我无权在 Timestream 中执行以下操作 LiveAnalytics
- 我无权执行 iam：PassRole
- 我想允许 AWS 账户之外的人访问我的 Timestream 获取资源 LiveAnalytics

我无权在 Timestream 中执行以下操作 LiveAnalytics

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。管理员是向您提供登录凭证的人。

当`mateojackson`IAM用户尝试使用控制台查看有关某项的详细信息时，会出现以下示例错误 *`table`* 但没有该表的`timestream:`*`Select`*权限。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 timestream: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 `timestream:`*`Select`* 操作访问 *`mytable`* 资源。

我无权执行 iam：PassRole

如果您收到错误消息，提示您无权执行`iam:PassRole`操作，则必须更新您的策略以允许您将角色传递给 Timestream。 LiveAnalytics

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。

当名为的IAM用户`marymajor`尝试使用控制台在 Timestream 中执行操作时，会发生以下示例错误。LiveAnalytics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。

我想允许 AWS 账户之外的人访问我的 Timestream 获取资源 LiveAnalytics

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 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解 Timestream 版是否 LiveAnalytics 支持这些功能，请参阅亚马逊 Timestream 的使用 LiveAnalytics 方式 IAM。
- 要了解如何提供对您拥有的资源的访问权限，请参阅《IAM用户指南》中的 AWS 账户 向其他IAM用户提供访问权限。 AWS 账户
- 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《用户指南》中的向经过外部身份验证的用户提供访问权限（联合身份验证）。IAM
- 要了解使用角色和基于资源的策略进行跨账户访问的区别，请参阅IAM用户指南IAM中的跨账户资源访问权限。

在 Timestream 中记录和监控 LiveAnalytics

监控是维护您的 AWS 解决方案的 Timestream 的可靠性、可用性和性能的重要组成部分。LiveAnalytics 您应该从 AWS 解决方案的所有部分收集监控数据，以便在出现多点故障时可以更轻松地进行调试。但是，在开始监控 Timestream 之前 LiveAnalytics，您应该创建一个包含以下问题的答案的监控计划：

- 监控目的是什么？
- 您将监控哪些资源？
- 监控这些资源的频率如何？
- 您将使用哪些监控工具？
- 谁负责执行监控任务？
- 出现错误时应通知谁？

下一步是通过测量不同时间和不同负载条件下的 LiveAnalytics 性能，为环境中的正常 Timestream 性能建立基准。在监控 Timestream 时 LiveAnalytics，请存储历史监控数据，以便您可以将其与当前性能数据进行比较，识别正常的性能模式和性能异常，并设计解决问题的方法。

要建立基准，您至少应监控以下各项：

- 系统错误，以便您可以确定是否有任何请求导致了错误。

主题

监控工具

AWS 提供了可用于监控 Timestream 的各种 LiveAnalytics工具。您可以配置其中的一些工具来为您执行监控任务，但有些工具需要手动干预。建议您尽可能实现监控任务自动化。

主题

自动监控工具

您可以使用以下自动监控工具来监视 Timestream， LiveAnalytics 并在出现问题时进行报告：

- A CloudWatch mazon Alarms — 在您指定的时间段内观察单个指标，并根据该指标在多个时间段内相对于给定阈值的值执行一项或多项操作。该操作是向亚马逊简单通知服务 (亚马逊SNS) 主题或 Amazon A EC2 uto Scaling 策略发送的通知。 CloudWatch 警报不会仅仅因为它们处于特定状态就调用操作；该状态必须已更改并保持了指定的时间段。有关更多信息，请参阅 使用 Amazon 进行监控 CloudWatch。

手动监控工具

监控 Timestream 的 LiveAnalytics 另一个重要部分是手动监控 CloudWatch 警报未涵盖的项目。 LiveAnalytics、 CloudWatch Trusted Advisor、和其他 AWS Management Console 仪表板的 Timestream 提供了 AWS 环境状态的 at-a-glance视图。

- CloudWatch 主页显示以下内容：
  - 当前告警和状态
  - 告警和资源图表
  - 服务运行状况

  此外，您还可以使用 CloudWatch 执行以下操作：
  - 创建自定义控制面板以监控您关心的服务
  - 绘制指标数据图，以排除问题并弄清楚趋势
  - 搜索和浏览您的所有 AWS 资源指标
  - 创建和编辑告警以接收问题通知

使用记录 LiveAnalytics API通话的时间流 AWS CloudTrail

Timestream f LiveAnalytics o AWS CloudTrail r 与一项服务集成，该服务提供用户、角色或 AWS 服务在 Timestream 中采取的操作的记录。 LiveAnalytics CloudTrail 将 Timestream 的数据定义语言 (DDL) API 调用捕获 LiveAnalytics 为事件。捕获的调用包括来自 Timestream 的 LiveAnalytics 控制台调用和对 Timestream 进行 LiveAnalytics API操作的代码调用。如果您创建跟踪，则可以允许将 CloudTrail事件持续传输到亚马逊简单存储服务 (Amazon S3) Service 存储桶，包括 Timestream 的事件。 LiveAnalytics如果您未配置跟踪，您仍然可以在 CloudTrail 主机上的事件历史记录中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向 Timestream 发出的请求 LiveAnalytics、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅AWS CloudTrail 用户指南。

LiveAnalytics 信息的时间流 CloudTrail

CloudTrail 在您创建 AWS 账户时已在您的账户上启用。在 Timestream 中发生活动时 LiveAnalytics，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。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，请参阅使用事件历史记录查看 CloudTrail 事件。

> **Warning**
>
> 目前，Timestream 为所有管理和`Query`API操作 LiveAnalytics 生成 CloudTrail 事件，但不为`WriteRecords`和`DescribeEndpoints`APIs生成事件。

要持续记录您的 AWS 账户中的事件，包括 Timestream 的事件 LiveAnalytics，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，当您在控制台中创建跟踪时，该跟踪将应用于所有AWS区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。

有关更多信息，请参阅《AWS CloudTrail 用户指南》中的以下主题：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 为以下各项配置亚马逊SNS通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件
- 从多个账户接收 CloudTrail 日志文件
- 记录数据事件

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用 root 还是 AWS Identity and Access Management (IAM) 用户凭证发出
- 请求是使用角色还是联合用户的临时安全凭证发出的
- 请求是否由其他 AWS 服务发出

有关更多信息，请参阅CloudTrail userIdentity元素。

对于QueryAPI活动：

- 创建可接收所有事件的跟踪，或者使用 Timestream 选择事件作为 LiveAnalytics 资源类型`AWS::Timestream::Database`或`AWS::Timestream::Table`。
- QueryAPI不访问任何数据库或表或者由于查询字符串格式错误而导致验证异常的请求记录在中，CloudTrail 其资源类型`AWS::Timestream::Database`和ARN值为：

  ```
  arn:aws:timestream:(region):(accountId):database/NO_RESOURCE_ACCESSED
  ```

  这些事件仅传递给接收资源类型的事件的跟踪`AWS::Timestream::Database`。

亚马逊 Timestream 实时分析中的弹性

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。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失效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。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

有关通过提供的 Timestream 数据保护功能的信息 AWS Backup，请参阅与... 合作 AWS Backup。

亚马逊 Timestream 实时分析中的基础设施安全

作为一项托管服务，Amazon Timestream Live Analytics 受亚马逊网络服务：安全流程概述白皮书中描述的 AWS 全球网络安全程序的保护。

您可以使用 AWS 已发布的API呼叫通过网络访问 Timestream Live Analytics。客户端必须支持传输层安全 (TLS) 1.0 或更高版本。我们建议使用 TLS 1.2 或更高版本。客户端还必须支持具有完全向前保密性的密码套件 ()，例如 Ephemeral Diffie-Hellman (PFS) 或 Elliptic Curve Ephemeral Diffie-Hellman ()。DHE ECDHE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委托人关联的私有访问密钥对请求进行签名。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。

Timestream Live Analytics 的架构使您的流量与您的 Timestream Live Analytics 实例所在的特定 AWS 区域隔离开来。

Timestream 中的配置和漏洞分析

配置和 IT 控制由您（我们的客户）共同 AWS 负责。有关更多信息，请参阅责任 AWS 共担模型。除了责任共担模式外，Timestream LiveAnalytics 用户还应注意以下几点：

- 客户有责任使用相关的客户端依赖项修补其客户端应用程序。
- 如果合适，客户应考虑进行渗透测试（参见https://aws.amazon.com/security/渗透测试/。）

Timestream 中的事件响应 LiveAnalytics

Amazon Timestream 的 LiveAnalytics 服务事件是在个人健康控制面板中报告的。您可以在此处了解有关仪表板的 AWS Health 更多信息。

timestream LiveAnalytics 支持使用进行 AWS CloudTrail报告。有关更多信息，请参阅 使用记录 LiveAnalytics API通话的时间流 AWS CloudTrail。

VPC端点 (AWS PrivateLink)

您可以通过创建接口VPC终端节点在您VPC和 Amazon Timestream 之间建立私有连接。 LiveAnalytics 接口端点由一项技术提供支持 AWS PrivateLink，该技术使您 LiveAnalytics APIs无需互联网网关、NAT设备、连接或 Direct Connect VPN 连接即可私密访问 Timest AWS ream。您中的实例VPC不需要公有 IP 地址即可与 Timestream 通信。 LiveAnalytics APIs您VPC和 Timestream 之间的流量 LiveAnalytics不会离开亚马逊网络。

每个接口端点均由子网中的一个或多个弹性网络接口表示。有关接口VPC终端节点的更多信息，请参阅 Amazon VPC 用户指南中的接口VPC终端节点 (AWS PrivateLink)。

为了开始使用适用于 LiveAnalytics 和VPC端点的 Timestream，我们提供了有关使用端点的 Timestream 的具体注意事项、为 Timestream 创建接口VPC端点、为其创建VPC终端节点策略以及将 Timestream 客户端（用于写入或查询） LiveAnalytics 与VPC端点一起使用 Timestream 客户端（用于写入或查询）的具体注意事项的信息。 LiveAnalytics LiveAnalytics SDK VPC

主题

- VPC端点如何与 Timestream 协作
- 为 Timestream 创建接口VPC端点 LiveAnalytics
- 为 Timestream 创建VPC终端节点策略 LiveAnalytics

VPC端点如何与 Timestream 协作

当您创建用于访问 Timestream Write 或 Timestream 查询的终VPC端节点时SDK，所有请求都将路由到亚马逊网络内的终端节点，并且不会访问公共互联网。更具体地说，您的请求将路由到您的账户已映射到给定区域的单元格的写入和查询终端节点。要了解有关 Timestream 的蜂窝架构和蜂窝特定端点的更多信息，可以参考。蜂窝架构例如，假设您的账户已映射到cell1中us-west-2，并且您已经为写入 (ingest-cell1.timestream.us-west-2.amazonaws.com) 和查询 (query-cell1.timestream.us-west-2.amazonaws.com) 设置了VPC接口终端节点。在这种情况下，使用这些终端节点发送的任何写入请求都将完全保留在 Amazon 网络中，并且无法访问公共互联网。

Timestream 端点VPC注意事项

在为 Timestream 创建VPC端点时，请考虑以下几点：

- 在为 Timestream 设置接口VPC终端节点之前 LiveAnalytics，请务必查看亚马逊VPC用户指南中的接口终端节点属性和限制。
- Timestream for LiveAnalytics 支持从您VPC调用其所有API操作。
- VPCTimestream 支持终端节点策略。 LiveAnalytics默认情况下，允许通过终端节点对 Timestream 进行 LiveAnalytics 完全访问权限。有关更多信息，请参阅 Amazon VPC 用户指南中的使用VPC终端节点控制对服务的访问。
- 由于 Timestream 的架构，访问写入和查询操作需要创建两个VPC接口端点，每个SDK接口端点对应一个。此外，您必须指定单元终端节点（您只能为映射到的 Timestream 单元格创建端点）。详细信息可以在本指南的 “为 Timestream 创建接口VPC端点 LiveAnalytics” 部分中找到。

既然你已经了解了 Timestream 版是如何 LiveAnalytics 与VPC端点配合使用的，那就为其创建一个 Timestream 的接口VPC端点。 LiveAnalytics

为 Timestream 创建接口VPC端点 LiveAnalytics

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 LiveAnalytics 服务的 Timestream 创建接口VPC终端节点。要为 Timestream 创建VPC终端节点，请完成下述特定于时间流的步骤。

> **Note**
>
> 在完成以下步骤之前，请确保您了解 Timestream VPC 端点的具体注意事项。

使用 Timestream 单元构建VPC终端节点服务名称

由于 Timestream 的架构独特，因此必须为每个VPC接口端点SDK（写入和查询）创建单独的接口端点。此外，您必须指定 Timestream 单元格端点（您只能为映射到的 Timestream 单元格创建端点）。要使用接口VPC端点从您的内部直接连接到 TimestreamVPC，请完成以下步骤：

1. 首先，找到一个可用的 Timestream 蜂窝端点。要查找可用的蜂窝终端节点，请使用`DescribeEndpoints`操作（可通过 “写入” 和 “查询”APIs）列出您的 Timestream 账户中可用的单元终端节点。有关更多详细信息，请参阅示例。
2. 选择要使用的单元终端节点后，请为 Timestream 写入或查询API创建VPC接口端点字符串：
   - 对于写作API：

```
com.amazonaws.<region>.timestream.ingest-<cell>
```

- 对于查询API：

```
com.amazonaws.<region>.timestream.query-<cell>
```

其中 *<region>* 是有效的 AWS 地区代码，并且 *<cell>* 是DescribeEndpoints 操作在 Endpoints 对象中返回的单元终端节点地址（例如cell1或cell2）之一。有关更多详细信息，请参阅示例。

3. 现在，您已经构造了VPC终端节点服务名称，请创建一个接口终端节点。当系统要求提供VPC终端节点服务名称时，请使用您在步骤 2 中构造的VPC终端节点服务名称。

示例：构造您的VPC终端节点服务名称

在以下示例中，DescribeEndpoints操作是在区域中 AWS CLI使用 Write API in the us-west-2 区域执行的：

```
aws timestream-write describe-endpoints --region us-west-2
```

此命令将返回以下输出：

```
{
    "Endpoints": [
        {
            "Address": "ingest-cell1.timestream.us-west-2.amazonaws.com",
            "CachePeriodInMinutes": 1440
        }
    ]
}
```

在这种情况下，*cell1* 是 *<cell>* ，以及 *us-west-2* 是 *<region>*。 因此，生成的VPC端点服务名称将如下所示：

```
com.amazonaws.us-west-2.timestream.ingest-cell1
```

现在，您已经为 Timestream 创建了接口VPC终端节点 LiveAnalytics，请为其创建一个 Timestream 的VPC终端节点策略。 LiveAnalytics

为 Timestream 创建VPC终端节点策略 LiveAnalytics

您可以将终端节点策略附加到您的终端节VPC点，以控制对 Timestream 的 LiveAnalytics访问权限。该策略指定以下信息：

- 可执行操作的主体。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅 Amazon VPC 用户指南中的使用VPC终端节点控制对服务的访问。

示例：Timestream 操作的VPC端点策略 LiveAnalytics

以下是 Timestream 的终端节点策略示例。 LiveAnalytics当连接到终端节点时，此策略允许所有委托人访问列出的 Timestream，以便对所有资源 LiveAnalytics 执行操作（在本例中为 `ListDatabases`）。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "timestream:ListDatabases"
         ],
         "Resource":"*"
      }
   ]
}
```

适用于 Amazon Timestream 的安全最佳实践 LiveAnalytics

Amazon Timestream LiveAnalytics 提供了许多安全功能，供您在制定和实施自己的安全策略时考虑。以下最佳实操是一般准则，并不代表完整的安全解决方案。这些最佳实操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

- LiveAnalytics 预防性安全最佳实践的时间流

LiveAnalytics 预防性安全最佳实践的时间流

以下最佳实践可以帮助您预测和防止 Timestream 中的安全事件。 LiveAnalytics

静态加密

LiveAnalytics Timestream for 使用存储在 AWS Key Management Service ()AWS KMS 中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。

Timestream for LiveAnalytics 使用单个服务默认密钥（AWS 自有CMK）来加密您的所有表。如果此密钥不存在，则会为您创建。服务默认密钥无法禁用。有关更多信息，请参阅静 LiveAnalytics 态加密的时间流。

使用IAM角色来验证对 Timestream 的访问权限 LiveAnalytics

要访问Timestream的用户、应用程序和其他 AWS 服务 LiveAnalytics，他们必须在 AWS API请求中包含有效的 AWS 凭证。您不应将 AWS 凭证直接存储在应用程序或EC2实例中。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。IAM角色使您能够获得可用于访问 AWS 服务和资源的临时访问密钥。

有关更多信息，请参阅 IAM 角色。

使用 Timestream IAM 策略进行 LiveAnalytics 基本授权

在授予权限时，你可以决定谁在获得权限、 LiveAnalyticsAPIs他们获得哪个 Timestream 的权限，以及你想允许对这些资源执行哪些具体操作。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），从而授予在 Timestream 上对 LiveAnalytics资源执行操作的权限。

可以通过以下方法实现：

- AWS 托管（预定义）策略
- 客户管理型策略
- 基于标签的授权

考虑客户端加密

如果您将敏感或机密数据存储在 Timestream 中 LiveAnalytics，则可能需要在尽可能靠近其来源的地方加密这些数据，以便您的数据在其整个生命周期中受到保护。对传输中和静态的敏感数据进行加密有助于确保任何第三方都无法获得您的纯文本数据。

使用其他服务

Amazon Timestream 版与各种 AWS 服务和流行的第三方工具 LiveAnalytics 集成。目前，Timestream LiveAnalytics 支持与以下内容的集成：

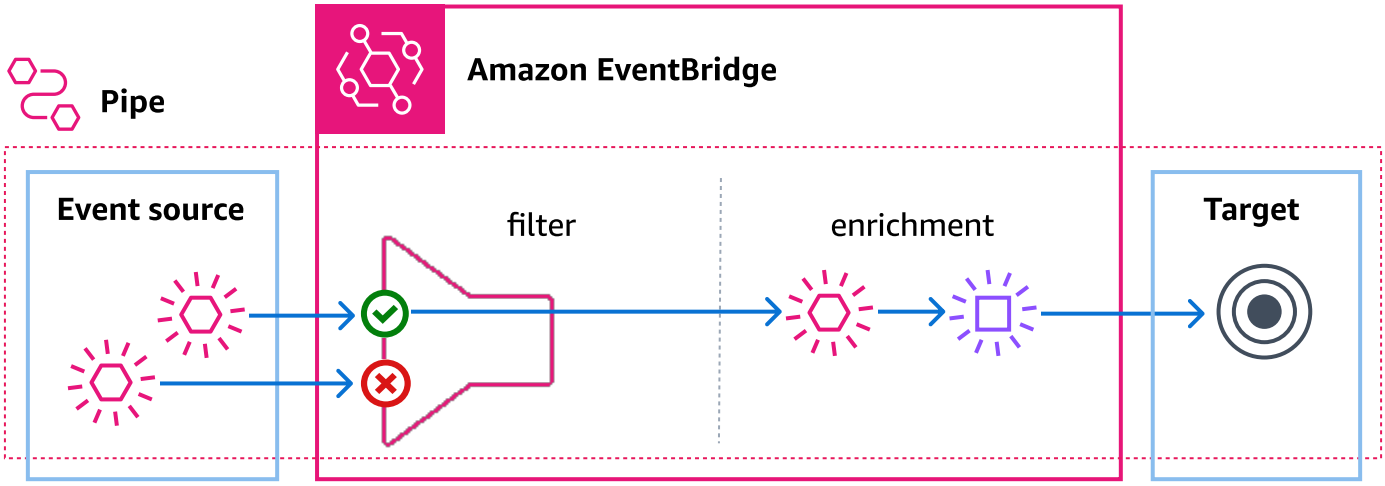
主题

Amazon DynamoDB

使用 EventBridge 管道将 DynamoDB 数据发送到 Timestream

你可以使用 Pip EventBridge es 将数据从 DynamoDB 流发送到表格的亚马逊 Timestream。LiveAnalytics

Pipes 旨在在支持的源和目标之间 point-to-point进行集成，并支持高级转换和扩展。Pipes 减少了开发事件驱动架构时对专业知识和集成代码的需求。要设置管道，请选择源、添加可选筛选、定义可选富集，然后为事件数据选择目标。



有关 EventBridge 管道的更多信息，请参阅《EventBridge 用户指南》中的EventBridge 管道。有关配置管道以将事件传送到 Amazon Timestream LiveAnalytics 表的信息，请参阅 Pip EventBridge es 目标细节。

AWS Lambda

您可以为创建与 Timestream 交互的 Lambda 函数。 LiveAnalytics例如，您可以创建一个定期运行的 Lambda 函数，以便在 Timestream 上执行查询，并根据满足一个或多个条件的查询结果发送SNS通知。要了解有关 Lambda 的更多信息，请参阅 Lambda 文档AWS 。

主题

- 使用 AWS Amazon Tim LiveAnalytics estream 为 Python 构建 Lambda 函数
- 使用 Amazon Timestre AWS am 构建 Lambda 函数 LiveAnalytics JavaScript
- 使用 Amazon Tim LiveAnalytics estre AWS am 为 Go 构建 Lambda 函数
- 使用 Amazon Tim LiveAnalytics estre AWS am 为 C# 构建 Lambda 函数

使用 AWS Amazon Tim LiveAnalytics estream 为 Python 构建 Lambda 函数

要使用 LiveAnalytics 带有 Python 的 Amazon Timestream 构建 AWS Lambda 函数，请按照以下步骤操作。

1. 创建一个IAM角色让 Lambda 代入该角色，该角色将授予访问 Timestream 服务所需的权限，如中所述。提供访问时间 LiveAnalytics 流

2. 编辑IAM角色的信任关系以添加 Lambda 服务。您可以使用以下命令更新现有角色，以便 AWS Lambda 可以代入该角色：

   a. 创建信任策略文档：

```
cat > Lambda-Role-Trust-Policy.json << EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

   b. 使用信任文档更新上一步中的角色

```
aws iam update-assume-role-policy --role-name <name_of_the_role_from_step_1> --
policy-document file://Lambda-Role-Trust-Policy.json
```

相关参考文献位于TimestreamWrite和TimestreamQuery。

使用 Amazon Timestre AWS am 构建 Lambda 函数 LiveAnalytics JavaScript

要使用 Amazon Time LiveAnalytics stre AWS am for with 构建 Lambda 函数，请按照此处概述的说明进行操作。 JavaScript

相关参考文献位于 v3 的 Timestream Write Client 和 JavaScript v3 的 Timestream 查询客户端 AWS SDK。 AWS SDK JavaScript

使用 Amazon Tim LiveAnalytics estre AWS am 为 Go 构建 Lambda 函数

要使用带有 Go 的 Amazon Time LiveAnalytics stream 构建 AWS Lambda 函数，请按照此处概述的说明进行操作。

相关参考是 timestreamwrite 和 timestreamquer y。

使用 Amazon Tim LiveAnalytics estre AWS am 为 C# 构建 Lambda 函数

要使用带有 C# 的 Amazon Time LiveAnalytics stream 构建 AWS Lambda 函数，请按照此处概述的说明进行操作。

相关参考资料可在 Amazon 上找到。 TimestreamWrite还有亚马逊。 TimestreamQuery。

AWS IoT Core

您可以使用物联网核心从物联网设备收集数据，并通过AWS 物联网核心规则操作将数据路由到 Amazon Timestream。 AWS IoT 规则操作指定触发规则时要执行的操作。您可以定义将数据发送到亚马逊 Timestream 表、亚马逊 DynamoDB 数据库和调用 Lambda 函数的操作。 AWS

物联网规则中的 Timestream 操作用于将来自传入消息的数据直接插入到 Timestream 中。该操作解析物联网核心SQL语句的结果并将数据存储在 Timestream 中。返回的SQL结果集中字段的名称用作度量:: name，字段的值是度量:: value。

例如，考虑SQL语句和示例消息负载：

```
SELECT temperature, humidity from 'iot/topic'
```

```
{
  "dataFormat": 5,
  "rssi": -88,
  "temperature": 24.04,
  "humidity": 43.605,
  "pressure": 101082,
  "accelerationX": 40,
  "accelerationY": -20,
  "accelerationZ": 1016,
  "battery": 3007,
  "txPower": 4,
  "movementCounter": 219,
  "device_id": 46216,
  "device_firmware_sku": 46216
}
```

如果使用上述SQL语句创建了 Timestream 的物联网核心规则操作，则将向 Timestream 添加两条记录，其测量名称分别为 24.04 和 43.605，测量值分别为 24.04 和 43.605。

您可以通过在SELECT语句中使用 AS 运算符修改正在添加到 Timestream 的记录的度量名称。下面的SQL语句将使用消息名称 temp 而不是 temperature 来创建一条记录。

度量的数据类型是根据消息负载值的数据类型推断出来的。JSON整数、双精度、布尔值和字符串等数据类型VARCHAR分别映射到 Timestream 数据类型BIGINTDOUBLE、BOOLEAN、和。也可以使用 cast () 函数将数据强制转换为特定的数据类型。您可以指定测量的时间戳。如果时间戳留空，则使用处理条目的时间。

您可以参阅 Timestream 规则操作文档了解更多详细信息

要创建物联网核心规则操作以将数据存储在 Timestream 中，请执行以下步骤：

主题

先决条件

1. 按照中所述的说明在 Amazon Timestream 中创建数据库。创建 数据库
2. 按照中所述的说明在 Amazon Timestream 中创建表。创建表

使用控制台

1. 使用适用于 AWS IoT Core 的 AWS 管理控制台创建规则，方法是单击 “管理” > “消息路由” > “规则”，然后单击 “创建规则”。
2. 将规则名称设置为您选择的名称，并将设置SQL为如下所示的文本

```
SELECT temperature as temp, humidity from 'iot/topic'
```

3. 从 “操作” 列表中选择 “时间流”
4. 指定 Timestream 数据库、表和维度名称以及将数据写入 Timestream 的角色。如果该角色不存在，则可以通过单击 “创建角色” 来创建一个角色
5. 要测试规则，请按照此处显示的说明进行操作。

使用 CLI

如果您尚未安装 AWS 命令行界面 (AWS CLI)，请从此处安装。

1. 将以下规则负载保存在名为 timestream_rule.json JSON 的文件中。Replace（替换）*arn:aws:iam::123456789012:role/TimestreamRole* 使用你的角色 arn，它授予 AWS 物联网访问权限，以便在 Amazon Timestream 中存储数据

```
{
    "actions": [
            {
                "timestream": {
                    "roleArn": "arn:aws:iam::123456789012:role/TimestreamRole",
                    "tableName": "devices_metrics",
                    "dimensions": [
                        {
                            "name": "device_id",
                            "value": "${clientId()}"
                        },
                        {
                            "name": "device_firmware_sku",
                            "value": "My Static Metadata"
                        }
                    ],
                    "databaseName": "record_devices"
                }
            }
    ],
    "sql": "select * from 'iot/topic'",
    "awsIotSqlVersion": "2016-03-23",
    "ruleDisabled": false
}
```

2. 使用以下命令创建主题规则

```
aws iot create-topic-rule --rule-name timestream_test --topic-rule-payload file://
<path/to/timestream_rule.json> --region us-east-1
```

3. 使用以下命令检索主题规则的详细信息

```
aws iot get-topic-rule --rule-name timestream_test
```

4. 将以下消息负载保存在名为 timestream_msg.json 的文件中

```
{
  "dataFormat": 5,
  "rssi": -88,
  "temperature": 24.04,
  "humidity": 43.605,
  "pressure": 101082,
  "accelerationX": 40,
  "accelerationY": -20,
  "accelerationZ": 1016,
  "battery": 3007,
  "txPower": 4,
  "movementCounter": 219,
  "device_id": 46216,
  "device_firmware_sku": 46216
}
```

5. 使用以下命令测试规则

```
aws iot-data publish --topic 'iot/topic' --payload file://<path/to/
timestream_msg.json>
```

示例应用程序

为了帮助您开始使用带有 AWS IoT Core 的 Timestream，我们创建了一个功能齐全的示例应用程序，它在 AWS IoT Core 和 Timestream 中创建了用于创建主题规则的必要工件，以及用于向主题发布数据的示例应用程序。

1. 按照中的说明克隆示例应用程序的 GitHub 存储库，以 AWS 实现 IoT Core 集成 GitHub
2. 按照中的说明使用 AWS CloudFormation 模板在 READMEAmazon Timestream 和 Io AWS T Core 中创建必要的项目，并向该主题发布示例消息。

视频教程

该视频解释了物联网核心如何与 Timestream 配合使用。

适用于 Apache Flink 的亚马逊托管服务

你可以使用 Apache Flink 将你的时间序列数据从适用于 Apache Flink 的亚马逊托管服务、亚马逊、Apache Kafka 和其他流媒体技术直接传输到 A MSK mazon Timestream 中。 LiveAnalytics我们

已经为 Timestream 创建了一个 Apache Flink 示例数据连接器。我们还创建了一个示例应用程序，用于将数据发送到亚马逊 Kinesis，这样数据就可以从 Kinesis 流到 Apache Flink 的托管服务，最后流向亚马逊 Timestream。所有这些工件都可以在中找到 GitHub。本视频教程描述了设置。

> **Note**
>
> Java 11 是使用 Apache Flink 应用程序托管服务的推荐版本。如果您有多个 Java 版本，请确保将 Java 11 导出到您的 JAVA _ HOME 环境变量中。

主题

示例应用程序

要开始使用，请按照以下步骤操作：

1. 按照中所述的说明在 Timestream 中创建名称`kdaflink`的数据库 创建 数据库
2. 按照中所述的说明在 Timestream 中创建一个名称`kinesisdata1`的表 创建表
3. 按照创建流中所述的说明，使用名称`TimestreamTestStream`创建 Amazon Kinesis 数据流
4. 按照 GitHub 中的说明克隆适用于 Timestream 的 Apache Flink 数据连接器的存储库 GitHub
5. 要编译、运行和使用示例应用程序，请按照 Apache Flink 示例数据连接器中的说明进行操作 README
6. 按照编译应用程序代码的说明编译适用于 Apache Flink 的托管服务
7. 按照上传 Apache Flink 流媒体代码的说明上传适用于 Apache Flink 应用程序的托管服务 Flink 应用程序二进制文件

    a. 单击 “创建应用程序” 后，单击该应用程序的 “IAM角色” 链接

    b. 附上AmazonKinesisReadOnlyAccess和的IAM政策AmazonTimestreamFullAccess。

    > **Note**
    >
    > 上述IAM政策不限于特定资源，不适合生产用途。对于生产系统，可以考虑使用限制对特定资源的访问的策略。

8. 按照 GitHub 中的说明克隆将数据写入 Kinesis 的示例应用程序的存储库 GitHub
9. 按照中的说明运行用于README向 Kinesis 写入数据的示例应用程序
10. 按照以下说明在 Timestream 中运行一个或多个查询，确保数据从 Kinesis 发送到托管服务，让 Apache Flink 发送到 Timestream 创建表

视频教程

本视频介绍了如何将 Timestream 与适用于 Apache Flink 的托管服务一起使用。

Amazon Kinesis

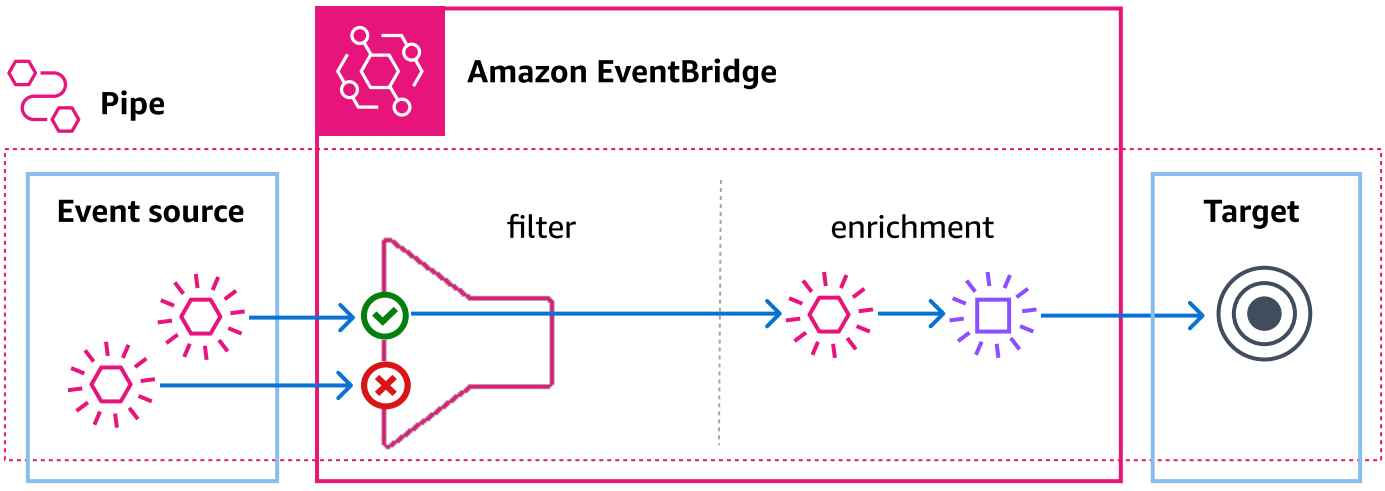
使用 适用于 Apache Flink 的亚马逊托管服务

您可以将数据从 Kinesis Data Streams 发送到 Timestream LiveAnalytics ，以便使用 Apache Flink 托管服务的 Timestream 示例数据连接器。有关更多信息，适用于 Apache Flink 的亚马逊托管服务请参阅 Apache Flink。

使用 EventBridge 管道将 Kinesis 数据发送到 Timestream

你可以使用 Pip EventBridge es 将数据从 Kinesis 流发送到 Amazon Time LiveAnalytics stream 作为表格。

Pipes 旨在在支持的源和目标之间 point-to-point进行集成，并支持高级转换和扩展。Pipes 减少了开发事件驱动架构时对专业知识和集成代码的需求。要设置管道，请选择源、添加可选筛选、定义可选富集，然后为事件数据选择目标。



这种集成使您能够利用时间序列数据分析功能 Timestream的强大功能，同时简化您的数据摄取管道。

将 Pip EventBridge es 与配合使用 Timestream 具有以下好处：

- 实时数据摄取：将数据从 Kinesis 直接流式传输到 Timestream LiveAnalytics，从而实现实时分析和监控。
- 无缝集成：利用 EventBridge 管道管理数据流，无需复杂的自定义集成。
- 增强的筛选和转换：在 Kinesis 记录存储之前对其进行筛选或转换 Timestream ，以满足您的特定数据处理要求。
- 可扩展性：利用内置的并行和批处理功能，处理高吞吐量数据流，确保高效的数据处理。

配置

要设置 Pi EventBridge pe 以将数据从 Kinesis 流式传输到 Timestream，请按照以下步骤操作：

1. 创建 Kinesis 流

   确保您有一个活跃的 Kinesis 数据流，您要从中提取数据。
2. 创建 Timestream 数据库和表

   设置存储数据的 Timestream 数据库和表。
3. 配置 EventBridge 管道：
   - 来源：选择你的 Kinesis 直播作为来源。
   - 目标：选择 Timestream 作为目标。
   - 批处理设置：定义批处理窗口和批处理大小，以优化数据处理并减少延迟。

> **Important**
>
> 设置管道时，我们建议通过摄取一些记录来测试所有配置的正确性。请注意，成功创建管道并不能保证管道的正确性，也不能保证数据流畅无误。可能存在运行时错误，例如表不正确、动态路径参数不正确或应用映射后的 Timestream 记录无效，这些错误将在实际数据流过管道时被发现。

以下配置决定了数据摄取的速率：

- BatchSize：将发送到 Timestream 的批次的最大大小。 LiveAnalytics射程：0-100。建议将此值保持为 100 以获得最大吞吐量。

- MaximumBatchingWindowInSeconds：在将批次发送到 Timestream 作为 LiveAnalytics 目标 batchSize 之前，等待填充的最长时间。根据传入事件的速率，此配置将决定摄取的延迟，建议将此值保持 Timestream 在 < 10 秒，以保持近乎实时地向其发送数据。
- ParallelizationFactor：每个分片中要同时处理的批次数。建议使用最大值 10 来获得最大吞吐量和近乎实时的摄取。

  如果您的直播被多个目标读取，请使用增强的扇出功能为您的管道提供专门的使用者，以实现高吞吐量。有关更多信息，请参阅使用Kinesis Data Streams 用户指南 Kinesis Data Streams API中的开发增强型扇出消费者。

> **Note**
>
> 可以实现的最大吞吐量受每个账户并发管道执行的限制。

以下配置可确保防止数据丢失：

- DeadLetterConfig：建议始终进行配置， DeadLetterConfig 以避免 LiveAnalytics 由于用户错误而无法将事件提取到 Timestream 时丢失任何数据。

使用以下配置设置优化管道的性能，这有助于防止记录导致速度减慢或阻塞。

- MaximumRecordAgeInSeconds: 超过此日期的记录将不会被处理，而是会直接移至DLQ。我们建议将此值设置为不高于目标 Timestream 表配置的内存存储保留期。
- MaximumRetryAttempts：记录发送到 DeadLetterQueue之前重试记录的次数。建议将其配置为 10。这应该能够帮助解决任何暂时性问题，对于持续存在的问题，记录将被移至直播的其余部分 DeadLetterQueue 并解除封锁。
- OnPartialBatchItemFailure：对于支持部分批处理的来源，我们建议您启用此功能并将其配置为 AUTOMATIC _，以便在删除/发送到之前BISECT对失败的记录进行额外重试。DLQ

配置示例

以下是如何配置 Pi EventBridge pe 以将数据从 Kinesis 流传输到表的 Timestream 示例：

Example IAM 的政策更新 Timestream

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:WriteRecords"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:123456789012:database/my-database/table/
my-table"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*"
        }
    ]
}
```

Example Kinesis 直播配置

```
{
  "Source": "arn:aws:kinesis:us-east-1:123456789012:stream/my-kinesis-stream",
  "SourceParameters": {
    "KinesisStreamParameters": {
        "BatchSize": 100,
        "DeadLetterConfig": {
            "Arn": "arn:aws:sqs:us-east-1:123456789012:my-sqs-queue"
        },
       "MaximumBatchingWindowInSeconds": 5,
        "MaximumRecordAgeInSeconds": 1800,
        "MaximumRetryAttempts": 10,
        "StartingPosition": "LATEST",
       "OnPartialBatchItemFailure": "AUTOMATIC_BISECT"
    }
  }
}
```

Example Timestream 目标配置

```
{
    "Target": "arn:aws:timestream:us-east-1:123456789012:database/my-database/table/my-
table",
    "TargetParameters": {
        "TimestreamParameters": {
            "DimensionMappings": [
                {
                    "DimensionName": "sensor_id",
                    "DimensionValue": "$.data.device_id",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "DimensionName": "sensor_type",
                    "DimensionValue": "$.data.sensor_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "DimensionName": "sensor_location",
                    "DimensionValue": "$.data.sensor_loc",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": [
                {
                    "MultiMeasureName": "readings",
                    "MultiMeasureAttributeMappings": [
                        {
                            "MultiMeasureAttributeName": "temperature",
                            "MeasureValue": "$.data.temperature",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "MultiMeasureAttributeName": "humidity",
                            "MeasureValue": "$.data.humidity",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "MultiMeasureAttributeName": "pressure",
                            "MeasureValue": "$.data.pressure",
                            "MeasureValueType": "DOUBLE"
                        }
                    ]
```

```
                }
            ],
            "SingleMeasureMappings": [],
            "TimeFieldType": "TIMESTAMP_FORMAT",
            "TimestampFormat": "yyyy-MM-dd HH:mm:ss.SSS",
            "TimeValue": "$.data.time",
            "VersionValue": "$.approximateArrivalTimestamp"
        }
    }
}
```

事件转换

EventBridge 管道允许您在数据到达之前对其进行转换 Timestream。您可以定义转换规则来修改传入的 Kinesis 记录，例如更改字段名称。

假设您的 Kinesis 直播中包含温度和湿度数据。在将这些字段插入之前，您可以使用 EventBridge 转换来重命名这些字段 Timestream。

最佳实践

批处理和缓冲

- 配置批处理窗口和大小，以便在写入延迟和处理效率之间取得平衡。
- 在处理之前，使用批处理窗口积累足够的数据，从而减少频繁的小批量处理的开销。

并行处理

利用该ParallelizationFactor设置来提高并发度，特别是对于高吞吐量流。这样可以确保可以同时处理来自每个分片的多个批次。

数据转换

利用 Pip EventBridge es 的转换功能对记录进行筛选和增强，然后再将其存储到其中 Timestream。这有助于使数据与您的分析要求保持一致。

安全性

- 确保用于 Pip EventBridge es 的IAM角色具有读取 Kinesis 和写入所需的权限 Timestream。
- 使用加密和访问控制措施来保护传输中的数据和静态数据。

调试失败

- 自动禁用管道

  如果目标不存在或存在权限问题，则管道将在大约 2 小时后自动禁用
- 限制

  管道能够自动退缩并重试，直到油门减小。
- 启用日志

  我们建议您启用ERROR级别的日志并包含执行数据，以便更深入地了解失败。如果出现任何故障，这些日志将包含request/response sent/received来自 Timestream。这可以帮助您了解相关的错误，并在修复错误后根据需要重新处理记录。

监控

我们建议您设置以下警报，以检测数据流的任何问题：

- 来源中记录的最大保存期限
  - `GetRecords.IteratorAgeMilliseconds`
- 管道中的故障指标
  - `ExecutionFailed`
  - `TargetStageFailed`
- Timestream 写入API错误
  - `UserErrors`

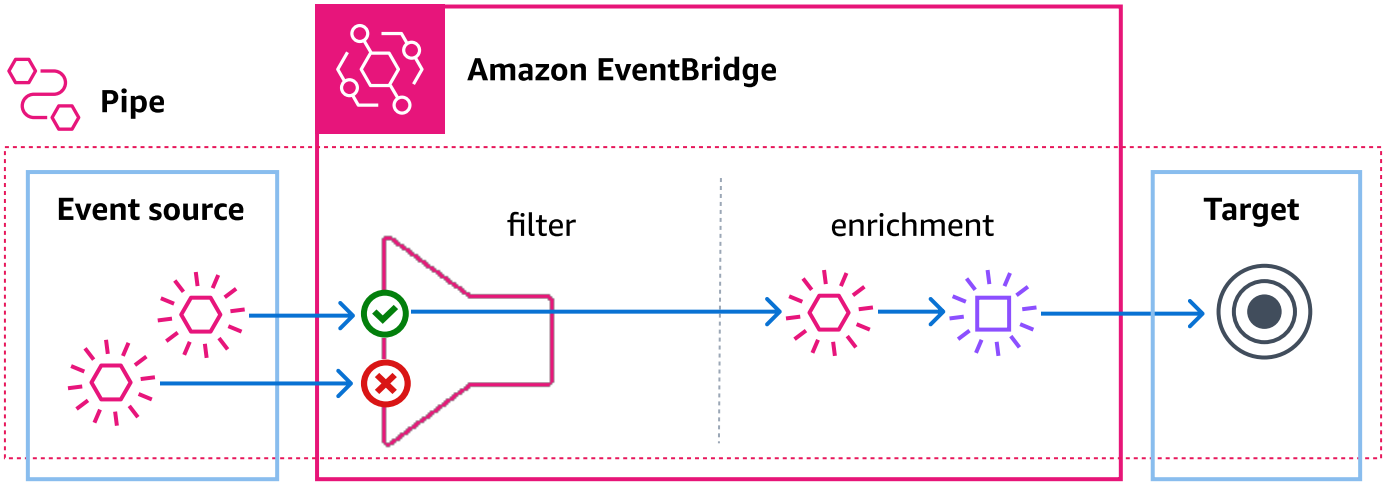
有关其他监控指标，请参阅《EventBridge 用户指南》 EventBridge中的监控。

Amazon MQ

使用 Pip EventBridge es 将 Amazon MQ 数据发送至 Timestream

你可以使用 Pip EventBridge es 将数据从亚马逊 MQ 经纪商发送到亚马逊 Time LiveAnalytics stream 以获取表格。

Pipes 旨在在支持的源和目标之间 point-to-point进行集成，并支持高级转换和扩展。Pipes 减少了开发事件驱动架构时对专业知识和集成代码的需求。要设置管道，请选择源、添加可选筛选、定义可选富集，然后为事件数据选择目标。



有关 EventBridge 管道的更多信息，请参阅《EventBridge 用户指南》中的EventBridge 管道。有关配置管道以将事件传送到 Amazon Timestream LiveAnalytics 表的信息，请参阅 Pip EventBridge es 目标细节。

Amazon MSK

使用适用于 Apache Flink 的托管服务向 Timestrea Amazon MSK m 发送数据 LiveAnalytics

Timestream 通过构建类似于 Apache Flink 托管服务的示例 Timestream 数据连接器的数据连接器，可以将数据从 Amazon MSK 发送到。有关更多信息，请参阅适用于 Apache Flink 的亚马逊托管服务。

使用 Kafka Connect 将亚马逊MSK数据发送到 Timestream LiveAnalytics

你可以使用 Kafka Connect 将你的时间序列数据 Amazon MSK 直接提取到 Timestream 中。LiveAnalytics

我们已经为创建了 Kafka Sink 连接器示例。 Timestream我们还创建了一个样本 Apache jMeter 测试计划，用于将数据发布到 Kafka 主题，这样数据就可以通过 Kaf Timestream ka Sink Connector 从该主题流向表格的 Timestream。 LiveAnalytics所有这些工件都可以在上找到 GitHub。

> **Note**
>
> Java 11 是使用 Timestream Kafka Sink 连接器的推荐版本。如果您有多个 Java 版本，请确保将 Java 11 导出到您的 JAVA _ HOME 环境变量中。

创建示例应用程序

要开始使用，请按照以下步骤操作。

1. 在 Timestream for 中 LiveAnalytics，创建一个名`kafkastream`为的数据库。

   有关详细说明???，请参阅程序。

2. 在 Timestream for 中 LiveAnalytics，创建一个名`purchase_history`为的表。

   有关详细说明???，请参阅程序。

3. 按照中共享的说明创建以下内容：、和。

   - 一个 Amazon MSK 集群
   - 配置为 Kafka 生产者客户端计算机的 Amazon EC2 实例
   - 一个 Kafka 话题

   有关详细说明，请参阅 kafka_ingestor 项目的先决条件。

4. 克隆 Timestream Kafka Sink 连接器存储库。

   有关详细说明，请参阅克隆 GitHub 存储库。

5. 编译插件代码。

   有关详细说明，请参阅连接器-从源代码构建。 GitHub

6. 将以下文件上传到 S3 存储桶：按照中所述的说明进行操作。

   - 目录中的 jar 文件 (kafka-connector-timestream-> VERSION <-jar-with-dependencies .jar) / `target`
   - JSON 架构文件示例，`purchase_history.json`。

   有关详细说明，请参阅Amazon S3 用户指南中的上传对象。

7. 创建两个VPC端点。MSK连接器将使用这些端点访问资源 AWS PrivateLink。

   - 一个可以访问 Amazon S3 存储桶
   - 一个可以访问 LiveAnalytics 表格的 Timestream。

   有关详细说明，请参阅VPC终端节点。

8. 使用上传的 jar 文件创建自定义插件。

   有关详细说明，请参阅《Amazon MSK 开发人员指南》中的插件。

9. 使用工作器配置参数中描述的JSON内容创建自定义工作器配置。按照中描述的说明进行操作

   有关详细说明，请参阅《Amazon MSK 开发人员指南》中的创建自定义工作器配置。

10. 创建服务执行 IAM 角色。

    有关详细说明，请参阅IAM 服务角色。

11. 使用在前面的步骤中创建的自定义插件、自定义工作 Amazon MSK 器配置和服务执行 IAM 角色以及示例连接器配置创建连接器。

    有关详细说明，请参阅《Amazon MSK 开发人员指南》中的创建连接器。

    请务必使用相应的值更新以下配置参数的值。有关详细信息，请参阅连接器配置参数。

    - `aws.region`
    - `timestream.schema.s3.bucket.name`
    - `timestream.ingestion.endpoint`

    连接器创建需要 5-10 分钟才能完成。当管道的状态更改为时，管道已准备就绪`Running`。

12. 发布持续的消息流，用于向创建的 Kafka 主题写入数据。

    有关详细说明，请参阅如何使用它。

13. 运行一个或多个查询，确保数据从 LiveAnalytics 表发送 Amazon MSK 到 MSK Connect to the Timestream。

    有关详细说明???，请参阅程序。

其他资源

这篇博客《使用 Kafka Connect 从你的 Kafka 集群实时无服务器数据提取到 Timestre LiveAnalytics am 中》解释了如何使用 Timestream for LiveAnalytics Kafka Sink Connec tor 设置 end-to-end管道，从使用 Apache jMeter 测试计划向 Kafka 主题发布数千条示例消息的 Kafka 生产者客户端计算机开始，然后在表的时间流中验证提取的记录。 LiveAnalytics

Amazon QuickSight

您可以使用亚马逊 QuickSight 来分析和发布包含您的亚马逊 Timestream 数据的数据控制面板。本节介绍如何创建新的 QuickSight 数据源连接、修改权限、创建新数据集和执行分析。本视频教程介绍了如何使用 Timestream 和 Amazon QuickSight。

> **Note**
>
> Amazon 中的所有数据集 QuickSight 都是只读的。您无法通过使用 Amazon QuickSight 移除数据源、数据集或字段来对 Timestream 中的实际数据进行任何更改。

主题

从以下地址访问亚马逊 Timestream QuickSight

您需要先授权亚马逊连接亚马逊 QuickSight Timestream，然后才能继续。如果未启用连接，则在尝试连接时会收到错误消息。 QuickSight 管理员可以授权 AWS 资源连接。要授权从 QuickSight Timestream 建立连接，请按照使用其他 AWS 服务：缩小访问范围中的步骤进行操作，在步骤 5 中选择 Amazon Timestream。

为 Timestream 创建新的 QuickSight 数据源连接

> **Note**
>
> 亚马逊 QuickSight 和亚马逊 Timestream 之间的连接在传输过程中使用 SSL (TLS1.2) 进行加密。您无法创建未加密的连接。

1. 请确保您已为亚马逊配置了访问亚马逊 QuickSight Timestream 的相应权限，如中所述。从以下地址访问亚马逊 Timestream QuickSight

2. 首先创建一个新数据集。从导航窗格中选择 “数据集”，然后选择 “新建数据集”。
3. 选择 Timestream 数据源卡。
4. 例如US Timestream Data，在数据源名称中，输入您的 Timestream 数据源连接的名称。

> **Note**
>
> 您可以通过与 Timestream 的连接创建许多数据集，因此最好使用简洁的名称。

5. 选择验证连接，检查是否可以成功连接到 Timestream。

> **Note**
>
> 验证连接仅验证您是否可以连接。但是，它不会验证特定的表或查询。

6. 选择创建数据来源以继续。
7. 对于数据库，选择选择... 查看可用选项列表。选择你要使用的那个。
8. 选择 “选择” 继续。
9. 选择以下操作之一：
   - 要将数据导入到 QuickSight的内存引擎（名为SPICE），请选择 “导入到” SPICE 以加快分析速度。
   - QuickSight 要允许在每次刷新数据集或使用分析或仪表板时对数据运行查询，请选择直接查询您的数据。
10. 选择编辑/预览，然后选择保存以保存数据集并将其关闭。

编辑 Timestream QuickSight 的数据源连接权限

以下过程介绍如何查看、添加和撤消其他 QuickSight 用户的权限，以便他们可以访问相同的 Timestream 数据源。这些人必须是活跃用户， QuickSight 然后才能添加他们。

> **Note**
>
> 在中 QuickSight，数据源有两个权限级别：用户和所有者。
>
> - 选择用户以允许读取权限。
> - 选择所有者以允许该用户编辑、共享或删除此 QuickSight 数据源。

1. 请确保您已为亚马逊配置了访问亚马逊 QuickSight Timestream 的相应权限，如中所述。从以下地址访问亚马逊 Timestream QuickSight
2. 选择左侧的数据集，然后向下滚动找到 Timestream 连接的数据来源。例如 US Timestream Data。
3. 选择Timestream数据源卡。
4. 选择Share data source。将显示当前权限列表。
5. （可选）要编辑权限，可以选择user或owner。
6. （可选）要撤消权限，请选择Revoke access。您撤销的用户无法使用此数据源创建新的数据集。但是，他们的现有数据集仍然可以访问该数据源。
7. 要添加权限，请选择Invite users，然后按照以下步骤添加用户：
   a. 添加人员以允许他们使用相同的数据源。
   b. 对于每个Permission，选择你要申请的。
8. 完成后，选择 Close。

为 Timestream 创建新的 QuickSight 数据集

1. 请确保您已为亚马逊配置了访问亚马逊 QuickSight Timestream 的相应权限，如中所述。从以下地址访问亚马逊 Timestream QuickSight
2. 选择左侧的数据集，然后向下滚动找到 Timestream 连接的数据来源。如果您有许多数据源，则可以使用页面顶部的搜索栏来查找名称部分匹配的数据源。
3. 选择 Timestream 数据源卡。然后选择 “创建数据集”。
4. 对于数据库，选择选择以查看可用选项列表。选择要使用的数据库。
5. 对于表，选择要使用的表。
6. 选择编辑/预览。
7. （可选）要添加更多数据，请选择右上角的添加数据。
   a. 选择 “切换数据源”，然后选择其他数据源。
   b. 按照 UI 提示完成数据添加。
   c. 将新数据添加到同一数据集后，选择配置此联接（两个红点）。为每个附加表设置联接。
   d. 如果要添加计算字段，请选择添加计算字段。
   e. 要使用 Sagemaker，请选择 A ug ment with。 SageMaker此选项仅在 QuickSight 企业版中可用。

f. 取消选中任何要省略的字段。

g. 更新您要更改的任何数据类型。

8. 完成后，选择保存，以保存并关闭数据集。

为 Timestream 创建新的分析

1. 请确保您已为亚马逊配置了访问亚马逊 QuickSight Timestream 的相应权限，如中所述。从以下地址访问亚马逊 Timestream QuickSight
2. 选择左侧的分析。
3. 选择以下操作之一：

   - 要创建新分析，请选择右侧的新分析。
   - 要将 Timestream 数据集添加到现有分析，请打开要编辑的分析。选择左上角附近的铅笔图标，然后选择添加数据组。

4. 通过选择左侧的字段开始第一个数据可视化。
5. 有关更多信息，请参阅使用分析-Amazon QuickSight

视频教程

这段视频解释了亚马逊如何 QuickSight 与 Timestream 合作。

Amazon SageMaker

您可以使用亚马逊 SageMaker 笔记本将您的机器学习模型与 Amazon Timestream 集成。为了帮助您入门，我们创建了一个处理来自 Timestream 的数据的示例 SageMaker 笔记本。数据从持续发送数据的多线程 Python 应用程序插入到 Timestream 中。示例 SageMaker 笔记本和示例 Python 应用程序的源代码可在中找到 GitHub。

1. 按照和中所述的说明创建数据库创建 数据库和表 创建表
2. 按照中的说明克隆多线程 Python 示例应用程序的 GitHub 存储库 GitHub
3. 按照中的说明克隆示例 Timestream SageMaker 笔记本的 GitHub 存储库。 GitHub
4. 按照 Timestream 中的说明运行应用程序，持续将数据提取到 Timestream 中 README
5. 按照此处所述的说明为亚马逊创建 Amazon SageMaker S3 存储桶。
6. 创建安装了最新 boto3 的 Amazon SageMaker 实例：除了此处描述的说明外，还要按照以下步骤操作：

a. 在 “创建笔记本实例” 页面上，单击 “其他配置”

b. 点击生命周期配置-可选，然后选择创建新的生命周期配置

c. 在创建生命周期配置向导框中，执行以下操作：

i. 在配置中填写所需的名称，例如 `on-start`

ii. 在 “启动笔记本” 脚本中，从 Github 复制粘贴脚本内容

iii. 在粘贴`PACKAGE=scipy`的脚本`PACKAGE=boto3`中替换为。

7. 点击创建配置

8. 前往 AWS 管理控制台中的IAM服务，找到为笔记本实例新创建的 SageMaker执行角色。

9. 将的IAM策略附加`AmazonTimestreamFullAccess`到执行角色。

> **Note**
>
> 该`AmazonTimestreamFullAccess`IAM政策不限于特定资源，不适合生产用途。对于生产系统，可以考虑使用限制对特定资源的访问的策略。

10. 当笔记本实例的状态为时 InService，选择 Ope n Jupyter 为该实例启动 SageMaker 笔记本

11. 选择 “上传” 按钮`Timestream_SageMaker_Demo.ipynb`将文件`timestreamquery.py`上传到笔记本中

12. 选择 `Timestream_SageMaker_Demo.ipynb`

> **Note**
>
> 如果你看到弹出的 “未找到内核”，请选择 conda_python3 并单击 “设置内核”。

13. 修改`DB_NAME`、`TABLE_NAME`、`bucket`、和`ENDPOINT`以匹配训练模型的数据库名称、表名、S3存储桶名称和区域。

14. 选择播放图标来运行各个单元格

15. 当你到达单元时`Leverage Timestream to find hosts with average CPU utilization across the fleet`，请确保输出返回至少 2 个主机名。

> **Note**
>
> 如果输出中的主机名少于 2 个，则可能需要重新运行示例 Python 应用程序，将数据提取到 Timestream 中，线程数和主机规模都更大。

16. 当你到达牢房时`Train a Random Cut Forest (RCF) model using the CPU utilization history`，`train_instance_type`根据训练作业的资源要求进行更改
17. 当你到达单元时`Deploy the model for inference`，`instance_type`根据推理任务的资源要求进行更改

> **Note**
>
> 训练模型可能需要几分钟。训练完成后，您将在单元格的输出中看到“已完成-训练作业已完成”消息。

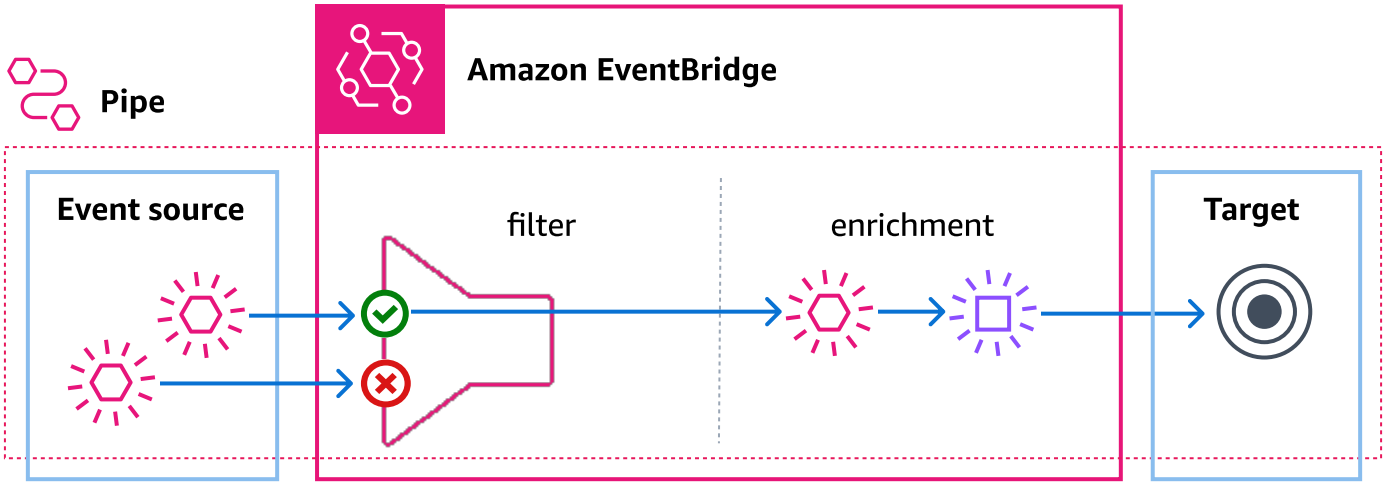
18. 运行单元`Stop and delete the endpoint`以清理资源。您也可以从 SageMaker 控制台停止和删除实例

Amazon SQS

使用 Pip EventBridge es 将亚马逊SQS数据发送至 Timestream

你可以使用 Pip EventBridge es 将数据从亚马逊SQS队列发送到 Amazon Timestream 表格。LiveAnalytics

Pipes 旨在在支持的源和目标之间 point-to-point进行集成，并支持高级转换和扩展。Pipes 减少了开发事件驱动架构时对专业知识和集成代码的需求。要设置管道，请选择源、添加可选筛选、定义可选富集，然后为事件数据选择目标。



有关 EventBridge 管道的更多信息，请参阅《EventBridge 用户指南》中的EventBridge 管道。有关配置管道以将事件传送到 Amazon Timestream LiveAnalytics 表的信息，请参阅 Pip EventBridge es 目标细节。

使用DBeaver与亚马逊 Timestream 合作

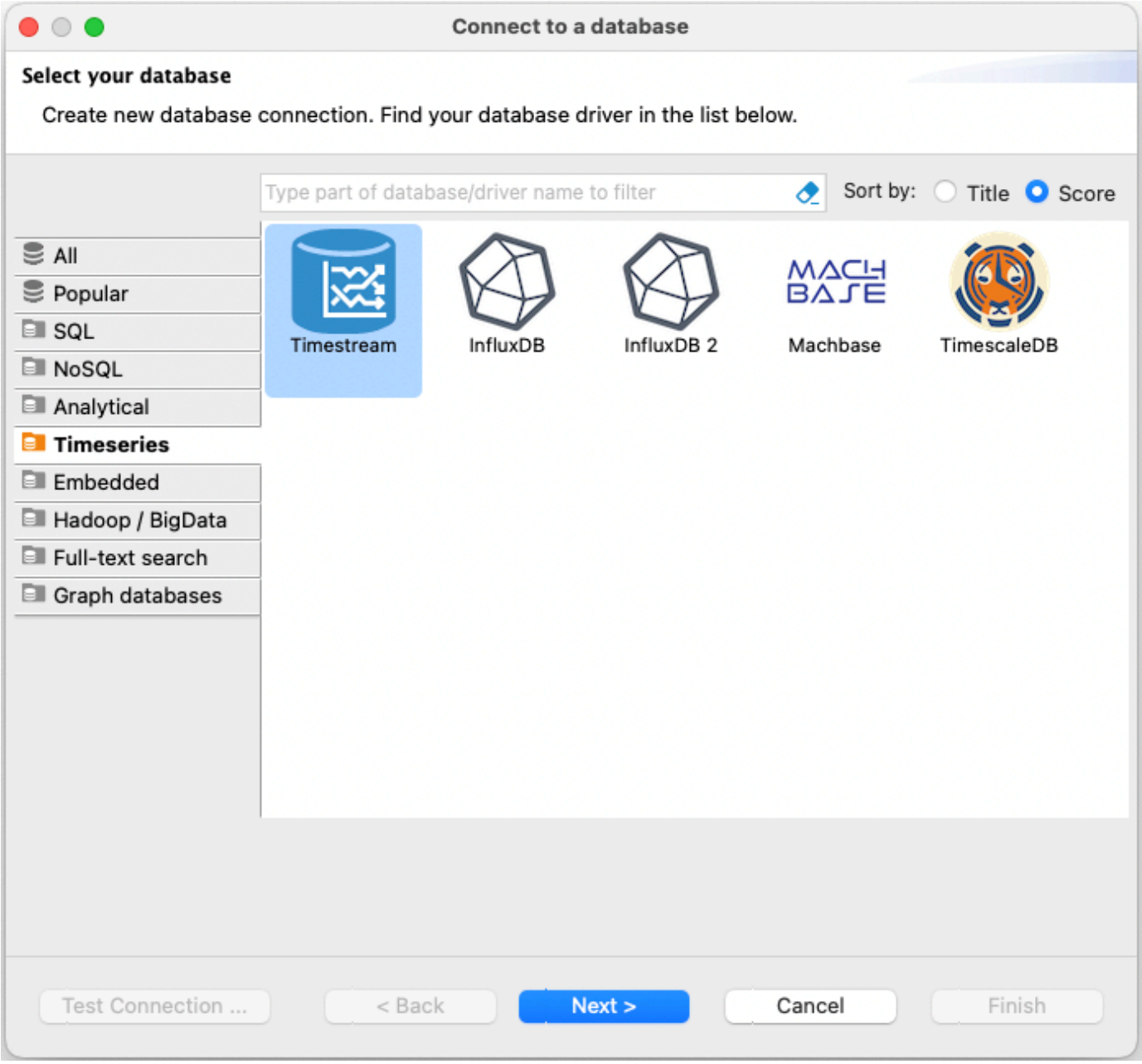
DBeaver是一款免费的通用SQL客户端，可用于管理任何带有JDBC驱动程序的数据库。由于其强大的数据查看、编辑和管理功能，它被开发人员和数据库管理员广泛使用。

使用DBeaver云连接选项，您可以本地DBeaver连接到亚马逊 Timestream。DBeaver提供了一个全面而直观的界面，可以直接在DBeaver应用程序中处理时间序列数据。使用您的证书，它还允许您完全访问可以从其他查询界面执行的任何查询。它甚至允许您创建图表，以便更好地理解和可视化查询结果。

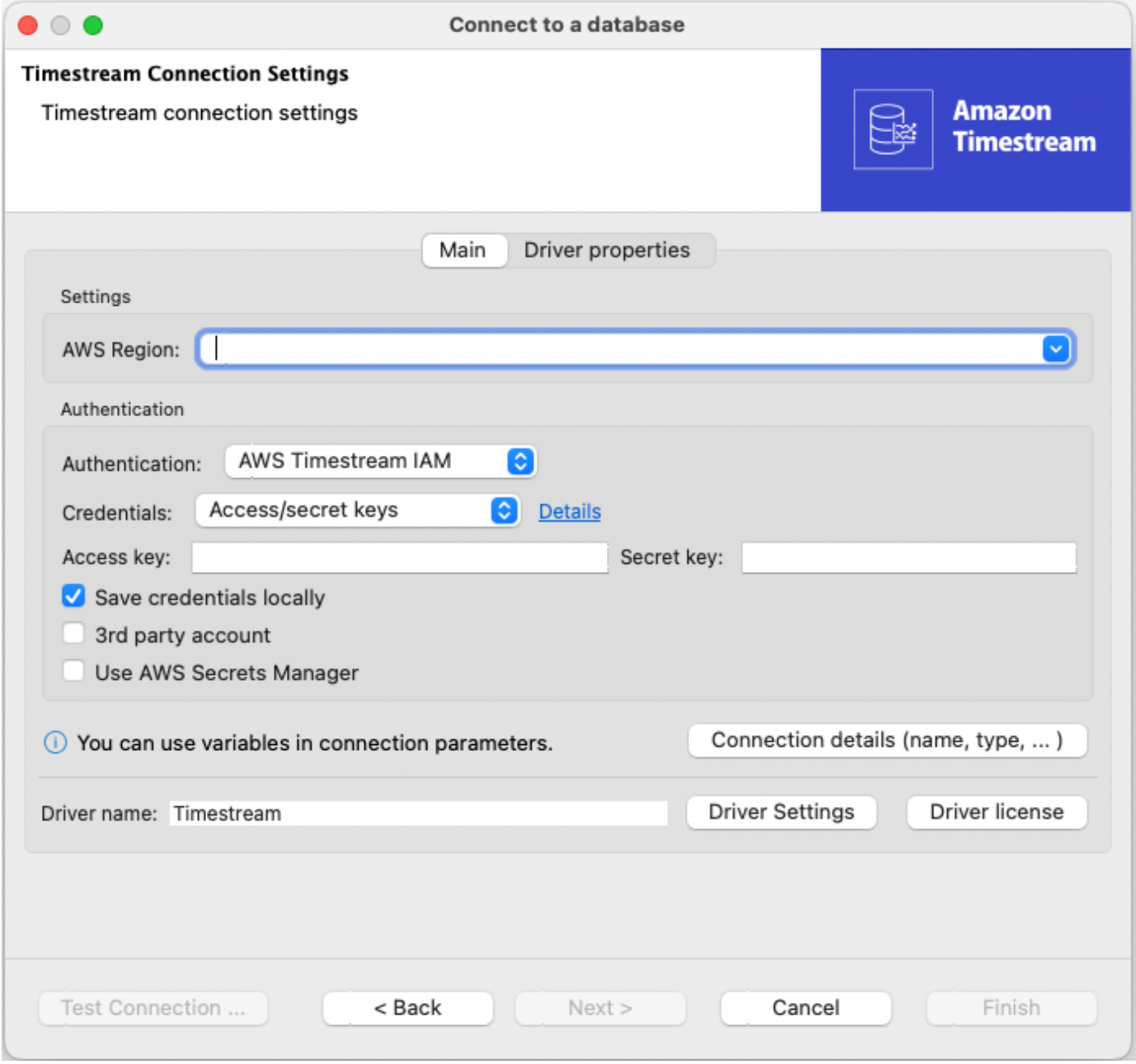
设置DBeaver为使用 Timestream

按照以下步骤进行设置DBeaver以使用 Timestream：

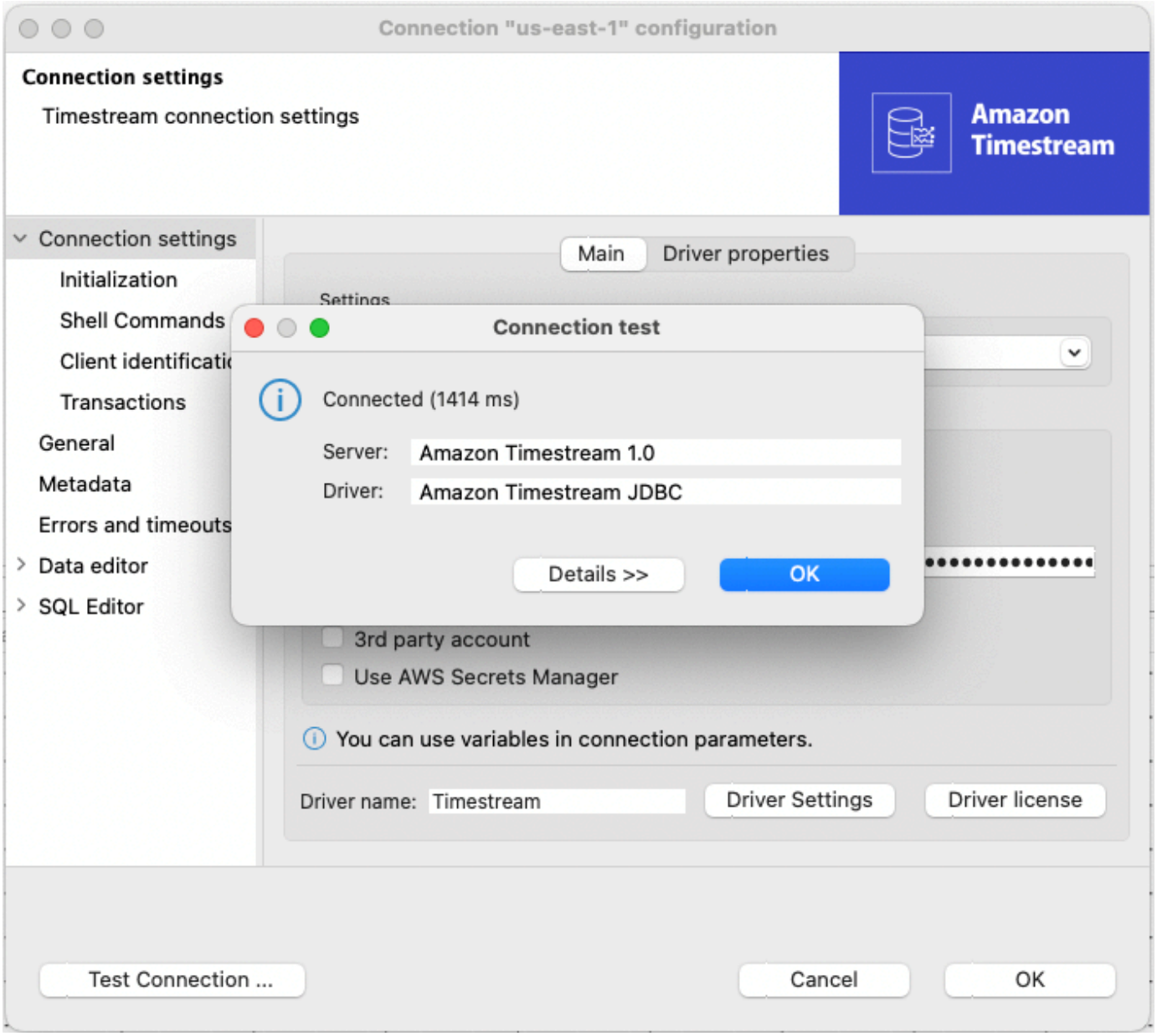
1. 下载并安装到您的本地计算机DBeaver上。
2. 启动DBeaver，导航到数据库选择区域，在左窗格中选择 Timeseries，然后在右窗格中选择时间流图标：



3. 在 Timestream 连接设置窗口中，输入连接到您的 Amazon Timestream 数据库所需的所有信息。请确保您输入的用户密钥具有访问您的 Timestream 数据库所需的权限。另外，请务必将您输入的信息和密钥与任何敏感信息一样DBeaver安全保密。



4. 测试连接以确保一切设置正确：



5. 如果连接测试成功，您现在可以像处理中的任何其他数据库一样与 Amazon Timestream 数据库进行交互。DBeaver例如，您可以导航到SQL编辑器或 ER Diagram 视图来运行查询：

6. DBeaver还提供了强大的数据可视化工具。要使用它们，请运行查询，然后选择图表图标以可视化结果集。绘图工具可以帮助您更好地了解一段时间内的数据趋势。

将 Amazon Timestream 与 Timestream 配对可以为管理时间序列数据DBeaver创建有效的环境。您可以将其无缝集成到现有的工作流程中，以提高生产力和效率。

Grafana

您可以使用 Grafana 可视化时间序列数据并创建警报。为了帮助您开始使用数据可视化，我们在 Grafana 中创建了一个示例仪表板，用于可视化从 Python 应用程序发送到 Timestream 的数据，并创建了一个描述设置的视频教程。

主题

- 示例应用程序
- 视频教程

示例应用程序

1. 要了解更多信息，请按照中所述的说明在 Timestream 中创建 数据库创建数据库和表。

   > **Note**
   >
   > Grafana 控制面板的默认数据库名称和表名分别设置为 GrafanaDB 和。 grafanaTable 使用这些名称可以最大限度地减少设置。

2. 安装 Python 3.7 或更高版本
3. 安装和配置 Timestream Python SDK
4. 克隆多线程 Python 应用程序的 GitHub 存储库，按照中的说明持续将数据提取到 Timestream 中 GitHub
5. 按照 Timestream 中的说明运行应用程序，持续将数据提取到 Timestream 中 README
6. 完成 “亚马逊托管 Grafana 入门” 或完成安装 Grafana。
7. 如果安装 Grafana 而不是使用亚马逊托管 Grafana，请完成安装 Grafana 的 Timestream 插件。
8. 使用您选择的浏览器打开 Grafana 控制面板。如果您已在本地安装了 Grafana，则可以按照 Grafana 文档中描述的说明进行登录
9. 启动 Grafana 后，前往数据源，点击添加数据源，搜索时间流，然后选择时间流数据源
10. 配置身份验证提供商和区域，然后单击 “保存并测试”
11. 设置默认宏
    a. 将 $__database 设置为你的 Timestream 数据库的名称（例如 GrafanaDB）
    b. 将 $__table 设置为你的 Timestream 表的名称（例如）grafanaTable
    c. 将 $__measure 设置为表格中最常用的度量

12. 单击 “保存并测试”
13. 单击 “仪表板” 选项卡
14. 单击 “导入” 以导入仪表板
15. 双击 “示例应用程序控制面板”
16. 点击仪表板设置
17. 选择变量
18. 更改 dbName 和 tableName 以匹配 Timestream 数据库和表的名称
19. 单击 “保存”
20. 刷新仪表板
21. 要创建警报，请按照 Grafana 文档中描述的说明创建 Grafana 托管警报规则
22. 要对警报进行故障排除，请按照 Grafana 故障排除文档中描述的说明进行操作
23. 如需了解更多信息，请参阅 Grafana 文档

视频教程

这段视频解释了 Grafana 是如何与 Timestream 配合使用的。

使用 SquaredUp 与亚马逊 Timestream 合作

SquaredUp是一个与 Amazon Timestream 集成的可观察性平台。您可以使用直观 SquaredUp的仪表板设计器来可视化、分析和监控您的时间序列数据。仪表板可以公开或私下共享，并且可以创建通知渠道，以便在显示器的运行状况发生变化时提醒您。

SquaredUp 与亚马逊 Timestream 搭配使用

1. 注册SquaredUp并免费开始使用。
2. 添加AWS 数据源。
3. 创建使用时间流查询数据流的仪表板磁贴。
4. （可选）启用对磁贴的监控、创建通知渠道或公开或私下共享控制面板。
5. （可选）创建其他切片以查看您的 Timestream 数据以及来自其他监控和可观测性工具的数据。

开源 Telegraf

你可以使用 Telegraf 的 Timestream LiveAnalytics 输出插件将指标 LiveAnalytics直接从开源 Telegraf 写入 Timestream 中。

本节解释了如何使用 Timestream 输出插件安装 Telegraf，如何使用 Timestream LiveAnalytics 输出插件运行 Telegraf，以及开源 Telegraf 如何与 Timestream 配合使用。 LiveAnalytics LiveAnalytics

主题

使用 Timestream 输出插件安装 Telegraf LiveAnalytics

从1.16版本开始，用于 LiveAnalytics 输出的 Timestream 插件已在 Telegraf 的官方版本中提供。要在大多数主流操作系统上安装输出插件，请按照 InfluxData Telegraf 文档中概述的步骤进行操作。要在 Amazon Linux 2 操作系统上安装，请按照以下说明进行操作。

在亚马逊 Linux 2 上安装带有 Timestream LiveAnalytics 输出插件的 Telegraf

要在亚马逊 Linux 2 上安装带有 Timestream 输出插件的 Telegraf，请执行以下步骤。

1. 使用软件yum包管理器安装 Telegraf。

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdb.repo
[influxdb]
name = InfluxDB Repository - RHEL \$releasever
baseurl = https://repos.influxdata.com/rhel/\$releasever/\$basearch/stable
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdb.key
EOF
```

2. 运行以下命令。

```
sudo sed -i "s/\$releasever/$(rpm -E %{rhel})/g" /etc/yum.repos.d/influxdb.repo
```

3. 安装并启动 Telegraf。

```
sudo yum install telegraf
sudo service telegraf start
```

使用 Timestream 输出插件运行 Telegraf LiveAnalytics

你可以按照以下说明使用 Timestream for 插件运行 Telegraf。 LiveAnalytics

1. 使用 Telegraf 生成示例配置。

```
telegraf --section-filter agent:inputs:outputs --input-filter cpu:mem --output-
filter timestream config > example.config
```

2. 使用管理控制台在 Timestream 中创建数据库 CLI，或SDKs。
3. 在example.config文件中，通过编辑该[[outputs.timestream]] 部分下的以下键来添加您的数据库名称。

```
database_name = "yourDatabaseNameHere"
```

4. 默认情况下，Telegraf 将创建一个表。如果您希望手动创建表，请设置create_table_if_not_exists为false并按照说明使用管理控制台创建表 CLI，或SDKs。
5. 在 e xample.config 文件中，在该部分下配置凭据。[[outputs.timestream]] 凭证应允许执行以下操作。

```
timestream:DescribeEndpoints
timestream:WriteRecords
```

> **Note**
>
> 如果将create_table_if_not_exists设置保留为true，请包括：
>
> ```
> timestream:CreateTable
> ```

> **Note**
>
> 如果设置为 describe_database_on_starttrue，请包括以下内容。

```
timestream:DescribeDatabase
```

6. 您可以根据自己的喜好编辑配置的其余部分。
7. 编辑完配置文件后，使用以下命令运行 Telegraf。

```
./telegraf --config example.config
```

8. 指标应在几秒钟内出现，具体取决于您的代理配置。你还应该在 Timestream 控制台中看到新的表 cpu 和 mem。

将 Telegraf/InfluxDB 指标映射到模型的时间流 LiveAnalytics

将数据从 Telegraf 写入 Timestream 时 LiveAnalytics，数据映射如下。

- 时间戳被写成时间字段。
- 标签以尺寸形式写入。
- 字段被写成度量。
- 测量值大多以表名形式写成（详情见下文）。

Telegraf 的 Timestream LiveAnalytics 输出插件提供了多种选项，用于在 Timestream 中组织和存储数据。 LiveAnalytics这可以用一个以线路协议格式的数据开头的示例来描述。

```
weather,location=us-midwest,season=summer temperature=82,humidity=71
1465839830100400200 airquality,location=us-west no2=5,pm25=16
1465839830100400200
```

以下对数据进行了描述。

- 测量名称为`weather`和`airquality`。
- 标签是`location`和`season`。
- 这些字段是`temperature`humidity、no2、和pm25。

主题

- 将数据存储在多个表中
- 将数据存储在单个表中

将数据存储在多个表中

您可以选择为每个测量创建一个单独的表，并将每个字段存储在每个表的单独行中。

配置是`mapping_mode = "multi-table"`。

- LiveAnalytics 适配器的时间流将创建两个表，即`weather`和。`airquality`
- 每个表行将仅包含一个字段。

生成的 LiveAnalytics 表、`weather`和`airquality`、的时间流将如下所示。

`weather`

时间	location	赛季	measure_name	measure_value::bigint
2016-06-13 17:43:50	美国中西部	夏天	温度	82
2016-06-13 17:43:50	美国中西部	夏天	湿度	71

`airquality`

时间	location	measure_name	measure_value::bigint
2016-06-13 17:43:50	美国中西部	no2	5
2016-06-13 17:43:50	美国中西部	pm25	16

将数据存储在单个表中

您可以选择将所有测量值存储在单个表中，并将每个字段存储在单独的表格行中。

配置是`mapping_mode = "single-table"`。使用时还有两种附加配置`single-table`，`single_table_name`和`single_table_dimension_name_for_telegraf_measurement_na`

- LiveAnalytics 输出的 Timestream 插件将创建一个名为的表 *`<single_table_name>`* 其中包括 *`<single_table_dimension_name_for_telegraf_measurement_name>`* 专栏。

- 该表可能在单个表行中包含多个字段。

生成的 LiveAnalytics 表格时间流将如下所示。

`weather`

时间	location	赛季	*<single_table_dimension_name_for_telegraf_measurement_name>*	measure_name	measure_value::bigint
2016-06-13 17:43:50	美国中西部	夏天	天气	温度	82
2016-06-13 17:43:50	美国中西部	夏天	天气	湿度	71
2016-06-13 17:43:50	美国中西部	夏天	空气质量	no2	5
2016-06-13 17:43:50	美国中西部	夏天	天气	pm25	16

JDBC

您可以使用 Java 数据库连接 (JDBC) 连接将 Timestream LiveAnalytics 连接到您的商业智能工具和其他应用程序，例如 SQLWork bench。Okta 和 Microsoft Azure AD 目前支持 LiveAnalytics JDBCSSO 驱动程序的时间流。

主题

- 为 Timestream 配置JDBC驱动程序 LiveAnalytics
- 连接属性
- JDBCURL例子

- 使用 Okta 设置 Timestream 进行 LiveAnalytics JDBC单点登录身份验证
- 使用 Microsoft Azure AD 为 LiveAnalytics JDBC单点登录身份验证设置时间流

为 Timestream 配置JDBC驱动程序 LiveAnalytics

按照以下步骤配置JDBC驱动程序。

主题

- 司机的时间 LiveAnalytics JDBC流 JARs
- LiveAnalytics JDBC驱动程序类和URL格式的时间流
- 示例应用程序

司机的时间 LiveAnalytics JDBC流 JARs

您可以通过直接下载或将 LiveAnalytics JDBC驱动程序添加为 Maven 依赖项来获取驱动程序的 Timestream。

- 直接下载:. 要直接下载 Timestream LiveAnalytics JDBC 驱动程序，请完成以下步骤：
  1. 导航到 https://github.com/awslabs/amazon-timestream-driver-jdbc/releases
  2. 您可以`amazon-timestream-jdbc-1.0.1-shaded.jar`直接与商业智能工具和应用程序一起使用
  3. 下载`amazon-timestream-jdbc-1.0.1-javadoc.jar`到您选择的目录。
  4. 在已下载的目录中`amazon-timestream-jdbc-1.0.1-javadoc.jar`，运行以下命令解压 Javadoc HTML 文件：

     ```
     jar -xvf amazon-timestream-jdbc-1.0.1-javadoc.jar
     ```
- 作为 Maven 依赖项：要将 LiveAnalytics JDBC驱动程序的 Timestream 添加为 Maven 依赖项，请完成以下步骤：
  1. 在您选择的编辑器中导航并打开应用程序的`pom.xml`文件。
  2. 将JDBC驱动程序作为依赖项添加到应用程序`pom.xml`的文件中：

     ```
     <!-- https://mvnrepository.com/artifact/software.amazon.timestream/amazon-
     timestream-jdbc -->
     <dependency>
     ```

```
    <groupId>software.amazon.timestream</groupId>
    <artifactId>amazon-timestream-jdbc</artifactId>
    <version>1.0.1</version>
</dependency>
```

LiveAnalytics JDBC驱动程序类和URL格式的时间流

Timestream 的 LiveAnalytics JDBC驱动程序类为：

```
software.amazon.timestream.jdbc.TimestreamDriver
```

Timestream JDBC 驱动程序需要以下JDBCURL格式：

```
jdbc:timestream:
```

要通过指定数据库属性 JDBCURL，请使用以下URL格式：

```
jdbc:timestream://
```

示例应用程序

为了帮助您开始使用 Timestream for LiveAnalytics w JDBC ith，我们在中 GitHub创建了一个功能齐全的示例应用程序。

1. 按照此处所述的说明创建包含示例数据的数据库。
2. 按照中的说明克隆示例应用程序的 GitHub JDBC存储库GitHub。
3. 按照中的说明README开始使用示例应用程序。

连接属性

LiveAnalytics JDBC驱动程序的 Timestream 支持以下选项：

主题

- 基本身份验证选项
- 标准客户信息选项
- 驱动程序配置选项

- SDK选项
- 端点配置选项
- 凭证提供商选项
- SAML基于 Okta 的身份验证选项
- SAML基于 Azure AD 的身份验证选项

> **Note**
>
> 如果未提供任何属性，则 LiveAnalytics JDBC驱动程序的 Timestream 将使用默认凭证链来加载凭证。

> **Note**
>
> 所有属性键都区分大小写。

基本身份验证选项

下表描述了可用的基本身份验证选项。

选项	描述	默认
AccessKeyId	AWS 用户访问密钥 ID。	NONE
SecretAccessKey	AWS 用户私有访问密钥。	NONE
SessionToken	访问启用了多重身份验证 (MFA) 的数据库所需的临时会话令牌。	NONE

标准客户信息选项

下表描述了 “标准客户机信息” 选项。

选项	描述	默认
ApplicationName	当前使用该连接的应用程序的名称。 ApplicationName 用于调试目的，不会发送给 Timestream 进行 LiveAnalytics服务。	驱动程序检测到的应用程序名称。

驱动程序配置选项

下表描述了驱动程序配置选项。

选项	描述	默认
EnableMetaDataPreparedStatement	启用 Timestream，以便 LiveAnalytics JDBC驱动程序返回其元数据PreparedStatements ，但在检索元数据 LiveAnalytics 时，这会产生额外的 Timestream 费用。	FALSE
区域	数据库的区域。	us-east-1

SDK选项

下表描述了该SDK选项。

选项	描述	默认
RequestTimeout	在超时之前等待查询请求的 AWS SDK时间（以毫秒为单位）。非正值禁用请求超时。	0
SocketTimeout	在超时之前等待通过打开的 AWS SDK连接传输数据的时间（以毫秒为单位）。值必须为	50000

选项	描述	默认
	非负数。值为将0禁用套接字超时。	
MaxRetryCountClient	中有 5XX 错误代码的可重试错误的最大重试次数。SDK该值必须为非负数。	NONE
MaxConnections	允许同时打开的 Timestream 服务HTTP连接的最大数量。LiveAnalytics 该值必须为正数。	50

端点配置选项

下表描述了端点配置选项。

选项	描述	默认
终端节点	LiveAnalytics 服务时间流的终端节点。	NONE

凭证提供商选项

下表描述了可用的凭据提供者选项。

选项	描述	默认
AwsCredentialsProviderClass	`PropertiesFileCredentialsProvider` 或中的一个`InstanceProfileCredentialsProvider` ，用于身份验证。	NONE
CustomCredentialsFilePath	包含 AWS 安全凭证`accessKey` 和的属性文	NONE

选项	描述	默认
	件的路径secretKey 。只有在指定为时AwsCredentialsProviderClass 才需要这样做PropertiesFileCredentialsProvider 。	

SAML基于 Okta 的身份验证选项

下表描述了 Okta 可用的SAML基于身份验证的选项。

选项	描述	默认
IdpName	用于SAML基于身份验证的身份提供者 (Idp) 名称。Okta或之一AzureAD。	NONE
IdpHost	指定 Idp 的主机名。	NONE
IdpUserName	指定 Idp 账户的用户名。	NONE
IdpPassword	指定 Idp 账户的密码。	NONE
OktaApplication身份证	Okta 提供的与应用程序的时间流关联的唯一 ID。LiveAnalytics AppId可以在应用程序元数据中提供的entityID字段中找到。考虑以下示例：entityID = http://www.okta.com//IdpAppID	NONE
角色 ARN	调用者担任的角色的 Amazon 资源名称 (ARN)。	NONE

选项	描述	默认
Idp ARN	中描述 Idp 的SAML提供商的 Amazon 资源名称 (ARN)。IAM	NONE

SAML基于 Azure AD 的身份验证选项

下表介绍了 Azure AD 可用的SAML基于身份验证的选项。

选项	描述	默认
IdpName	用于SAML基于身份验证的身份提供者 (Idp) 名称。`Okta`或之一`AzureAD`。	NONE
IdpHost	指定 Idp 的主机名。	NONE
IdpUserName	指定 Idp 账户的用户名。	NONE
IdpPassword	指定 Idp 账户的密码。	NONE
AADApplicationID	在 Azure AD 上注册的应用程序的唯一 ID。	NONE
AADClientSecret	与 Azure AD 上注册的应用程序关联的客户端密钥，用于授权获取令牌。	NONE
AADTenant	Azure AD 租户 ID。	NONE
Idp ARN	中描述 Idp 的SAML提供商的 Amazon 资源名称 (ARN)。IAM	NONE

JDBCURL例子

本节介绍如何创建JDBC连接URL，并提供示例。要指定可选的连接属性，请使用以下URL格式：

```
jdbc:timestream://PropertyName1=value1;PropertyName2=value2...
```

> **Note**
>
> 所有连接属性都是可选的。所有属性键都区分大小写。

以下是一些JDBC连接示例URLs。

包含基本身份验证选项和区域的示例：

```
jdbc:timestream://
AccessKeyId=<myAccessKeyId>;SecretAccessKey=<mySecretAccessKey>;SessionToken=<mySessionToken
east-1
```

客户信息、地区和SDK选项的示例：

```
jdbc:timestream://ApplicationName=MyApp;Region=us-
east-1;MaxRetryCountClient=10;MaxConnections=5000;RequestTimeout=20000
```

使用在环境变量中设置凭据的默认 AWS 凭证提供程序链进行连接：

```
jdbc:timestream
```

使用在连接中设置了凭据的默认 AWS 凭证提供程序链进行连接：URL

```
jdbc:timestream://
AccessKeyId=<myAccessKeyId>;SecretAccessKey=<mySecretAccessKey>;SessionToken=<mySessionToken
```

使用 PropertiesFileCredentialsProvider 作为身份验证方法的 Connect：

```
jdbc:timestream://
AwsCredentialsProviderClass=PropertiesFileCredentialsProvider;CustomCredentialsFilePath=<pat
 to properties file>
```

使用 InstanceProfileCredentialsProvider 作为身份验证方法的 Connect：

```
jdbc:timestream://AwsCredentialsProviderClass=InstanceProfileCredentialsProvider
```

使用 Okta 凭据作为身份验证方法进行连接：

```
jdbc:timestream://
IdpName=Okta;IdpHost=<host>;IdpUserName=<name>;IdpPassword=<password>;OktaApplicationID=<id>
```

使用 Azure AD 凭据作为身份验证方法进行连接：

```
jdbc:timestream://
IdpName=AzureAD;IdpUserName=<name>;IdpPassword=<password>;AADApplicationID=<id>;AADClientSec
```

使用特定的端点连接：

```
jdbc:timestream://Endpoint=abc.us-east-1.amazonaws.com;Region=us-east-1
```

使用 Okta 设置 Timestream 进行 LiveAnalytics JDBC单点登录身份验证

Timestream fo LiveAnalytics r 支持使用 Okta 进行 LiveAnalytics JDBC单点登录身份验证的 Timestream。要使用 Timestream 对 Okta 进行 LiveAnalytics JDBC单点登录身份验证，请完成下面列出的每个部分。

主题

- 先决条件
- AWS 俄克拉荷马州的账户联盟
- 为 Okta 设置 SAML

先决条件

在使用 Timestream 通过 Okta 进行 LiveAnalyticsJDBC单点登录身份验证之前，请确保满足以下先决条件：

- 中 AWS 创建身份提供商和角色的管理员权限。
- 一个 Okta 账户（https://www.okta.com/login/前往创建账户）。
- 访问亚马逊 Timestream LiveAnalytics

现在，您已经完成了先决条件，可以继续AWS 俄克拉荷马州的账户联盟。

AWS 俄克拉荷马州的账户联盟

LiveAnalytics JDBC驱动程序的 Timestream 支持 Okta 中的 AWS 账户联合。要在 Okta 中设置 AWS 账户联盟，请完成以下步骤：

1. 使用以下URL方式登录 Okta 管理员控制面板：

```
https://<company-domain-name>-admin.okta.com/admin/apps/active
```

> **Note**
>
> 将 < company-domain-name > 替换为您的域名。

2. 成功登录后，选择添加应用程序并搜索AWS 账户联合。
3. 选择 “添加”
4. 将登录名更改URL为相应的URL。
5. 选择 下一步。
6. 选择 SAML2.0 作为登录方法
7. 选择身份提供者元数据以打开元数据XML文件。将该文件保存在本地。
8. 将所有其他配置选项留空。
9. 选择完成。

现在，您已经在 Okta 中完成了 AWS 账户联合，您可以继续。为 Okta 设置 SAML

为 Okta 设置 SAML

1. 选择 Sign On（登录）选项卡。选择视图。
2. 在 “设置” 部分中选择 “设置说明” 按钮。

查找 Okta 元数据文档

1. 要查找文档，请访问：

```
https://<domain>-admin.okta.com/admin/apps/active
```

> **Note**
>
> <domain>是您的 Okta 账户的唯一域名。

2. 选择AWS 账户联合应用程序
3. 选择 “登录” 选项卡

使用 Microsoft Azure AD 为 LiveAnalytics JDBC单点登录身份验证设置时间流

Timestream fo LiveAnalytics r 支持使用 Microsoft Azure AD 进行 LiveAnalytics JDBC单点登录身份验证的时间流。要使用 Timestream 对 Microsoft Azure AD 进行 LiveAnalytics JDBC单点登录身份验证，请完成下面列出的每个部分。

主题

- 先决条件
- 设置 Azure AD
- 在中设置IAM身份提供商和角色 AWS

先决条件

在使用 Timestream 对 Microsoft Azure AD 进行 LiveAnalyticsJDBC单点登录身份验证之前，请确保你已满足以下先决条件：

- 中 AWS 创建身份提供商和角色的管理员权限。
- Azure Active Directory 帐户（ 前往 https://azure.microsoft.com/en-ca/服务/活动目录 /创建帐户 ）
- 访问亚马逊 Timestream LiveAnalytics

设置 Azure AD

1. 登录 Azure 门户
2. 在 Azure 服务列表中选择 Azure 活动目录。这将重定向到 “默认目录” 页面。
3. 在侧边栏的 “管理” 部分下选择 “企业应用程序”
4. 选择 + 新建应用程序。
5. 查找并选择 Amazon Web Services。

6. 在侧边栏的 “管理” 部分下选择 “单点登录”
7. 选择SAML作为单点登录方法
8. 在基本SAML配置部分，URL为标识符和回复输入以下内容URL：

```
https://signin.aws.amazon.com/saml
```

9. 选择保存
10. XML在 “SAML签名证书” 部分下载联合元数据。这将在以后创建IAM身份提供者时使用
11. 返回 “默认目录” 页面，然后在 “管理” 下选择 “应用程序注册”。
12. 从 “所有应用程序” 部分 LiveAnalytics中选择 “时间流”。该页面将被重定向到应用程序的概述页面

> **Note**
>
> 记下应用程序（客户端）ID 和目录（租户）ID。这些值是创建连接时所必需的。

13. 选择 “证书和机密”
14. 在 “客户密钥” 下，使用 + 新客户机密创建新的客户机密钥。

> **Note**
>
> 请注意生成的客户端密钥，因为在创建与 Timestream 的连接时，这是必需的 LiveAnalytics。

15. 在侧边栏的 “管理” 下，选择 “API权限”
16. 在 “已配置的权限” 中，使用添加权限来授予 Azure AD 登录 Timestream 的 LiveAnalytics权限。在 “请求API权限” 页面上选择 Microsoft Graph。
17. 选择 “委派权限”，然后选择 “用户读取” 权限
18. 选择 “添加权限”
19. 为 “默认目录” 选择 “授予管理员同意”

在中设置IAM身份提供商和角色 AWS

完成以下每个部分，IAM为使用 Microsoft Azure AD 进行 LiveAnalytics JDBC单点登录身份验证的 Timestream 进行设置：

主题

- 创建SAML身份提供商
- 创建 IAM 角色
- 创建IAM策略
- 预置

创建SAML身份提供商

要为时间流创建SAML身份提供商，以便使用 Microsoft Azure AD 进行 LiveAnalytics JDBC单点登录身份验证，请完成以下步骤：

1. 登录到 AWS 管理控制台
2. 选择 “服务”，然后在 “安全、身份和合规性” IAM下选择
3. 在 “访问管理” 下选择 “身份提供商”
4. 选择 “创建提供者”，然后选择SAML作为提供程序类型。输入提供商名称。此示例将使用 A zureADProvider。
5. 上传之前下载的联合元数据XML文件
6. 选择 “下一步”，然后选择 “创建”。
7. 完成后，该页面将重定向回身份提供者页面

创建 IAM 角色

要为时间流创建IAM角色以便使用 Microsoft Azure AD 进行 LiveAnalytics JDBC单点登录身份验证，请完成以下步骤：

1. 在侧栏中，选择 “访问管理” 下的 “角色”
2. 选择 Create role
3. 选择 SAML2.0 联合作为可信实体
4. 选择 Azure 广告提供商
5. 选择 “允许编程访问 AWS 和管理控制台”
6. 选择 Next: Permissions。
7. 附加权限策略或继续下一步：标签
8. 添加可选标签或继续下一步：查看
9. 输入角色名称。此示例将使用 A zureSAMLRole
10提供角色描述

11选择 “创建角色” 来完成

创建IAM策略

要为时间流创建使用 Microsoft Azure AD 进行 LiveAnalytics JDBC单点登录身份验证的IAM策略，请完成以下步骤：

1. 在侧栏中，选择 “访问管理” 下的 “策略”
2. 选择创建策略并选择JSON选项卡
3. 添加以下策略

```
{
"Version": "2012-10-17",
"Statement": [
    {
          "Effect": "Allow",
          "Action": [
                "iam:ListRoles",
                "iam:ListAccountAliases"
           ],
           "Resource": "*"
      }
]
}
```

4. 选择 Create policy (创建策略)
5. 输入策略名称。此示例将使用 TimestreamAccessPolicy。
6. 选择 “创建策略”
7. 在侧栏中，选择访问管理下的角色。
8. 选择之前创建的 Azure AD 角色，然后在 “权限” 下选择 “附加策略”。
9. 选择先前创建的访问策略。

预置

要为 Timestream 配置身份提供商，以便使用 Microsoft Azure AD 进行 LiveAnalytics JDBC单点登录身份验证，请完成以下步骤：

1. 返回 Azure 门户

2. 在 Azure 服务列表中选择 Azure 活动目录。这将重定向到 “默认目录” 页面
3. 在侧边栏的 “管理” 部分下选择 “企业应用程序”
4. 选择 “配置”
5. 为置备方法选择自动模式
6. 在 “管理员凭证” 下，输入您的 AwsAccessKeyID 以获取客户端密钥和密钥令SecretAccessKey牌
7. 将置备状态设置为 “开”
8. 选择 “保存”。这允许 Azure AD 加载必要的IAM角色
9. “当前周期” 状态完成后，选择边栏上的 “用户和群组”
10选择 + 添加用户
11选择要为其提供时间流访问权限的 Azure AD 用户 LiveAnalytics
12选择在中创建的 IAM Azure AD 角色和相应的 Azure 身份提供商 AWS
13选择 “分配”

ODBC

Amazon Timestream 的开源ODBC驱动程序为开发人员 LiveAnalytics 提供了与 Timestream 的 SQL关系接口，并支持从 Power BI Desktop 和 Microsoft Excel 等商业智能 (BI) 工具进行连接。LiveAnalytics LiveAnalytics ODBC驱动程序的 Timestream 目前已在 Windows、macOS 和 Linux 上线，还SSO支持 Okta 和微软 Azure Active Directory (AD)。

有关更多信息，请参阅 Amazon Timestream 以获取 LiveAnalyticsODBC驱动程序文档。 GitHub

主题

- 为驱动程序设置时间 LiveAnalytics ODBC流
- ODBC驱动程序的连接字符串语法和选项
- 驱动程序的 Timestream 的 LiveAnalytics ODBC连接字符串示例
- 对与ODBC驱动程序的连接进行故障排除

为驱动程序设置时间 LiveAnalytics ODBC流

在您的 AWS 账户中设置对 Timestream LiveAnalytics 的访问权限

如果您尚未将 AWS 账户设置为使用 Timestream LiveAnalytics，请按照中的说明进行操作。访问 Timestream LiveAnalytics

在您的系统上安装ODBC驱动程序

从ODBC GitHub存储库中为您的系统下载相应的 Timestream ODBC 驱动程序安装程序，然后按照适用于您的系统的安装说明进行操作：。

- Windows 安装指南
- macOS 安装指南
- Linux 安装指南

为ODBC驱动程序设置数据源名称 (DSN)

按照系统DSN配置指南中的说明进行操作：

- Windows DSN 配置
- macOS 配置 DSN
- Linux DSN 配置

设置您的商业智能 (BI) 应用程序以与ODBC驱动程序配合使用

以下是设置几个常用 BI 应用程序以与ODBC驱动程序配合使用的说明：

- 设置微软 Power BI。
- 设置微软 Excel
- 设置 Tableau

对于其他应用程序

ODBC驱动程序的连接字符串语法和选项

为ODBC驱动程序指定连接字符串选项的语法如下：

```
DRIVER={Amazon Timestream ODBC Driver};(option)=(value);
```

可用选项如下：

驱动程序连接选项

- **Driver**（必填）-正在使用的驱动程序ODBC。

默认为 Amazon Timestream。

- **DSN**— 用于配置连接的数据源名称 (DSN)。

  默认为 NONE。

- **Auth**— 身份验证模式。其必须是以下内容之一：
  - AWS_PROFILE— 使用默认凭证链。
  - IAM— 使用 AWS IAM凭证。
  - AAD— 使用 Azure 活动目录 (AD) 身份提供商。
  - OKTA— 使用 Okta 身份提供商。

  默认为 AWS_PROFILE。

端点配置选项

- **EndpointOverride**— LiveAnalytics 服务的 Timestream 的端点覆盖。这是一个高级选项，可以覆盖该区域。例如：

```
query-cell2.timestream.us-east-1.amazonaws.com
```

- **Region**— LiveAnalytics 服务端点时间流的签名区域。

  默认为 us-east-1。

凭证提供者选项

- **ProfileName**— 配置文件中的 AWS 配置文件名称。

  默认为 NONE。

AWS IAM身份验证选项

- **UID**或 **AccessKeyId**- AWS 用户访问密钥 ID。如果连接字符串中同时提供UID和，AccessKeyId则除非该UID值为空，否则将使用该值。

  默认为 NONE。

- **PWD**或 **SecretKey**-AWS 用户私有访问密钥。如果连接字符串中同时提供PWD和，SecretKey则除非PWD值为空，否则将使用带的值。

  默认为 NONE。

- **SessionToken**— 在启用了多重身份验证 (MFA) 的情况下访问数据库所需的临时会话令牌。请勿在输入 = 中包含尾随字符。

  默认为 NONE。

SAML基于 Okta 的身份验证选项

- **IdPHost**— 指定 IdP 的主机名。

  默认为 NONE。

- **UID**或 **IdPUserName**— 指定 IdP 账户的用户名。如果连接字符串中同时提供UID和，IdPUserName则除非该UID值为空，否则将使用该值。

  默认为 NONE。

- **PWD**或 **IdPPassword**— 指定 IdP 帐户的密码。如果连接字符串中同时提供PWD和，IdPPassword则除非该PWD值为空，否则将使用该值。

  默认为 NONE。

- **OktaApplicationID**— Okta 提供的与应用程序的时间流关联的唯一 ID。 LiveAnalytics 在应用程序元数据中提供的entityID字段中可以找到应用程序 ID (AppId)。一个例子是：

  ```
  entityID="http://www.okta.com//(IdPAppID)
  ```

  默认为 NONE。

- **RoleARN**— 调用者担任的角色的 Amazon 资源名称 (ARN)。

  默认为 NONE。

- **IdPARN**— 中描述 IdP 的SAML提供商的 Amazon 资源名称 (ARN)。IAM

  默认为 NONE。

SAML基于 Azure 活动目录的身份验证选项

- **UID**或 **IdPUserName**— 指定 IdP 账户的用户名。

  默认为 NONE。

- **PWD**或 **IdPPassword**— 指定 IdP 帐户的密码。

  默认为 NONE。

- **AADApplicationID**— 在 Azure AD 上注册的应用程序的唯一 ID。

  默认为 NONE。

- **AADClientSecret**— 与 Azure AD 上注册的应用程序关联的客户端密钥，用于授权获取令牌。

  默认为 NONE。

- **AADTenant**— Azure AD 租户 ID。

  默认为 NONE。

- **RoleARN**— 调用者担任的角色的 Amazon 资源名称 (ARN)。

  默认为 NONE。

- **IdPARN**— 中描述 IdP 的SAML提供商的 Amazon 资源名称 (ARN)。IAM

  默认为 NONE。

AWS SDK（高级）选项

- **RequestTimeout**— 在超时之前 AWS SDK等待查询请求的时间（以毫秒为单位）。任何非正值都将禁用请求超时。

  默认为 3000。

- **ConnectionTimeout**— 在超时之前 AWS SDK等待通过打开的连接传输数据的时间（以毫秒为单位）。值为 0 则禁用连接超时。此值不能为负数。

  默认为 1000。

- **MaxRetryCountClient**— 中有 5xx 错误代码的可重试错误的最大重试次数。SDK该值不能为负数。

  默认为 0。

- **MaxConnections**— 允许与 Timestream 服务同时打开的最大HTTP连接数。该值必须为正数。

  默认为 25。

ODBC驱动程序日志选项

- **LogLevel**— 驱动程序日志记录的日志级别。必须为以下值之一：
  - 0 (OFF)。
  - 1 (ERROR)。
  - 2 (WARNING)。
  - (3INFO)。
  - 4 (DEBUG)。

  默认值为 1 (ERROR)。

  警告：使用DEBUG日志模式时，驾驶员可能会记录个人信息。
- **LogOutput**— 用于存储日志文件的文件夹。

  默认值为：
  - Windows:%USERPROFILE%，或者如果不可用，则为%HOMEDRIVE%%HOMEPATH%。
  - macOS 和 Linux：$HOME如果不可用，则为函数getpwuid(getuid())返回pw_dir值中的字段。

SDK日志选项

AWS SDK日志级别与 LiveAnalytics ODBC驱动程序日志级别的时间流是分开的。设置一个不会影响另一个。

SDK日志级别是使用环境变量设置的TS_AWS_LOG_LEVEL。有效值为：

- OFF
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

- FATAL

如果TS_AWS_LOG_LEVEL未设置，则SDK日志级别将设置为默认值，即WARN。

通过代理连接

该ODBC驱动程序支持 LiveAnalytics 通过代理连接到 Amazon Timestream。要使用此功能，请根据您的代理设置配置以下环境变量：

- **TS_PROXY_HOST**— 代理主机。
- **TS_PROXY_PORT**— 代理端口号。
- **TS_PROXY_SCHEME**— 代理方案，要http么是https。
- **TS_PROXY_USER**— 代理身份验证的用户名。
- **TS_PROXY_PASSWORD**— 用于代理身份验证的用户密码。
- **TS_PROXY_SSL_CERT_PATH**— 用于连接HTTPS代理的SSL证书文件。
- **TS_PROXY_SSL_CERT_TYPE**— 代理客户端SSL证书的类型。
- **TS_PROXY_SSL_KEY_PATH**— 用于连接HTTPS代理的私钥文件。
- **TS_PROXY_SSL_KEY_TYPE**— 用于连接HTTPS代理的私钥文件的类型。
- **TS_PROXY_SSL_KEY_PASSWORD**— 用于连接代理的私钥文件的密码。HTTPS

驱动程序的 Timestream 的 LiveAnalytics ODBC连接字符串示例

使用IAM凭证连接到ODBC驱动程序的示例

```
Driver={Amazon Timestream ODBC Driver};Auth=IAM;AccessKeyId=(your access key
 ID);secretKey=(your secret key);SessionToken=(your session token);Region=us-east-2;
```

使用配置文件连接ODBC驾驶员的示例

```
Driver={Amazon Timestream ODBC Driver};ProfileName=(the profile name);region=us-west-2;
```

驱动程序将尝试使用中提供的凭据进行连接~/.aws/credentials，或者如果在环境变量中指定了文件AWS_SHARED_CREDENTIALS_FILE，则使用该文件中的凭据进行连接。

使用 Okta 连接ODBC驱动程序的示例

```
driver={Amazon Timestream ODBC Driver};auth=okta;region=us-west-2;idPHost=(your host at
 Okta);idPUsername=(your user name);idPPassword=(your password);OktaApplicationID=(your
 Okta AppId);roleARN=(your role ARN);idPARN=(your Idp ARN);
```

使用 Azure 活动目录连接到ODBC驱动程序的示例 (AAD)

```
driver={Amazon Timestream ODBC Driver};auth=aad;region=us-west-2;idPUsername=(your
 user name);idPPassword=(your password);aadApplicationID=(your AAD
 AppId);aadClientSecret=(your AAD client secret);aadTenant=(your AAD
 tenant);roleARN=(your role ARN);idPARN=(your idP ARN);
```

使用指定端点和日志级别为 2 (WARNING) 连接到ODBC驱动程序的示例

```
Driver={Amazon Timestream ODBC Driver};Auth=IAM;AccessKeyId=(your access
 key ID);secretKey=(your secret key);EndpointOverride=ingest.timestream.us-
west-2.amazonaws.com;Region=us-east-2;LogLevel=2;
```

对与ODBC驱动程序的连接进行故障排除

> **Note**
>
> 如果已在中指定了用户名和密码DSN，则当ODBC驱动程序管理器要求提供用户名和密码时，无需再次指定它们。

在连接字符串中多次传递连接字符串选项时，`Re-writing (connection string option) (have you specified it several times?`会出现`01S02`带有消息的错误代码。多次指定一个选项会引发错误。使用DSN和连接字符串建立连接时，如果已在中指定了连接选项DSN，则不要在连接字符串中再次指定该选项。

VPC端点 (AWS PrivateLink)

您可以通过创建接口VPC终端节点在您VPC和 Amazon Timestream 之间建立私有连接。 LiveAnalytics 有关更多信息，请参阅 VPC端点 (AWS PrivateLink)。

最佳实践

要充分实现 Amazon Timestream 的优势 LiveAnalytics，请遵循下面描述的最佳实践。

> **Note**
>
> 运行 proof-of-concept应用程序时，在评估 Timestream 的性能和规模时，请考虑您的应用程序在几个月或几年内将积累的数据量。 LiveAnalytics随着数据随时间的推移而增长，您会注意到 Timestream 的性能基本 LiveAnalytics 保持不变，因为其无服务器架构可以利用大量的并行性来处理更大的数据量，并自动扩展以满足应用程序的需求。

主题

数据建模

Amazon Timestream for 旨在收集、存储和分析来自应用程序和设备的时间序列数据，这些数据会发出一系列带有时间戳的数据。 LiveAnalytics 为了获得最佳性能，发送到 Timestream 的数据 LiveAnalytics 必须具有时间特征，并且时间必须是数据的典型组成部分。

Timestream for 允许您 LiveAnalytics 灵活地以不同的方式对数据进行建模，以满足您的应用程序需求。在本节中，我们将介绍其中的几种模式，并为您提供优化成本和绩效的指南。熟悉 Timestream 中有关维度和度量等 LiveAnalytics 概念的关键。在本节中，您将了解有关以下内容的更多信息：

在决定是创建单个表还是创建多个表来存储数据时，请考虑以下几点：

- 将哪些数据放在同一个表中，与何时要将数据分隔到多个表和数据库中。
- 如何在用于 LiveAnalytics 多测量记录的 Timestream 与单测量记录之间进行选择，以及使用多测量记录进行建模的好处，尤其是在您的应用程序同时跟踪多个测量值时。
- 将哪些属性建模为维度或度量。
- 如何有效使用度量名称属性来优化查询延迟。

主题

单桌与多桌

在应用程序中对数据进行建模时，另一个重要方面是如何将数据建模为表和数据库。Timestream 中的数据库和表 LiveAnalytics 是用于访问控制、指定KMS密钥、保留期等的抽象。Timestream 用于LiveAnalytics自动对数据进行分区，旨在扩展资源以匹配应用程序的摄取、存储、查询负载和要求。

Timestream 中的表 LiveAnalytics 可以扩展到存储的 PB 级数据、数十 GB /秒的数据写入，查询每小时可以处理数百个。TBsTimestream 中的查询 LiveAnalytics 可以跨越多个表和数据库，提供联接和并集，让您可以无缝访问多个表和数据库中的数据。因此，在决定如何在 Timestream 中组织数据时，数据规模或请求量通常不是主要考虑的问题。 LiveAnalytics在决定将哪些数据放在同一个表中，哪些数据放在不同的表中，或者放在不同的数据库中的表中，以下是一些重要的注意事项。

- 按表的粒度支持数据保留策略（内存存储保留、磁存储保留等）。因此，需要不同保留策略的数据需要放在不同的表中。
- AWS KMS 用于加密数据的密钥是在数据库级别配置的。因此，不同的加密密钥要求意味着数据需要存放在不同的数据库中。
- Timestream for LiveAnalytics 支持在表和数据库的粒度上进行基于资源的访问控制。在决定将哪些数据写入同一个表，哪些数据写入不同的表时，请考虑您的访问控制要求。
- 在决定将哪些数据存储在哪个表中时，请注意维度数量、度量名称和多度量属性名称的限制。
- 在决定如何组织数据时，请考虑您的查询工作负载和访问模式，因为查询延迟和编写查询的难易程度将取决于此。
  - 如果您将经常查询的数据存储在同一个表中，这通常会简化您编写查询的方式，这样您就可以经常避免编写联接、并集或公用表表达式。这通常还会降低查询延迟。您可以使用维度和度量名称上的谓词来筛选与查询相关的数据。

    例如，假设您存储来自六大洲设备的数据。如果您的查询经常访问来自各大洲的数据以获得全球汇总视图，那么将来自这些大洲的数据存储在同一个表中将使查询更易于编写。另一方面，如果您将数据存储在不同的表上，则仍然可以在同一个查询中合并数据，但是，您将需要编写一个查询来合并来自不同表的数据。

- Timestream for 对您的数据 LiveAnalytics 使用自适应分区和索引。因此，仅对与您的查询相关的数据向查询收费。例如，如果您有一个表存储来自六大洲一百万台设备的数据，如果您的查询中包含形式为`WHERE device_id = 'abcdef'`或的谓词`WHERE continent = 'North America'`，则仅对设备或该大陆的数据进行查询收费。
- 只要有可能，如果您使用度量名称将同一表中不同时发出或不经常查询的数据分开，那么使用诸如`WHERE measure_name = 'cpu'`查询之类的谓词，不仅可以获得计量优势，Timestream for LiveAnalytics 还可以有效地消除查询谓词中没有使用度量名称的分区。这使您能够将具有不同度量名称的相关数据存储在同一个表中，而不会影响查询延迟或成本，并且可以避免将数据分散到多个表中。度量名称本质上用于对数据进行分区和修剪与查询无关的分区。

多度量记录与单度量记录

Timestream for LiveAnalytics 允许您写入每条记录有多个度量（多度量）或每条记录单个度量（单度量）的数据。

多重测量记录

在许多用例中，您正在跟踪的设备或应用程序可能会在同一时间戳发出多个指标或事件。在这种情况下，您可以将同一时间戳发出的所有指标存储在同一个多指标记录中。也就是说，存储在同一多度量记录中的所有度量在同一行数据中显示为不同的列。

例如，假设您的应用程序正在从同时测量的设备发出诸如 cpu、内存、disk_iops 之类的指标。以下是此类表的示例，其中同时发出的多个指标存储在同一行中。你会看到两台主机每秒发出一次指标。

Hostname	measure_name	时间	cpu	内存	disk_iops
host-24gJU	指标	2021-12-01 19:00:00	35	54.9	38.2
host-24gJU	指标	2021-12-01 19:00:01	36	58	39
host-28gJU	指标	2021-12-01 19:00:00	15	55	92
host-28gJU	指标	2021-12-01 19:00:01	16	50	40

单项测量记录

当您的设备在不同的时间段发出不同的指标，或者您使用的是发出metrics/events at different time periods (for instance, when a device's reading/state更改的自定义处理逻辑时，单一测量记录是合适的）。由于每个度量都有唯一的时间戳，因此可以将这些度量存储在 Timestream 中各自的记录中。LiveAnalytics例如，以物联网传感器为例，它可以跟踪土壤温度和湿度，只有在检测到与之前报告的条目相比有变化时，它才会发出记录。以下示例提供了使用单一测量记录发出此类数据的示例。

device_id	measure_name	时间	measure_value::double	measure_value::bigint
sensor-sea478	温度	2021-12-01 19:22:32	35	NULL
sensor-sea478	温度	2021-12-01 18:07:51	36	NULL
sensor-sea478	水汽	2021-12-01 19:05:30	NULL	21
sensor-sea478	水汽	2021-12-01 19:00:01	NULL	23

比较单项测量和多项测量记录

Timestream for 允许您 LiveAnalytics 灵活地根据应用程序的要求和特征将数据建模为单一测量或多度量记录。如果您的应用程序需要的话，单个表可以存储单项测量和多项测量记录。通常，当您的应用程序同时发出多个度量/事件时，通常建议将数据建模为多度量记录，以实现高性能的数据访问和具有成本效益的数据存储。

例如，如果您考虑DevOps 使用一个跟踪来自数十万台服务器的指标和事件的用例，则每台服务器会定期发出 20 个指标和 5 个事件，其中事件和指标会同时即时发出。可以使用单一测量记录或多度量记录对数据进行建模（有关生成的架构，请参阅开源数据生成器）。在此用例中，使用多度量记录与单一测量记录对数据进行建模会得到：

- 摄取计量-多重测量记录可使写入的摄取字节减少大约 40%。
- 摄取批处理-多重测量记录会导致发送的数据批量更大，这意味着客户端需要更少的线程和更少的线程CPU来处理摄取。

- 存储计量-多重测量记录可将存储空间减少约8倍，从而显著节省内存和磁性存储的存储空间。
- 查询延迟-与单度记录相比，多测量记录可降低大多数查询类型的查询延迟。
- Query metered byt es s-对于扫描的数据小于 10MB 的查询，单度记录和多度量记录都是可比较的。对于访问单个度量并扫描大于 10MB 数据的查询，单个度量记录通常会导致计量的字节数较低。对于引用 3 个或更多度量的查询，多度量记录会减少计量的字节数。
- 易于表达多度量查询-当您的查询引用多个度量时，使用多度量记录对数据进行建模可以更轻松地编写更紧凑的查询。

前面的因素会有所不同，具体取决于您要跟踪的指标数量、数据有多少维度等。虽然前面的示例为一个示例提供了一些具体的数据，但我们在许多应用场景和用例中看到，如果您的应用程序在同一时刻发出多个度量，则将数据存储为多度量记录会更有效。此外，多度量记录为您提供了数据类型的灵活性，并将多个其他值存储为上下文（例如，存储请求IDs和其他时间戳，稍后将讨论）。

请注意，多度量记录也可以对稀疏度量进行建模，例如前面的单个度量记录示例：您可以使用 measure_name 来存储度量的名称，并使用通用的多度量属性名称，例如 value_double 来存储度量，value_bigint 用于存储度DOUBLE量，value_timestamp 用于存储BIGINT其他值等。TIMESTAMP

维度和度量

Timestream 中的表格 LiveAnalytics 允许您存储维度（标识要存储的设备/数据的属性）和度量（您正在跟踪的指标/值），有关更多详细信息，请参阅 Time stream 了解概念。 LiveAnalytics 当您在Timestream上为应用程序建模时 LiveAnalytics，如何将数据映射到维度和度量会影响您的摄取和查询延迟。以下是有关如何将数据建模为维度和度量的指南，您可以将其应用于您的用例。

选择尺寸

标识发送时间序列数据的源的数据自然适合维度，维度是不会随着时间的推移而变化的属性。例如，如果您有一台服务器发布指标，则标识服务器的属性（例如主机名、区域、机架、可用区）是维度的候选对象。同样，对于具有多个报告时间序列数据的传感器的物联网设备，设备 ID、传感器 ID 等是维度的候选对象。

如果要将数据写成多度量记录，则当您在表上执行DESCRIBE或运行SELECT语句时，维度和多度量属性会以列形式出现在表中。因此，在编写查询时，可以在同一个查询中自由使用维度和度量。但是，在构建写入记录以摄取数据时，在选择将哪些属性指定为维度以及哪些属性指定为度量值时，请记住以下几点：

- 维度名称、值、度量名称和时间戳唯一标识时间序列数据。Timestream for LiveAnalytics 使用此唯一标识符自动删除重复数据。也就是说，如果 Timestream for LiveAnalytics 接收的两个数据点

具有相同的维度名称、维度值、度量名称和时间戳值，如果这些值具有相同的版本号，则用于重复数据删除的 Timestream。 LiveAnalytics如果新的写入请求的版本低于 Timestream 中已有的数据 LiveAnalytics，则写入请求将被拒绝。如果新的写入请求具有更高的版本，则新值将覆盖旧值。因此，如何选择维度值将影响这种重复数据消除行为。

- 尺寸名称和值无法更新，测量值可以更新。因此，最好将任何可能需要更新的数据建模为测量值。例如，如果您在工厂车间有一台可以改变颜色的机器，则可以将颜色建模为度量值，除非您想将该颜色也用作重复数据删除所需的识别属性。也就是说，测量值可用于存储只会随着时间的推移而缓慢变化的属性。

请注意，Timestream 中的表 LiveAnalytics 不会限制维度名称和值的唯一组合的数量。例如，在一个表中可以存储数十亿个这样的唯一值组合。但是，正如您将在以下示例中看到的那样，谨慎选择维度和度量可以显著优化请求延迟，尤其是查询延迟。

尺寸IDs独特

如果您的应用场景要求您存储每个数据点的唯一标识符（例如，请求 ID、交易 ID 或关联 ID），则将 ID 属性建模为度量值将显著改善查询延迟。使用多度量记录对数据进行建模时，ID 会与您的其他维度和时间序列数据显示在同一行中，因此您的查询可以继续有效地使用它们。例如，考虑到服务器发出的每个数据点都具有唯一的请求 ID 属性的DevOps 用例，与将唯一的请求 ID 建模为维度相比，将请求 ID 建模为度量值可以将不同查询类型的查询延迟降低多达 4 倍。

对于并非每个数据点都完全唯一但具有数十万或数百万个唯一值的属性，您可以使用类似的类比。您可以将这些属性建模为维度值或测量值。如果如前所述，这些值是写入路径上的重复数据消除所必需的，或者您经常在查询中将其用作谓词（例如，在WHERE子句中，在该属性值上有相等谓词，例如您的应用程序跟踪数百万台设备device_id = 'abcde'的位置），则需要将其建模为维度。

数据类型丰富，包含多度量记录

多重测量记录使您可以灵活地对数据进行有效建模。存储在多度量记录中的数据在表格中以列的形式出现，类似于维度，因此查询维度和度量值同样容易。你在前面讨论的示例中看到了其中的一些模式。在下面，您将找到其他模式，以有效使用多指标记录来满足应用程序的用例。

多度量记录支持数据类型的属性DOUBLE、BIGINT、VARCHARBOOLEAN、和。TIMESTAMP因此，它们自然适合不同类型的属性：

- 位置信息：例如，如果您想跟踪位置（以纬度和经度表示），那么与将其存储为VARCHAR维度相比，将其建模为多度量属性可以缩短查询延迟，尤其是在您对纬度和经度有谓词的情况下。

- 记录中的多个时间戳：如果您的应用场景要求您跟踪时间序列记录的多个时间戳，则可以将它们建模为多度量记录中的其他属性。此模式可用于存储带有未来时间戳或过去时间戳的数据。请注意，每条记录仍将使用时间列中的时间戳对记录进行分区、索引和唯一标识。

特别是，如果查询中包含谓词的数字数据或时间戳，则将这些属性建模为多度量属性而不是维度将降低查询延迟。这是因为在使用多度量记录中支持的丰富数据类型对此类数据进行建模时，如果将此类数据建模为维度，则可以使用原生数据类型VARCHAR来表示谓词，而不是将值从其他数据类型转换为其他数据类型。

在多度量记录中使用度量名称

Timestream 中的表 LiveAnalytics 支持名为度量名称的特殊属性（或列）。您可以为写入 Timestream 的每条记录为此 LiveAnalytics属性指定一个值。对于单项测量记录，自然会使用指标的名称（例如 cpu、服务器指标的内存，或者传感器指标的温度、压力）。使用多度量记录时，由于多度量记录中的属性是命名的（这些名称成为表中的列名）、CPU、内存或温度，因此压力可以变成多度量属性名称。因此，一个自然的问题是如何有效地使用度量名称。

Timestream for LiveAnalytics 使用度量名称属性中的值对数据进行分区和索引。因此，如果表有多个不同的度量名称，并且查询使用这些值作为查询谓词，则 Timestream for LiveAnalytics 可以使用其自定义分区和索引来删除与查询无关的数据。例如，如果您的表具有 CPU 和内存度量名称，并且您的查询具有谓词`WHERE measure_name = 'cpu'`，则 Timestream for LiveAnalytics 可以有效地修剪与查询无关的度量名称的数据，例如，本示例中具有度量名称内存的行。即使在多度量记录中使用度量名称，这种修剪也适用。您可以有效地使用度量名称属性作为表的分区属性。度量名称以及维度名称和值以及时间用于对 LiveAnalytics 表的 Timestream 中的数据进行分区。请注意 LiveAnalytics 表的 Timestream 中允许的唯一度量名称数量的限制。另请注意，度量名称也与度量值数据类型相关联，例如，单个度量名称只能与一种测量值类型相关联。该类型可以是`DOUBLE`、、`BIGINT``BOOLEAN``VARCHAR`、和之一`MULTI`。使用度量名称存储的多度量记录的数据类型为`MULTI`。由于单个多指标记录可以存储具有不同数据类型（`DOUBLE`、、`BIGINT``VARCHAR``BOOLEAN`、和`TIMESTAMP`）的多个指标，因此您可以将不同类型的数据关联到多指标记录中。

以下各节介绍几个不同的示例，说明如何有效地使用度量名称属性将不同类型的数据组合到同一个表中。

物联网传感器报告质量和价值

假设您有来自物联网传感器的应用程序监控数据。每个传感器跟踪不同的测量值，例如温度、压力。除了实际值外，传感器还会报告测量质量，这是对读数精度的衡量标准，也是测量的单位。由于质量、单

位和值是一起发射的，因此可以将它们建模为多度量记录，如下面的示例数据所示，其中 device_id 是维度，质量、值和单位是多度量属性：

device_id	measure_name	时间	质量	值	单位
sensor-sea478	温度	2021-12-01 19:22:32	92	35	c
sensor-sea478	温度	2021-12-01 18:07:51	93	34	c
sensor-sea478	pressure	2021-12-01 19:05:30	98	31	psi
sensor-sea478	pressure	2021-12-01 19:00:01	24	132	psi

这种方法允许您将多度量记录的优点与使用度量名称的值对数据进行分区和修剪相结合。如果查询引用单个测量值，例如温度，则可以在查询中包含度量名称谓词。以下是此类查询的示例，该查询还会投影质量高于 90 的测量单位。

```
SELECT device_id, time, value AS temperature, unit
FROM db.table
WHERE time > ago(1h)
    AND measure_name = 'temperature'
    AND quality > 90
```

在查询中使用 measure_name 谓词可以让 Timestream 有效地修剪与查询无关的分区和数据，从而缩短查询延迟。 LiveAnalytics

如果所有指标都是在相同的时间戳发出的，并且/或者在同一个查询中同时查询多个指标，则也可以将所有指标存储在同一个多指标记录中。例如，您可以使用属性温度_质量、温度_值、温度_单位、压力_质量、压力_值、压力_单位等来构造多度量记录。前面讨论的关于使用单度量记录与多度量记录对数据进行建模的许多要点都适用于你决定如何对数据进行建模。请考虑您的查询访问模式以及数据的生成方式，以选择一种能够优化成本、提取和查询延迟以及便于编写查询的模型。

同一个表中包含不同类型的指标

另一个可以将多度量记录与度量名称值组合在一起的用例是对从同一设备独立发出的不同类型的数据进行建模。考虑一下服务器发出两种类型的数据的 DevOps 监控用例：定期发布的指标和不规则的事件。这种方法的一个例子是数据生成器对 DevOps 用例进行建模中讨论的架构。在这种情况下，您可以使用不同的度量名称将同一服务器发出的不同类型的数据存储在同一个表中。例如，同时发出的所有指标都与度量名称指标一起存储。与指标在不同时刻发出的所有事件都与度量名称事件一起存储。表的度量架构（例如，SHOW MEASURES查询的输出）为：

measure_name	data_type	尺寸
events	多	[{“data_type”: “varchar”、“dimension_name”: “availability_zone”}、{“data_type”: “varchar”、“dimension_name”: “instance_type”: “varchar”、“dimension_name”: “instance_name”}、{“data_type”: “varchar”、“varchar”、“dimension_name”: “instance_name”}、{“data_tyd imension_name”: “process_name”}、{“data_type”: “varchar”、“dimension_name”: “jdimension_name”: “jdimen_name”: “jdimen_type”: “jdimension_name”: “region”}, {“data_type”: “region”}, {“data_type”: “dimension_type”: “region”}, {“data_type”: “dimension_“ varchar”、“dimension_name”: “silo”}]
指标	多	[{“data_type”: “varchar”、“dimension_name”: “availability_zone”}、{“data_type”: “varchar”、“dimension_name”: “instance_type”: “varchar”、“dimension_name”:

measure_name	data_type	尺寸
		“instance_name”}、{“data_type”: “varchar”、“varchar”、“dimension_name”: “instance_name”}、{“data_tyd imension_name”: “os_version”} , {“data_type”: “varchar” , “dimension_name”: “cell”} , {“data_type”: “dimensio n_name”: “silo”} , {“data_type”: “varchar” , “dimension_name”: “silo”} , {“data_type”: “varchar” , “dimension_name”: “silo”} , {“data_type”: “varchcha r” , “dimension_name”: “instance_type”}]

在本例中，您可以看到事件和指标也有不同的维度集，其中事件具有不同的维度 jdk_version 和 process_name，而指标具有不同的维度 instance_type 和 os_version。

使用不同的度量名称允许您编写带有谓词的查询`WHERE measure_name = 'metrics'`，例如仅获取指标。此外，将从同一个实例发出的所有数据放在同一个表中意味着你也可以使用 instance_name 谓词编写一个更简单的查询来获取该实例的所有数据。例如，`WHERE instance_name = ‘instance-1234’`没有 measure_name 谓词的形式谓词将返回特定服务器实例的所有数据。

对多度量记录进行分区的建议

> **Important**
>
> 此部分已被弃用！
> 这些建议已经过时。现在，使用客户定义的分区键可以更好地控制分区。

我们已经看到，时间序列生态系统中越来越多的工作负载需要摄取和存储大量数据，同时在通过一组高基数维度值访问数据时需要低延迟的查询响应。

由于这些特性，本节中的建议对于具有以下特性的客户工作负载非常有用。

- 已采用或想要采用多重衡量标准。
- 预计会有大量数据进入系统，这些数据将被长期存储。
- 其主要访问（查询）模式要求低延迟响应时间。
- 要知道，最重要的查询模式涉及谓词中的某种过滤条件。此过滤条件基于高基数维度。例如，考虑按照、、serverID UserId DeviceId、主机名等进行的事件或聚合。

在这些情况下，所有多度量值的单一名称无济于事，因为我们的引擎使用多度量名称对数据进行分区，而使用单个值会限制您获得的分区优势。这些记录的分区主要基于两个维度。假设时间在 x 轴上，维度名称的哈希值在 y 轴measure_name上。measure_name在这些情况下，几乎像分区密钥一样起作用。

我们的建议如下。

- 在针对像我们提到的用例这样的用例对数据进行建模时measure_name，请使用主查询访问模式的直接衍生物。例如：
  - 您的用例需要从最终用户的角度跟踪应用程序性能和 QoE。这也可能是跟踪单个服务器或物联网设备的测量结果。
  - 如果要查询和筛选依 UserId据，则需要在摄取时找到与之关联measure_name的最佳方式。UserId
  - 由于多度量表只能保存 8192 个不同的度量名称，因此无论采用哪种公式，生成的不同值都不应超过 8192 个。
- 我们成功应用于字符串值的一种方法是对字符串值应用哈希算法。然后使用哈希结果的绝对值和 8192 执行模运算。

```
measure_name = getMeasureName(UserId)
int getMeasureName(value) {
    hash_value =  abs(hash(value))
    return hash_value % 8192
}
```

- 我们还添加了abs()删除符号的功能，从而消除了值介于 -8192 到 8192 之间的可能性。这应该在模运算之前执行。
- 通过使用此方法，您的查询的运行时间仅为在未分区的数据模型上运行所需时间的一小部分。
- 查询数据时，请确保在谓词中包含使用新派生的 measure_name 值的筛选条件。例如：
  -
```
SELECT * FROM your_database.your_table
WHERE host_name = 'Host-1235' time BETWEEN '2022-09-01'
```

```
   AND '2022-09-18'
   AND measure_name = (SELECT
cast(abs(from_big_endian_64(xxhash64(CAST('HOST-1235' AS varbinary))))%8192 AS
varchar))
```

- 这将最大限度地减少扫描的分区总数，从而使您的数据随着时间的推移可以更快地转化为查询。

请记住，如果您想从此分区架构中获得好处，则需要在客户端计算哈希值，然后将其作为静态值传递给 Timestream，然后 LiveAnalytics 将其作为静态值传递给查询引擎。前面的示例提供了一种验证引擎是否可以在需要时解析生成的哈希值的方法。

时间	host_name	location	服务器类型	cpu_usage	可用内存	cpu_temp
2022-09-07 21:48:44 .000 0	host-1235	美国东部1	5.8xl	55	16.2	78
R2022-09-07 21:48:44 .000 0	host-3587	美国西部1	5.8xl	62	18.1	81
2022-09-07 21:48:45. 000 000 000 000	host-258743	欧盟中部	5.8xl	88	9.4	91
2022-09-07 21:48:45 .000 0	host-35654	美国东部2	5.8xl	29	24	54
R2022-09-07 21:48:45 .000 0	host-254	美国西部1	5.8xl	44	32	48

要measure_name按照我们的建议生成相关内容，有两条路径取决于您的摄取模式。

1. 用于历史数据的批量摄取-如果您要使用自己的代码进行批处理，则可以将转换添加到编写代码中。

   在前面的示例的基础上构建。

```
        List<String> hosts = new ArrayList<>();

        hosts.add("host-1235");
        hosts.add("host-3587");
        hosts.add("host-258743");
        hosts.add("host-35654");
        hosts.add("host-254");

        for (String h: hosts){
            ByteBuffer buf2 = ByteBuffer.wrap(h.getBytes());
            partition = abs(hasher.hash(buf2, 0L)) % 8192;
            System.out.println(h + " - " + partition);

        }
```

   输出

```
host-1235 - 6445
host-3587 - 6399
host-258743 - 640
host-35654 - 2093
host-254 - 7051
```

   生成的数据集

时间	host_name	location	measure_name	服务器类型	cpu_usage	可用内存	cpu_temp
2022-09-07 21:48:44 .00	host-1235	美国东部 1	6445	5.8xl	55	16.2	78

时间	host_name	location	measure_name	服务器类型	cpu_usage	可用内存	cpu_temp
R2022-09-07 21:48:44 .00	host-3587	美国西部1	6399	5.8xl	62	18.1	81
2022-09-07 21:48:45.000 000 000 000	host-258743	欧盟中部	640	5.8xl	88	9.4	91
2022-09-07 21:48:45 .00	host-35654	美国东部2	2093	5.8xl	29	24	54
R2022-09-07 21:48:45 .00	host-254	美国西部1	7051	5.8xl	44	32	48

2. 用于实时摄取-您需要在数据传measure_name入时生成动态数据。

在这两种情况下，我们都建议您在两端（摄取和查询）测试哈希生成算法，以确保获得相同的结果。

以下是一些生成哈希值的代码示例。host_name

Example Python

```
>>> import xxhash
>>> from bitstring import BitArray
>>> b=xxhash.xxh64('HOST-ID-1235').digest()
>>> BitArray(b).int % 8192
### 3195
```

Example Go

```
package main

import (
    "bytes"
    "fmt"
    "github.com/cespare/xxhash"
)

func main() {
    buf := bytes.NewBufferString("HOST-ID-1235")
    x := xxhash.New()
    x.Write(buf.Bytes())
    // convert unsigned integer to signed integer before taking mod
    fmt.Printf("%f\n", abs(int64(x.Sum64())) % 8192)
}

func abs(x int64) int64 {
    if (x < 0) {
        return -x
    }
    return x
}
```

Example Java

```
import java.nio.ByteBuffer;

import net.jpountz.xxhash.XXHash64;

public class test {
    public static void main(String[] args) {
        XXHash64 hasher = net.jpountz.xxhash.XXHashFactory.fastestInstance().hash64();

        String host = "HOST-ID-1235";
        ByteBuffer buf = ByteBuffer.wrap(host.getBytes());

        Long result = Math.abs(hasher.hash(buf, 0L));
        Long partition = result % 8192;

        System.out.println(result);
        System.out.println(partition);
```

```
    }
}
```

Example Maven 中的依赖关系

```
        <dependency>
            <groupId>net.jpountz.lz4</groupId>
            <artifactId>lz4</artifactId>
            <version>1.3.0</version>
        </dependency>
```

安全性

- 要持续访问的 Timestream LiveAnalytics，请确保加密密钥是安全的，不会被撤销或无法访问。
- 监控来自的API访问日志 AWS CloudTrail。审核并撤销未经授权用户的任何异常访问模式。
- 请遵循中描述的其他指导方针适用于 Amazon Timestream 的安全最佳实践 LiveAnalytics。

将 Amazon Timestream 配置为 LiveAnalytics

为内存存储和磁存储配置数据保留期，以满足数据处理、存储、查询性能和成本要求。

- 设置内存存储的数据保留期，使其符合应用程序处理迟到数据的要求。延迟到达的数据是指时间戳早于当前时间的传入数据。它来自在将数据发送到 Timestream 之前对事件进行批处理的资源 LiveAnalytics，以及间歇性连接的资源（例如间歇性在线的物联网传感器）发出。
- 如果您预计迟到的数据偶尔会带有比内存存储保留时间更早的时间戳，则应为表启用磁性存储写入功能。 MagneticStoreWritesProperties 为表设置输入后，该表将接受时间戳早于内存存储保留期但 EnableMagneticStoreWrites 处于磁性存储保留期内的数据。
- 考虑一下您计划在 Timestream 上运行的查询的特征， LiveAnalytics 例如查询类型、频率、时间范围和性能要求。这是因为内存存储和磁存储针对不同的场景进行了优化。内存存储经过优化，可处理发送到 Timestream 的少量最新数据的快速 point-in-time查询。 LiveAnalytics磁存储器针对快速分析查询进行了优化，这些查询可以处理发送到Timestream的 LiveAnalytics中到大量数据。
- 您的数据保留期限还应受到系统成本要求的影响。

  例如，假设您的应用程序延迟到达的数据阈值为 2 小时，而您的应用程序发送了许多查询，这些查询处理的是一天、一周或数月的数据。在这种情况下，您可能需要为内存存储配置较小的保留期（2-3 小时），并允许更多数据流向磁存储，因为磁存储已针对快速分析查询进行了优化。

了解增加或缩短现有表的内存存储和磁存储的数据保留期的影响。

- 当您缩短内存存储的保留期时，数据将从内存存储移动到磁性存储，并且这种数据传输是永久性的。的 Timestream LiveAnalytics 不会从磁性存储中检索数据来填充内存存储。当您缩短磁存储的保留期时，数据将从系统中删除，并且数据删除是永久性的。
- 当您延长内存存储或磁性存储的保留期时，更改将对 LiveAnalytics 从该时起发送到 Timestream 的数据生效。的 Timestream LiveAnalytics 不会从磁性存储中检索数据来填充内存存储。例如，如果内存存储的保留期最初设置为 2 小时，然后增加到 24 小时，则内存存储将需要 22 小时才能包含价值 24 小时的数据。

写入

- 确保传入数据的时间戳不早于为内存存储配置的数据保留期，也不得晚于中定义的 future 摄取周期。配额 LiveAnalytics 除非您为表启用磁存储写入，否则发送时间戳超出这些范围的数据将导致 Timestream 拒绝这些数据。如果启用磁存储写入，请确保传入数据的时间戳不早于为磁存储配置的数据保留时间。
- 如果您预计数据会延迟到达，请打开表的磁存储写入功能。这将允许摄取时间戳超出内存存储保留期但仍处于磁性存储保留期内的数据。您可以通过更新表格中的`EnableMagneticStoreWrites`标志来`MagneticStoreWritesProperties`进行设置。默认情况下，此属性为 false。请注意，写入磁库的内容不会立即可供查询。它们将在 6 小时内上市。
- 通过确保摄取的数据的时间戳在内存存储保留范围内，将高吞吐量工作负载定位到内存存储。对磁存储的写入仅限于可以同时接收数据库摄取的活动磁存储分区的最大数量。你可以在中看到这个`ActiveMagneticStorePartitions`指标 CloudWatch。为了减少活跃的磁存储分区，目标是减少同时摄入磁存储的序列数量和持续时间。
- 在向 Timestream 发送数据时 LiveAnalytics，在单个请求中批处理多条记录以优化数据提取性能。
  - 将来自相同时间序列的记录和具有相同度量名称的记录进行批处理是有益的。
  - 在单个请求中批量处理尽可能多的记录，前提是这些请求在中定义的服务限制之内配额。
  - 尽可能使用常用属性来降低数据传输和摄取成本。有关更多信息，请参阅 WriteRecords API。
- 如果您在向 Timestream 写入数据时遇到部分客户端故障 LiveAnalytics，则可以在解决拒绝原因后重新发送一批提取失败的记录。
- 按时间戳排序的数据具有更好的写入性能。
- Amazon Timestream 版旨在根据您的应用程序的需求自动进行扩展。 LiveAnalytics 当 Timestream for notices 的应用程序写入请求激增时，您的应用程序可能会遇到某种程度的初始内存存储限制。 LiveAnalytics 如果您的应用程序遇到内存存储限制，请继续以相同（或更高）的速率向

Timestream 发送数据，以使 Timestream 能够自动扩展以满足应用程序的需求。 LiveAnalytics LiveAnalytics 如果你看到磁性存储的节流，你应该降低磁性存储的摄取速率，直到你下降的次数。`ActiveMagneticStorePartitions`

批量加载

中描述了批量加载的最佳实践Batch 加载最佳实践。

查询

以下是使用 Amazon Timestream 进行查询的建议最佳实践。 LiveAnalytics

- 仅包括查询所必需的度量和维度名称。添加多余的列会增加数据扫描量，从而影响查询的性能。
- 在生产环境中部署查询之前，我们建议您查看查询见解，以确保时空修剪效果最佳。有关更多信息，请参阅 使用查询见解来优化 Amazon Timestream 中的查询。
- 在可能的情况下，将数据计算推送到 Timestream，以便在子句和SELECT子句中 LiveAnalytics 使用内置的聚合和标量函数（如果适用），以提高查询性能并WHERE降低成本。请参阅SELECT和聚合函数。
- 如果可能，请使用近似函数。例如，使用 APPROX _ DISTINCT 而不是 COUNT (DISTINCTcolumn_name) 来优化查询性能并降低查询成本。请参阅 聚合函数。
- 使用CASE表达式来执行复杂的聚合，而不必多次从同一个表中进行选择。请参阅 该CASE声明。
- 如果可能，请在查询的WHERE子句中包含一个时间范围。这可以优化查询性能和成本。例如，如果您只需要数据集中最后一小时的数据，则需要添加时间谓词，例如 time > ago (1h)。请参阅SELECT和间隔和持续时间。
- 当查询访问表中的度量子集时，请务必在查询的WHERE子句中包含度量名称。
- 在比较查询WHERE子句中的维度和度量时，请尽可能使用相等运算符。维度和度量名称上的相等谓词可以提高查询性能并降低查询成本。
- 尽可能避免使用WHERE子句中的函数来优化成本。
- 避免多次使用LIKE子句。而是在筛选字符串列上的多个值时使用正则表达式。请参阅 正则表达式函数。
- 仅在查询的 B GROUP Y 子句中使用必要的列。
- 如果查询结果需要ORDER按特定顺序排列，请在最外层查询的 BY 子句中明确指定该顺序。如果您的查询结果不需要排序，请避免使用 BY 子句ORDER来提高查询性能。
- 如果您只需要查询中的前 N 行，请使用LIMIT子句。
- 如果您使用 BY 子句ORDER来查看前或后 N 个值，请使用子LIMIT句来降低查询成本。

- 使用返回的响应中的分页令牌来检索查询结果。有关更多信息，请参阅查询。
- 如果您已开始运行查询，但意识到该查询不会返回您要查找的结果，请取消该查询以节省成本。有关更多信息，请参阅CancelQuery。
- 如果您的应用程序遇到限制，请继续以相同的速率向亚马逊 Timestream LiveAnalytics 发送数据，以使 Amazon Timestream 能够自动扩展 LiveAnalytics 以满足应用程序的查询吞吐量需求。
- 如果您的应用程序的查询并发要求超过了 Timestream 的默认限制 LiveAnalytics，请联系以获取 AWS Support 增加限制。

计划查询

计划查询可通过预先计算一些全队范围的聚合统计数据来帮助您优化仪表板。因此，一个自然而然的问题是，如何处理自己的用例，确定要预先计算哪些结果，以及如何使用存储在派生表中的这些结果来创建仪表板。此过程的第一步是确定要预先计算哪些面板。以下是一些高级指导方针：

- 考虑一下用于填充面板的查询所扫描的字节、仪表板重新加载的频率以及将加载这些仪表板的并发用户数量。您应该从加载频率最高的仪表板开始，然后扫描大量数据。聚合仪表板示例中的前两个仪表板以及向下钻取示例中的聚合仪表板就是此类仪表板的良好示例。
- 考虑重复使用了哪些计算。虽然可以为面板中使用的每个面板和每个变量值创建计划查询，但您可以通过寻找使用一种计算来预先计算多个面板所需的数据的途径，从而显著优化成本和计划查询数量。
- 请考虑计划查询的频率，以刷新派生表中的物化结果。您需要分析仪表板的刷新频率、仪表板中查询的时间窗口、预计算中使用的时间分箱以及仪表板中的面板。例如，如果绘制过去几天每小时汇总数据的仪表板每隔几小时才刷新一次，则您可能需要将计划查询配置为仅每 30 分钟或每小时刷新一次。另一方面，如果您的仪表板可以绘制每分钟聚合数据，并且每分钟左右刷新一次，则您希望计划查询每分钟或几分钟刷新一次结果。
- 考虑使用计划查询可以进一步优化哪些查询模式（从查询成本和查询延迟的角度来看）。例如，在计算仪表板中经常用作变量的唯一维度值时，或者返回传感器发射的最后一个数据点或某个日期之后传感器发出的第一个数据点时，等等。本指南中讨论了其中一些示例模式。

在您移动仪表板以查询派生表时，上述注意事项将对您的节省、仪表板中数据的新鲜度以及计划查询所产生的成本产生重大影响。

客户端应用程序和支持的集成

在与 Timestream 相同的区域运行您的客户端应用程序 LiveAnalytics ，以减少网络延迟和数据传输成本。有关使用其他服务的更多信息，请参阅使用其他服务。以下是其他一些有用的链接。

- 使用 AWS 开发的最佳实践 AWS SDK for Java
- 使用 AWS Lambda 函数的最佳实践
- 适用于 Apache Flink 的亚马逊托管服务的最佳实践
- 在 Grafana 中创建仪表板的最佳实践

常规

- 使用 Timestream 时，请务必遵守 AWS Well-Architected 框架。 LiveAnalytics本白皮书围绕卓越运营、安全性、可靠性、性能效率和成本优化方面的最佳实践提供了指导。

计量和成本优化

使用 Amazon Timestream for LiveAnalytics，您只需为实际用量付费。写入、存储的数据和通过查询扫描的数据的计 LiveAnalytics 量表的时间流。每个计量维度的价格在定价页面上指定。您可以使用亚马逊 Timestream LiveAnalytics 定价计算器估算您的每月账单。

本节介绍如何在 Timestream 中对写入、存储和查询进行 LiveAnalytics计量。还提供了示例场景和计算。此外，还列出了成本优化的最佳实践。你可以在下面选择一个主题：

主题

- 写入
- 存储
- 查询
- 成本优化
- 使用 Amazon 进行监控 CloudWatch

写入

每个时间序列事件的写入大小是按时间戳和一个或多个维度名称、维度值、度量名称和度量值的大小之和计算得出的。时间戳的大小为 8 字节。维度名称、维度值和度量名称的大小等于表示每个维度名称、维度值和度量名称的字符串的 UTF -8 编码字节的长度。度量值的大小取决于数据类型。布尔数据类型为 1 字节，bigint 和 double 为 8 字节，字符串为 UTF -8 编码字节的长度。每次写入以 1 KiB 为单位进行计数。

下面提供了两个计算示例：

主题

计算时间序列事件的写入大小

考虑一个代表EC2实例CPU利用率的时间序列事件，如下所示：

时间	region	az	vpc	Hostname	measure_name	measure_value::double
1602983435238563000	us-east-1	1d	vpc-1a2b3c4d	host-24gJU	CPU_利用率	35.0

时间序列事件的写入大小可以计算为：

- 时间 = 8 字节
- 第一维 = 15 字节 (`region+us-east-1`)
- 第二维 = 4 字节 (`az+1d`)
- 第三维 = 15 字节 (`vpc+vpc-1a2b3c4d`)
- 第四维 = 18 字节 (`hostname+host-24Gju`)
- 度量名称 = 15 字节 (`cpu_utilization`)
- 度量值 = 8 字节

时间序列事件的写入大小 = 83 字节

计算写入次数

现在假设 100 个EC2实例，与中描述的实例类似计算时间序列事件的写入大小，每 5 秒钟发出一次指标。EC2实例的每月总写入量将根据每次写入存在的时间序列事件数量以及批处理时间序列事件时是否使用公共属性而有所不同。以下每种情况都提供了计算每月总写入量的示例：

主题

- 每次写入一个时间序列事件
- 以写入方式批量处理时间序列事件
- 对时间序列事件进行批处理并在写入中使用常用属性

每次写入一个时间序列事件

如果每次写入仅包含一个时间序列事件，则每月总写入量按以下公式计算：

- 100 个时间序列事件 = 每 5 秒 100 次写入
- x 每分钟 12 次写入 = 1,200 次写入
- x 60 分钟/小时 = 72,000 次写入
- x 每天 24 小时 = 1,728,000 次写入
- x 30 天/月 = 51,840,000 次写入

每月写入总数 = 51,840,000

以写入方式批量处理时间序列事件

假设每次写入以 1 KB 为单位进行测量，则一次写入可以包含一批 12 个时间序列事件（998 字节），每月总写入量计算如下：

- 100 个时间序列事件 = 每 5 秒 9 次写入（每次写入 12 个时间序列事件）
- x 每分钟 12 次写入 = 108 次写入
- x 60 分钟/小时 = 6,480 次写入
- x 每天 24 小时 = 155,520 次写入
- x 30 天/月 = 4,665,600 次写入

每月写入总数 = 4,665,600

对时间序列事件进行批处理并在写入中使用常用属性

如果区域、az、vpc 和度量名称在 100 个EC2实例中是通用的，则每次写入只能指定一次常用值，这些常用值被称为常用属性。在这种情况下，公共属性的大小为 52 字节，时间序列事件的大小为 27 字节。假设每次写入以 1 KiB 为单位进行测量，则一次写入可以包含 36 个时间序列事件和公共属性，每月总写入量按以下公式计算：

- 100 个时间序列事件 = 每 5 秒 3 次写入（每次写入 36 个时间序列事件）
- x 每分钟 12 次写入 = 36 次写入
- x 60 分钟/小时 = 2,160 次写入
- x 每天 24 小时 = 51,840 次写入
- x 30 天/月 = 1,555,200 次写入

每月写入总数 = 1,555,200

> **Note**
>
> 由于使用了批处理、常用属性以及将写入操作四舍五入到 1KB 的单位，因此时间序列事件的存储大小可能与写入大小不同。

存储

内存存储和磁存储中每个时间序列事件的存储大小按时间戳、维度名称、维度值、度量名称和度量值的大小之和计算。时间戳的大小为 8 字节。维度名称、维度值和度量名称的大小是代表维度名称、维度值和度量名称的每个字符串的 UTF -8 编码字节的长度。度量值的大小取决于数据类型。布尔数据类型为 1 字节，bigint 和 double 为 8 字节，字符串为 UTF -8 编码字节。每个度量都作为单独的记录存储在 Amazon Timestream 中 LiveAnalytics，也就是说，如果您的时间序列事件有四个度量，则存储的该时间序列事件将有四条记录。

考虑到代表EC2实例CPU利用率的时间序列事件示例（参见计算时间序列事件的写入大小），时间序列事件的存储大小计算公式为：

- 时间 = 8 字节
- 第一维 = 15 字节 (`region+us-east-1`)
- 第二维 = 4 字节 (`az+1d`)
- 第三维 = 15 字节 (`vpc+vpc-1a2b3c4d`)
- 第四维 = 18 字节 (`hostname+host-24Gju`)
- 度量名称 = 15 字节 (`cpu_utilization`)
- 度量值 = 8 字节

时间序列事件的存储大小 = 83 字节

> **Note**
>
> 存储器以 GB 小时计量，磁性存储以 GB /月计量。

查询

查询的费用基于您的应用程序使用的 Timestream 计算单位 (TCUs) 的持续时间（以TCU小时为单位），如亚马逊时间流定价页面上所述。Amazon Timestream for LiveAnalytics “查询引擎” 在处理查询时会删除不相关的数据。使用预测和谓词（包括时间范围、度量名称和/或维度名称）的查询使查询处理引擎能够修剪大量数据，并有助于降低查询成本。

成本优化

要优化写入、存储和查询的成本，请在 Amazon Timestream 中使用以下最佳实践： LiveAnalytics

- 每次写入时批处理多个时间序列事件，以减少写入请求的数量。
- 考虑使用多重测量记录，它允许您在单个写入请求中写入多个时间序列度量，并以更紧凑的方式存储数据。这减少了写入请求的数量以及数据存储成本和查询成本。
- 使用带批处理的常用属性在每次写入时批处理更多时间序列事件，从而进一步减少写入请求的数量。
- 设置内存存储的数据保留期，使其符合应用程序处理迟到数据的要求。延迟到达的数据是指时间戳早于当前时间且在内存存储保留期之外的传入数据。
- 设置磁性存储器的数据保留期以满足您的长期数据存储需求。
- 在编写查询时，请仅包含查询所必需的度量和维度名称。添加多余的列会增加数据扫描量，因此也会增加查询成本。我们建议您查看查询见解，以评估所含维度和度量的修剪效率。
- 如果可能，请在查询的WHERE子句中包含一个时间范围。例如，如果您只需要数据集中最后一小时的数据，请包括一个时间谓词，例如`time > ago(1h)`。
- 当查询访问表中的度量子集时，请务必在查询的WHERE子句中包含度量名称。
- 如果您已开始运行查询，但意识到该查询不会返回您要查找的结果，请取消该查询以节省成本。

使用 Amazon 进行监控 CloudWatch

您可以监控 Timestream 是否 LiveAnalytics 使用亚马逊 CloudWatch，亚马逊收集来自 Timestream 的原始数据并将其处理 LiveAnalytics为可读的指标。 near-real-time这些统计数据会保存两周，以便您能够访问历史信息，并更好地了解 Web 应用程序或服务的执行情况。默认情况下， LiveAnalytics 指标数

据的 Timestream 会 CloudWatch 在 1 分钟或 15 分钟内自动发送到。有关更多信息，请参阅什么是亚马逊 CloudWatch？ 在《亚马逊 CloudWatch 用户指南》中。

主题

- 如何使用 Timestream 来获取 LiveAnalytics 指标？
- LiveAnalytics 指标和维度的时间流
- 创建 CloudWatch 警报以监控 Timestream LiveAnalytics

如何使用 Timestream 来获取 LiveAnalytics 指标？

Timestream 报告的指标 LiveAnalytics 提供了您可以用不同方式进行分析的信息。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。

我如何？	相关指标
How can I determine if any system errors occurred?	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。
How can I monitor the amount of data in the memory store?	您可以在指定的时间段内进行监控MemoryCumulativeBytesMetered ，以监视存储在内存存储中的数据量（以字节为单位）。该指标每小时发出一次，您可以跟踪存储在账户中的字节数，也可以跟踪数据库粒度上存储的字节。内存存储以 GB 小时为单位（存储 1GB 数据一小时的成本）。因此，将每小时价值乘以您所在地区的千兆小时定价，即可得出每小时产生的成本。MemoryCumulativeBytesMetered 维度：操作（存储） DatabaseName、指标名称
How can I monitor the amount of data in the magnetic store?	可以在指定的时间段内进行监控MagneticCumulativeBytesMetered ，以监控存储在磁存储器中的数据量（以字节为单位）。该指标每小时发出一次，您可以跟踪存储在账户中的字节数，也可以跟踪数据库粒度上存储的字节。内存存储按每月 GB（存储 1GB 数据一个月的成本）计量。因此，将每小时价值乘以您所在地区的每月 GB 定价，即可得出每小时产生的成本。MagneticCumulativeBytesMetered 例如，如果的值MagneticCumulativeBytesMetered 为 107374182

我如何？	相关指标
	400 字节 (100GB)，则磁性存储中 1GB 数据的每小时费用 = (0.03)（us-east-1 定价）/(30.4*24)。将该值乘以 GB 为单位将得出该MagneticCumulativeBytesMetered 小时约 0.004 美元。 维度：操作（存储） DatabaseName、指标名称
How can I monitor the data scanned by queries?	您可以在指定的时间段内进行监CumulativeBytesMetered 控，以监控发送到 Timestream 的查询（以字节为 LiveAnalytics单位）扫描的数据。此指标在查询执行后发出，您可以跟踪按账户和数据库粒度扫描的数据。您可以通过将该指标的值乘以您所在地区的每 GB 扫描定价来计算特定时期的查询成本。此指标考虑了计划查询扫描的字节。 维度：操作（查询）、 DatabaseName、指标名称
How can I monitor the data scanned by scheduled queries?	您可以在指定的时间段内进行监控CumulativeBytesMetered ，以监控 Timestream 为执行的计划查询（以字节为 LiveAnalytics单位）扫描的数据。此指标在查询执行后发出，您可以跟踪按账户和数据库粒度扫描的数据。您可以通过将该指标的值乘以您所在地区的每 GB 扫描定价来计算特定时期的查询成本。 **Note** 查询CumulativeBytesMetered 中还考虑了计量的字节。 维度：操作 (TriggeredScheduledQuery) DatabaseName、指标名称

我如何？	相关指标
How can I monitor the number of records ingested?	您可以在指定的时间段内进行监控，以监控摄取的记录数量。NumberOfRecords 您可以跟踪存储在账户中的字节，也可以按数据库粒度进行跟踪。当查询结果写入单独的表时，您还可以使用此指标来监控计划查询所做的写入操作。 使用时 WriteRecords API，会为每个WriteRecords 请求发出指标， CloudWatch 操作维度为WriteRecords 。使用BatchLoad 或时 ScheduledQuery APIs，将按服务确定的间隔发出指标，直到任务完成。此指标的 CloudWatch 操作维度为BatchLoad 或ScheduledQuery ，具体取决于API所使用的维度。 维度：操作（WriteRecords BatchLoad、或 ScheduledQuery）DatabaseName、指标名称
How can I monitor the cost of records ingested?	您可以CumulativeBytesMetered 进行监控以监控所摄取的字节数，从而产生成本。您可以跟踪存储在账户中的字节，也可以按数据库粒度进行跟踪。摄取的记录以累积字节计量。将的CumulativeBytesMetered 值乘以您所在地区的写入定价，即可得出所产生的摄取成本。 使用时 WriteRecords API，会为每个WriteRecords 请求发出此指标， CloudWatch 操作维度为WriteRecords 。使用BatchLoad 或时 ScheduledQuery API，将按服务确定的间隔发出指标，直到任务完成。此指标的 CloudWatch 操作维度是BatchLoad 或ScheduledQuery 取决于所API使用的维度。 维度：操作（WriteRecords BatchLoad、或 ScheduledQuery）DatabaseName、指标名称

我如何？	相关指标
How can I monitor the Timestream Compute Units (TCUs) used in my account?	您可以在指定的时间段内进行监QueryTCU控，以监控账户中用于查询工作负载的计算单位。在账户的活动查询工作负载期间，每分钟都会发出此指标的最大和最小计算单位。 单位：Count 有效统计数据：最小值、最大值 指标：ResourceCount 尺寸：Service: Timestream 、Resource: QueryTCU、Type: Resource、Class: OnDemand

LiveAnalytics 指标和维度的时间流

当您与 Timestream 进行互动时 LiveAnalytics，它会向亚马逊 CloudWatch发送以下指标和维度。所有指标每分钟汇总和报告一次。您可以使用以下过程查看 Timestream 的 LiveAnalytics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/Timestream for LiveAnalytics.。

要查看指标，请使用 AWS CLI

- 在命令提示符处，使用以下命令。

```
aws cloudwatch list-metrics --namespace "AWS/Timestream"
```

指标的时间流维度 LiveAnalytics

Timestream 的 LiveAnalytics 指标由账户、表名或操作的值限定。您可以使用 CloudWatch 控制台检索 Timestream 中下表中任意维度的 LiveAnalytics 数据：

维度	描述
DatabaseName	此维度将数据限制为 LiveAnalytics 数据库的特定时间流。该值可以是当前区域和当前 AWS 账户中的任何数据库
Operation	此维度将 LiveAnalytics 运算（例如、、或ScheduledQuery ）的数据限制在其中一个时间流BatchLoad 内。Storage WriteRecords 有关可用值的列表，请参阅 LiveAnalytics 查询 API参考的时间流。
TableName	此维度将数据限制在 LiveAnalyticss 数据库的时间流中的特定表中。

Important

CumulativeBytesMetered，UserErrors而且SystemErrors指标只有Operation维度。 SuccessfulRequestLatency指标总是有Operation维度，但也可能有DatabaseName和TableName维度，具体取决于的值Operation。这是因为用于 LiveAnalytics表级操作的 Timestream 具有DatabaseName和TableName作为维度，但账户级别的操作没有。

指标的时间 LiveAnalytics 流

Note

Amazon 以一分钟为间 CloudWatch 隔汇总以下所有时间流的 LiveAnalytics 指标。

一般指标

指标	描述
SuccessfulRequestLatency	在指定时间段 LiveAnalytics 内成功向 Timestream 发出的请求。 SuccessfulRequestLatency 可以提供两种不同的信息： • 成功请求所经过的时间（最小值、最大值、总和或平均值）。 • 成功的请求数 (SampleCount)。 SuccessfulRequestLatency 仅反映 Timestream 中的活动，不考虑网络延迟或客户端活动。 LiveAnalytics 单位：Milliseconds 尺寸 • DatabaseName • TableName • Operation 有效统计数据： • Minimum • Maximum • Average • SampleCount • P10 • p50 • p90 • p95 • p99

写入和存储指标

指标	描述
MagneticStoreRejectedRecordCount	被异步拒绝的磁存储写入记录的数量。如果新记录的版本低于当前版本，或者新记录的版本等于当前版本但具有不同的数据，则可能会发生这种情况。 单位：Count 尺寸 • DatabaseName • TableName • Operation 有效统计数据： • Sum • SampleCount
MagneticStoreRejectedUploadUserFailures	由于用户错误而未上传的磁性存储被拒绝的记录报告的数量。这可能是由于IAM权限配置不正确或删除了 S3 存储桶。 单位：Count 尺寸 • DatabaseName • TableName • Operation 有效统计数据： • Sum • SampleCount

指标	描述
MagneticStoreRejectedUpload SystemFailures	由于系统错误而未上传的磁性存储被拒绝的记录报告的数量。 单位：Count 尺寸 • DatabaseName • TableName • Operation 有效统计数据： • Sum • SampleCount
ActiveMagneticStorePartitions	在给定时间主动摄取数据的磁存储分区的数量。 单位：Count 尺寸 • DatabaseName • Operation 有效统计数据： • Sum • SampleCount

指标	描述
MagneticStorePendingRecordsLatency	对不可查询的磁性存储进行的最早写入。写入磁库的记录将在 6 小时内可供查询。 单位：Milliseconds 尺寸 • DatabaseName • TableName • Operation 有效统计数据： • Minimum • Maximum • Average • SampleCount • P10 • p50 • p90 • p95 • p99
MemoryCumulativeBytesMetered	存储在存储器中的数据量，以字节为单位 单位：Bytes 维度：Operation 有效统计数据： • Average

指标	描述
MagneticCumulativeBytesMetered	存储在磁存储中的数据量，以字节为单位 单位：Bytes 维度：Operation 有效统计数据： • Average
CumulativeBytesMetered	通过摄取到 Timestream 计量的数据量，以字节为单位。 LiveAnalytics 单位：Bytes 维度：Operation 有效统计数据：Sum
NumberOfRecords	收录到 Timestream 的记录数。 LiveAnalytics 单位：Count 维度：Operation 有效统计数据：Sum

查询指标

指标	描述
CumulativeBytesMetered	发送到 Timestream 的查询所扫描的数据量 LiveAnalytics，以字节为单位。 单位：Bytes 维度：Operation 有效统计数据：

指标	描述
	• Sum
ResourceCount	账户中用于查询工作负载的时间流计算单位 (TCUs)。在账户的活动查询工作负载期间，每分钟都会发出此指标的最大和最小计算单位。 单位：Count 有效统计数据：最小值、最大值 尺寸：Service: Timestrea m 、Resource: QueryTCU、Type: Resource、Class: OnDemand

错误指标

指标	描述
SystemErrors	为此向 Timestream 发 LiveAnalytics 出的请求会在指定的时间段内生成。 SystemError SystemError 通常表示内部服务错误。 单位：Count 维度：Operation 有效统计数据： • Sum • SampleCount
UserErrors	为此 LiveAnalytics 向 Timestream 发出的请求会在指定的时间段内生成 InvalidRequest 错误。InvalidRequest 通常表示客户端错误，例如参数组合无效、尝试更新不存在的表或请求签名不正确。 UserErrors 表示当前 AWS 区域和当前 AWS 账户的无效请求的总和。

指标	描述
	单位：Count 维度：Operation 有效统计数据： • Sum • SampleCount

> **Important**
>
> 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 LiveAnalytics 控制台的 Timestream 获得，或者使用 CloudWatch 控制台 AWS CLI、或 AWS SDKs用于所有指标。

创建 CloudWatch 警报以监控 Timestream LiveAnalytics

您可以为 Timestream 创建亚马逊 CloudWatch 警报，当警报状态发生变化时 LiveAnalytics ，该警报会发送亚马逊简单通知服务 (AmazonSNS) 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。该操作是向亚马逊SNS主题或 Auto Scaling 策略发送的通知。

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。

故障排除

本节包含有关对 Timestream 进行 LiveAnalytics故障排除的信息。

主题

- 操控 WriteRecords 油门
- 处理被拒绝的记录

- UNLOAD从 Timestream 进行故障排 LiveAnalytics
- LiveAnalytics 特定错误代码的时间流

操控 WriteRecords 油门

随着 Timestream 的扩展以适应应用程序的数据摄取需求，您对 Timestream 的内存存储写入请求可能会受到限制。如果您的应用程序遇到限制异常，则必须继续以相同（或更高）的吞吐量发送数据，以便 Timestream 能够根据应用程序的需求自动扩展。

如果磁存储分区接受摄取的最大限制，则您对 Timestream 的磁存储写入请求可能会受到限制。您将看到一条限制消息，指示您检查此数据库的 `ActiveMagneticStorePartitions` Cloudwatch 指标。此油门最长可能需要 6 小时才能解决。为了避免这种限制，您应该将内存存储用于任何高吞吐量的摄取工作负载。对于磁性存储摄取，您可以通过限制摄取的系列数量和持续时间来将数据摄入到更少的分区中

有关数据摄取最佳实践的更多信息，请参阅。写入

处理被拒绝的记录

如果 Timestream 拒绝记录，您将收到一条`RejectedRecordsException`包含有关拒绝的详细信息。有关如何从 WriteRecords 响应中提取此信息的更多信息，请参阅处理写入失败。

所有拒绝的内容都将包含在此响应中，但磁性存储的更新除外，其中新唱片的版本小于或等于现有唱片的版本。在这种情况下，Timestream 将不会更新版本更高的现有记录。Timestream 将拒绝版本较低或相等的新记录，并将这些错误异步写入您的 S3 存储桶。要接收这些异步错误报告，您应该`MagneticStoreWriteProperties`在表中设置该`MagneticStoreRejectedDataLocation`属性。

UNLOAD从 Timestream 进行故障排 LiveAnalytics

以下是与该UNLOAD命令相关的故障排除指南。

类别	错误消息	如何进行问题排查
S3 密钥长度	UNLOAD使用目标中提供的 S3 前缀 [%s] 时，结果文件密钥将超过 S3 允许的密钥长度。	使用UNLOAD语句导出查询结果时，由 S3 存储桶名称和前缀长度之和组成的 S3 密钥长度超过了支持的最大 S3 密钥长

类别	错误消息	如何进行问题排查
	有关更多详细信息，请参阅文档。	度。我们建议减少前缀或存储桶名称的长度。
	UNLOAD使用 partitioned_by [%s] 时的结果文件密钥将超过 S3 允许的密钥长度。有关更多详细信息，请参阅文档。	使用UNLOAD语句导出查询结果时，使用 partitioned_by 列的 S3 密钥长度超过了支持的最大 S3 密钥长度。我们建议使用备用列进行分区或缩短 partitioned_column 的长度（如果可行）。
	UNLOAD使用 S3 前缀 [%s] 和 partitioned_by [%s] 时的结果文件密钥将超过 S3 允许的密钥长度。有关更多详细信息，请参阅文档。	使用UNLOAD语句导出查询结果时，由 S3 存储桶名称、前缀和 partitioned_by 列名长度之和组成的 S3 密钥长度超过了支持的最大 S3 密钥长度。我们建议减少前缀、存储桶名称长度，或者使用备用列对数据进行分区。
	生成的 S3 对象密钥:%s 太长。有关更多详细信息，请参阅文档。	使用UNLOAD语句处理查询时，分区列中的一个值超过了支持的最大 S3 密钥长度。可以在生成的对象键中找到分区列和值。

类别	错误消息	如何进行问题排查
S3 油门	我们已检测到 Amazon S3 正在限制命令中的UNLOAD写入操作。有关更多信息，请参阅 Amazon Timestream 文档	请参阅此处的 S3 文档。当多个读取器/写入器访问同一个文件夹时，S3 的API调用率可能会受到限制。请审核所提供存储桶的呼叫量。如果您对多个并发UNLOAD查询使用同一个存储桶，请尝试使用不同的存储桶进行相同的查询。如果您将同一个存储桶用于除 Timestream 之外的多个操作 LiveAnalytics UNLOAD，请考虑将UNLOAD结果移动到单独的存储桶。

LiveAnalytics 特定错误代码的时间流

本节包含 Timestream 的 LiveAnalytics特定错误代码。

LiveAnalytics 写入API错误的时间流

InternalServerException

HTTP状态码：500

ThrottlingException

HTTP状态码：429

ValidationException

HTTP状态码：400

ConflictException

HTTP状态码：409

AccessDeniedException

您没有足够的访问权限，无法执行该操作。

HTTP状态码：403

ServiceQuotaExceededException

HTTP状态码：402

ResourceNotFoundException

HTTP状态码：404

RejectedRecordsException

HTTP状态码：419

InvalidEndpointException

HTTP状态码：421

LiveAnalytics 查询API错误的时间流

ValidationException

HTTP状态码：400

QueryExecutionException

HTTP状态码：400

ConflictException

HTTP状态码：409

ThrottlingException

HTTP状态码：429

InternalServerException

HTTP状态码：500

InvalidEndpointException

HTTP状态码：421

配额

本主题介绍了 Amazon Timestream 中的当前配额，也称为限制。 LiveAnalytics除非另行指定，否则每个配额将基于区域应用。

主题

默认限额

下表包含 LiveAnalytics 配额的时间流和默认值。

displayName	描述	defaultValue
每个账户的数据库数	每个数据库可以创建的最大数量 AWS 账户。	500
每个账户中表的数量	每张表可以创建的最大数量 AWS 账户。	50000
的油门速率 CRUD APIs	当前区域中每个账户每秒允许的最大Create/Update/List/Describe/Delete database/table/scheduled查询API请求数。	1
每个账户的计划查询数	您可以为每人创建的最大计划查询数 AWS 账户。	10000
活动磁存储分区的最大数量	每个数据库的最大活动磁存储分区数。接收摄取后，分区可能在长达六小时内保持活动状态。	250

displayName	描述	defaultValue
maxQueryTCU	TCUs您可以为账户设置的最大查询次数。	1000

服务限制

下表包含 LiveAnalytics 服务限制的时间流和默认值。要通过控制台编辑表的数据保留期，请参阅编辑表。

displayName	描述	defaultValue
未来摄入周期（分钟）	与当前系统时间相比，时间序列数据的最长交付周期（以分钟为单位）。例如，如果将来的摄取周期为 15 分钟，则 Timestream for LiveAnalytics 将接受比当前系统时间最多提前 15 分钟的数据。	15
内存存储的最短保留期（小时）	每个表的数据必须保留在内存存储中的最短持续时间（以小时为单位）。	1
内存存储的最长保留期（小时）	每个表的数据可以保留在内存存储中的最长持续时间（以小时为单位）。	8766
磁介质存储的最短保留期（天）	每个表的数据必须保留在磁介质存储中的最短持续时间（以天为单位）。	1
磁介质存储的保留期限（天）	数据可以保留在磁介质存储中的最长持续时间（以天为单位）。该值相当于 200 年。	73000

displayName	描述	defaultValue
磁性存储器的默认保留期（以天为单位）	每个表在磁性存储中保留数据的默认值（以天为单位）。该值相当于 200 年。	73000
内存存储的默认保留期（以小时为单位）	数据在内存存储中保留的默认持续时间（以小时为单位）。	6
每个表的维度数	每个表的最大维度数。	128
每个表的度量名称	每个表的唯一度量名称的最大数量。	8192
每个系列的维度名称维度值对的大小	每个系列的维度名称和维度值对的最大大小。	2KB
最大记录大小	记录的最大大小。	2KB
每个 WriteRecords API请求的记录	WriteRecords API请求中的最大记录数。	100
维度名称长度	维度名称的最大字节数。	60 个字节
度量名称长度	度量名称的最大字节数。	256 字节
数据库名称长度	数据库名称的最大字节数。	256 字节
表名称长度	表名的最大字节数。	256 字节
QueryString 长度（以 kiB 为单位）	查询字符串的最大长度（以 KiB 为单位），以 UTF -8 个编码字符为单位。	256
查询的执行持续时间（小时）	查询的最长执行持续时间（以小时为单位）。耗时过长的查询将超时。	1

displayName	描述	defaultValue
查询见解	当前区域中每个账户每秒启用查询见解时允许的最大查询API请求数。	1
查询结果的元数据大小	查询结果的最大元数据大小。	100KB
查询结果的数据大小	查询结果的最大数据大小。	5GB
每个多重度量记录的度量数	每个多重度量记录的最大度量数。	256
每个多重度量记录的度量值大小	每个多重度量记录的度量值的最大大小。	2048
每个表跨多重度量记录的唯一度量	在单个表中定义的所有多重度量记录中的唯一度量。	1024
每个账户的时间流计算单位 (TCUs)	TCUs每个账户的默认最大值。	200

支持的数据类型

下表描述了度量值和维度值支持的数据类型。

描述	价值的时间 LiveAnalytics 流
度量值支持的数据类型。	大整数、双精度、字符串、布尔值、MULTI、时间戳
维度值支持的数据类型。	String

批量加载

批量加载中的当前配额（也称为限制）如下所示。

描述	价值的时间 LiveAnalytics 流
最大批量加载任务大小	最大批量加载任务大小不能超过 100 GB。
文件数量	一个批量加载任务不能超过 100 个文件。
最大文件大小	批量加载任务中的最大文件大小不能超过 5 GB。
CSV文件行大小	CSV文件中的一行不能超过 16 MB。这是一个硬性限制，无法提高。
活跃的批量加载任务	一个表的活动批量加载任务不能超过 5 个，一个账户的活动批量加载任务不能超过 10 个。Timestream LiveAnalytics 将限制新的批量加载任务，直到有更多资源可用。

命名约束

下表描述了命名限制。

描述	价值的时间 LiveAnalytics 流
维度名称的最大长度。	60 个字节
度量名称的最大长度。	256 字节
表名或数据库名的最大长度。	256 字节
表和数据库名称	• 我们建议您不要使用系统标识符。 • 可以包含 a-z A-Z 0-9 _（下划线）-（破折号）。 （点）。 • 所有名称都必须编码为 UTF -8，并且区分大小写。 **Note** 表名和数据库名使用 UTF -8 二进制表示法进行比较。这意味着ASCII字符的比较区分大小写。
度量名称	• 不得包含系统标识符或冒号 ':'。

描述	价值的时间 LiveAnalytics 流
	• 不得以保留前缀 (`ts_,measure_value`) 开头。 **Note** 表名和数据库名使用 UTF -8 二进制表示法进行比较。这意味着ASCII字符的比较区分大小写。
维度名称	• 不得包含冒系统标识符号 ':' 或双引号 (“)。 • 不得以保留前缀 (`ts_,measure_value`) 开头。 • 不得包含此处列出的 Unicode 字符 [0,31] 或 “\ u2028” 或 “\ u2029”。 **Note** 使用 UTF -8 二进制表示法比较维度和度量名称。这意味着ASCII字符的比较区分大小写。
所有列名	列名不能重复。由于多度量记录将维度和度量表示为列，因此维度的名称不能与度量的名称相同。名称区分大小写。

保留关键字

以下所有内容均为保留关键字：

- ALTER
- AND
- AS
- BETWEEN
- BY
- CASE
- CAST

- CONSTRAINT
- CREATE
- CROSS
- CUBE
- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_USER
- DEALLOCATE
- DELETE
- DESCRIBE
- DISTINCT
- DROP
- ELSE
- END
- ESCAPE
- EXCEPT
- EXECUTE
- EXISTS
- EXTRACT
- FALSE
- FOR
- FROM
- FULL
- GROUP
- GROUPING
- HAVING
- IN

- INNER
- INSERT
- INTERSECT
- INTO
- IS
- JOIN
- LEFT
- LIKE
- LOCALTIME
- LOCALTIMESTAMP
- NATURAL
- NORMALIZE
- NOT
- NULL
- 开
- 或
- ORDER
- OUTER
- PREPARE
- RECURSIVE
- RIGHT
- ROLLUP
- SELECT
- TABLE
- THEN
- TRUE
- UESCAPE
- UNION

- UNNEST
- USING
- VALUES
- WHEN
- WHERE
- WITH

系统标识符

我们将列名 “measure_value”、“ts_non_existent_col” 和 “time” 保留为系统标识符的时间流。LiveAnalytics此外，列名不能以 “ts_” 或 “measure_name” 开头。系统标识符区分大小写。使用 UTF -8 二进制表示进行比较的标识符。这意味着标识符的比较区分大小写。

> **Note**
>
> 系统标识符不能用于维度或度量名称。我们建议您不要使用系统标识符作为数据库或表名。

UNLOAD

有关该UNLOAD命令的限制，请参阅使用将查询结果从 Timestream 导出UNLOAD到 S3。

查询语言参考

> **Note**
>
> 本查询语言参考包括来自天合软件基金会（前身为Presto软件基金会）的以下第三方文档，该文档根据Apache许可2.0版获得许可。除非遵守本许可，否则不得使用此文件。要获取 Apache 许可证 2.0 版的副本，请访问 Apache 网站。

Timestream for LiveAnalytics 支持一种用于处理数据的丰富查询语言。你可以在下面看到可用的数据类型、运算符、函数和结构。

您还可以在本示例查询节中立即开始使用 Timestream 的查询语言。

主题

支持的数据类型

Timestream LiveAnalytics 的查询语言支持以下数据类型。

> **Note**
>
> 数据类型中描述了支持写入的数据类型。

数据类型	描述
`int`	表示 32 位整数。
`bigint`	表示 64 位有符号整数。
`boolean`	逻辑的两个真值之一，`True`和`False`。
`double`	表示 64 位可变精度数据类型。实现二进制浮点运算的IEEE标准754。 **Note** 查询语言用于读取数据。有函数`Infinity`和`NaN`双精度值可以在查询中使用。但是你不能将这些值写入Timestream。
`varchar`	可变长度的字符数据，最大大小为 2KB。
`array[T,...]`	包含一个或多个指定数据类型的元素 *T*其中，*T* 可以是 Timestream 中支持的任何一种数据类型。
`row(T,...)`	包含一个或多个数据类型的命名字段 *T*。 这些字段可以是 Timestream 支持的任何数据类型，可使用点字段引用运算符进行访问： `.`
`date`	表示表单中的日期*YYYY*-*MM*-*DD*。其中 *YYYY* 是年份，*MM* 是月份，而且 *DD* 分别是当天。支持的范围是从`1970-01-01` 到`2262-04-11` 。 示例： `1971-02-03`
`time`	表示一天中的时间UTC。`time`数据类型以“*HH*.*MM*.*SS*.*sssssssss* .支持纳秒精度” 的形式表示。

数据类型	描述
	示例： `17:02:07.496000000`
`timestamp`	使用纳秒精度 time in 表示时间中的UTC实例。 *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.*sssssssss* Query 支持的时间戳范围在 to 之间1677-09-21 00:12:44.000000000 。2262-04-11 23:47:16.854775807

数据类型	描述

interval	将时间间隔表示为字符串文字*X**t*，由两部分组成，*X* 以及 *t*.
	
	X 是大于或等于的数值0，并且 *t* 是一个时间单位，如秒或小时。单位不是复数的。时间单位 *t* is 必须是以下字符串文字之一：
	
	• nanosecond
	• microsecond
	• millisecond
	• second
	• minute
	• hour
	• day
	• ns（与nanosecond ）
	• us（与microsecond ）
	• ms（与millisecond ）
	• s（与second）
	• m（与minute）
	• h（与hour）
	• d（与day）
	
	示例：
	
	`17s`
	
	`12second`
	
	`21hour`
	
	`2d`

数据类型	描述
timeseries[row(timestamp, *T*,...)]	表示在一段时间间隔内以array由row对象组成的形式记录的测量值。每个都row包含一个timestamp 和一个或多个数据类型的度量值 *T*其中，*T* 可以是bigint、booleandouble、或中的任何一个varchar。行按升序排列。timestamp 时间序列数据类型表示一段时间内的度量值。
unknown	表示空数据。

内置时间序列功能

Timestream for LiveAnalytics 提供了内置的时间序列功能，可将时间序列数据视为一流的概念。

内置的时间序列功能可以分为两类：视图和函数。

您可以在下面阅读有关每个构造的信息。

主题

- 时间序列视图
- 时间序列函数

时间序列视图

Timestream for LiveAnalytics 支持以下用于将数据转换为timeseries数据类型的函数：

主题

- CREATE_TIME_SERIES
- UNNEST

CREATE_TIME_SERIES

CREATE_ TIME _ SERIES 是一个聚合函数，它获取时间序列的所有原始测量值（时间和度量值），并返回时间序列数据类型。此函数的语法如下：

```
CREATE_TIME_SERIES(time, measure_value::<data_type>)
```

其中，<data_type>是度量值的数据类型，可以是 bigint、boolean、double 或 varchar 之一。第二个参数不能为空。

考虑存储在名为 m etrics 的表中的EC2实例的CPU利用率，如下所示：

时间	region	az	vpc	instance_id	measure_name	measure_value::double
2019-12-04 19:00:00.000 000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	CPU_利用率	35.0
2019-12-04 19:00:01.000 000000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	CPU_利用率	38.2
2019-12-04 19:00:02.000 000000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	CPU_利用率	45.3
2019-12-04 19:00:00.000 000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	CPU_利用率	54.1
2019-12-04 19:00:01.000 000000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	CPU_利用率	42.5
2019-12-04 19:00:02.000 000000 000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	CPU_利用率	33.7

运行查询：

```
SELECT region, az, vpc, instance_id, CREATE_TIME_SERIES(time, measure_value::double) as
 cpu_utilization FROM metrics
    WHERE measure_name='cpu_utilization'
    GROUP BY region, az, vpc, instance_id
```

将返回所有以度量值cpu_utilization为度量值的系列。在本例中，我们有两个系列：

region	az	vpc	instance_id	CPU_利用率
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567 890abcdef0	[{时间：2019 -12-04 19:00:00. 000 00000，mea sure_value:: double: 35.0}， {时间：2019-12- 04 19:00:01. 000 000 000 000，measu re_value:: double: 45.3}， {时间：2019-12- 04 19:00:02.000 000 000，measu re_value:: 45.3}]
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567 890abcdef1	[{时间：2019 -12-04 19:00:00. 000 00000，mea sure_value:: double: 35.1}， {时间：2019-12- 04 19:00:01. 000 000 000 000，measu re_value:: 45.7}，{时间

region	az	vpc	instance_id	CPU_利用率
				：2019-12-04 19:00:02.000 000 000，measure_value:: 45.7}]

UNNEST

UNNEST是一个表格函数，可让您将timeseries数据转换为平面模型。语法如下：

UNNESTtimeseries将 a 转换为两列，即、time和。value您也可以使用别名，UNNEST如下所示：

```
UNNEST(timeseries) AS <alias_name> (time_alias, value_alias)
```

其中，<alias_name>是平面表的别名，time_alias是time列的别名，value_alias是value列的别名。

例如，假设队列中的一些EC2实例配置为以 5 秒的间隔发布指标，而另一些实例则以 15 秒的间隔发布指标，并且您需要在过去 6 小时内以 10 秒为粒度发布所有实例的平均指标。要获取这些数据，请使用 CREATETIME_ 将指标转换为时间序列模型SERIES。然后，你可以使用 INTERPOLATE_ LINEAR 以 10 秒的粒度获取缺失值。接下来，使用将数据转换回平面模型 UNNEST，然后使用AVG来获取所有实例的平均指标。

```
WITH interpolated_timeseries AS (
    SELECT region, az, vpc, instance_id,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(time, measure_value::double),
                SEQUENCE(ago(6h), now(), 10s)) AS interpolated_cpu_utilization
    FROM timestreamdb.metrics
    WHERE measure_name= 'cpu_utilization' AND time >= ago(6h)
    GROUP BY region, az, vpc, instance_id
)
SELECT region, az, vpc, instance_id, avg(t.cpu_util)
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_cpu_utilization) AS t (time, cpu_util)
GROUP BY region, az, vpc, instance_id
```

上面的查询演示了UNNEST使用别名的方法。以下是未使用别名的相同查询的示例 UNNEST：

```
WITH interpolated_timeseries AS (
    SELECT region, az, vpc, instance_id,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(time, measure_value::double),
                SEQUENCE(ago(6h), now(), 10s)) AS interpolated_cpu_utilization
    FROM timestreamdb.metrics
    WHERE measure_name= 'cpu_utilization' AND time >= ago(6h)
    GROUP BY region, az, vpc, instance_id
)
SELECT region, az, vpc, instance_id, avg(value)
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_cpu_utilization)
GROUP BY region, az, vpc, instance_id
```

时间序列函数

Amazon Timestream for LiveAnalytics 支持时间序列函数，例如导数、积分和相关性以及其他函数，以便从您的时间序列数据中获得更深入的见解。本节提供每个函数的用法信息以及示例查询。在下面选择一个主题以了解更多信息。

主题

- 插值函数
- 导数函数
- 积分函数
- 关联函数
- 筛选和缩小功能

插值函数

如果您的时间序列数据缺少特定时间点的事件值，则可以使用插值法估计这些缺失事件的值。Amazon Timestream 支持四种插值变体：线性插值、三次样条插值、上次观测向前移动 (locf) 插值法和恒定插值。本节提供 LiveAnalytics插值函数的 Timestream 的用法信息以及示例查询。

使用情况信息

函数	输出数据类型	描述
`interpolate_linear (timeseries, array[timestamp])`	时间序列	使用线性插值法填充缺失的数据。
`interpolate_linear (timeseries, timestamp)`	double	使用线性插值法填充缺失的数据。
`interpolate_spline _cubic(timeseries, array[timestamp])`	时间序列	使用三次样条插值法填充缺失的数据。
`interpolate_spline _cubic(timeseries, timestamp)`	double	使用三次样条插值法填充缺失的数据。
`interpolate_locf(t imeseries, array[tim estamp])`	时间序列	使用上次采样值填充缺失的数据。
`interpolate_locf(t imeseries, timestamp )`	double	使用上次采样值填充缺失的数据。
`interpolate_fill(t imeseries, array[tim estamp], double)`	时间序列	使用常量值填充缺失的数据。
`interpolate_fill(t imeseries, timestamp , double)`	double	使用常量值填充缺失的数据。

查询示例

Example

找出过去 2 小时内特定EC2主机以 30 秒为间隔分箱的平均CPU利用率，使用线性插值填充缺失值：

```
WITH binned_timeseries AS (
SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double),
 2) AS avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
SELECT hostname,
    INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
            SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
FROM binned_timeseries
GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

Example

找出过去 2 小时内特定EC2主机以 30 秒为间隔分箱的平均CPU利用率，并根据上次执行的观测值使用插值填充缺失值：

```
WITH binned_timeseries AS (
SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double),
 2) AS avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
SELECT hostname,
    INTERPOLATE_LOCF(
```

```
        CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
            SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
FROM binned_timeseries
GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

导数函数

导数用于计算给定指标的变化率，并可用于主动响应事件。例如，假设您计算了过去 5 分钟内EC2实例CPU利用率的导数，并且发现了一个显著的正导数。这可能表明对工作负载的需求有所增加，因此您可以决定启动更多EC2实例以更好地处理工作负载。

Amazon Timestream 支持导数函数的两种变体。本节提供 LiveAnalytics 导数函数的时间流的用法信息以及示例查询。

使用情况信息

函数	输出数据类型	描述
`derivative_linear(timeseries, interval)`	时间序列	计算指定点中每个点`timeseries` 的导数`interval`。
`non_negative_derivative_linear(timeseries, interval)`	时间序列	与相同`derivative_linear(timeseries, interval)` ，但仅返回正值。

查询示例

Example

找出过去 1 小时内每 5 分钟CPU利用率的变化率：

```
SELECT DERIVATIVE_LINEAR(CREATE_TIME_SERIES(time, measure_value::double), 5m) AS
 result
```

```
FROM “sampleDB”.DevOps
WHERE measure_name = 'cpu_utilization'
AND hostname = 'host-Hovjv' and time > ago(1h)
GROUP BY hostname, measure_name
```

Example

计算一个或多个微服务产生的错误增加率：

```
WITH binned_view as (
    SELECT bin(time, 5m) as binned_timestamp, ROUND(AVG(measure_value::double), 2) as
 value
    FROM “sampleDB”.DevOps
    WHERE micro_service = 'jwt'
    AND time > ago(1h)
    AND measure_name = 'service_error'
    GROUP BY bin(time, 5m)
)
SELECT non_negative_derivative_linear(CREATE_TIME_SERIES(binned_timestamp, value), 1m)
 as rateOfErrorIncrease
FROM binned_view
```

积分函数

您可以使用积分来查找时间序列事件每单位时间曲线下方的区域。例如，假设您正在跟踪应用程序每单位时间内收到的请求量。在这种情况下，您可以使用积分函数来确定在特定时间段内按指定间隔处理的请求总量。

Amazon Timestream 支持积分函数的一种变体。本节提供 LiveAnalytics 积分函数的 Timestream 的用法信息以及示例查询。

使用情况信息

函数	输出数据类型	描述

`integral_trapezoidal(timeseries(double))`
 	
`integral_trapezoidal(timeseries(doub`
	double	使用梯形法则根据`interval day to second`为`timeseries` 提供的指定值近似积分。日到秒的时间间隔参数是可选的，默认

函数	输出数据类型	描述
le), interval day to second) integral_trapezoidal(timeseries(bigint)) integral_trapezoidal(timeseries(bigint), interval day to second) integral_trapezoidal(timeseries(integer), interval day to second) integral_trapezoidal(timeseries(integer))		值为1s。有关间隔的更多信息，请参见间隔和持续时间。

查询示例

Example

计算过去一小时内特定主机每五分钟处理的请求总量：

```
SELECT INTEGRAL_TRAPEZOIDAL(CREATE_TIME_SERIES(time, measure_value::double), 5m) AS
 result FROM sample.DevOps
WHERE measure_name = 'request'
AND hostname = 'host-Hovjv'
AND time > ago (1h)
GROUP BY hostname, measure_name
```

关联函数

给定两个长度相似的时间序列，相关函数提供了一个相关系数，该系数解释了两个时间序列如何随时间推移而变化。相关系数的范围-1.0为到1.0。 -1.0表示两个时间序列以相同的速度朝相反的方向发

展。而1.0表示两个时间序列以相同的速率朝同一个方向发展。值为0表示两个时间序列之间没有相关性。例如，如果石油价格上涨，石油公司的股票价格上涨，那么石油价格上涨的趋势和石油公司的价格上涨将具有正相关系数。高正相关系数表明两个价格的走势相似。同样，债券价格和债券收益率之间的相关系数为负，这表明随着时间的推移，这两个值的趋势相反。

Amazon Timestream 支持关联函数的两种变体。本节提供 LiveAnalytics 关联函数的时间流的用法信息以及示例查询。

使用情况信息

函数	输出数据类型	描述
`correlate_pearson(timeseries, timeseries)`	double	计算两`timeseries` 者的 Pearson 相关系数。时间序列必须具有相同的时间戳。
`correlate_spearman(timeseries, timeseries)`	double	计算两`timeseries` 者的 Spearman 相关系数。时间序列必须具有相同的时间戳。

查询示例

Example

```
WITH cte_1 AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, measure_value::double),
        SEQUENCE(min(time), max(time), 10m)) AS result
    FROM sample.DevOps
    WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv' AND time > ago(1h)
    GROUP BY hostname, measure_name
),
cte_2 AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, measure_value::double),
        SEQUENCE(min(time), max(time), 10m)) AS result
    FROM sample.DevOps
    WHERE measure_name = 'cpu_utilization'
```

```
    AND hostname = 'host-Hovjv' AND time > ago(1h)
    GROUP BY hostname, measure_name
)
SELECT correlate_pearson(cte_1.result, cte_2.result) AS result
FROM cte_1, cte_2
```

筛选和缩小功能

Amazon Timestream 支持对时间序列数据执行筛选和减少操作的功能。本节提供 LiveAnalytics 筛选和缩减函数的时间流的用法信息，以及示例查询。

使用情况信息

函数	输出数据类型	描述
`filter(timeseries(T), function(T, Boolean))`	时间序列 (T)	根据输入的时间序列构造一个时间序列，使用传递的`function`返回`true`值。
`reduce(timeseries(T), initialState S, inputFunction(S, T, S), outputFunction(S, R))`	R	返回从时间序列中减去的单个值。`inputFunction` 将按顺序对时间序列中的每个元素调用。除了获取当前元素外，还 inputFunction 会获取当前状态（最初`initialState` ）并返回新状态。`outputFunction` 将调用以将最终状态转换为结果值。`outputFunction` 可以是标识函数。

查询示例

Example

构造测量值大于 70 的主机和过滤点CPU利用率的时间序列：

```
WITH time_series_view AS (
```

```
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT FILTER(cpu_user, x -> x.value > 70.0) AS cpu_above_threshold
from time_series_view
```

Example

构造主机CPU利用率的时间序列并确定测量值的总和平方：

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT REDUCE(cpu_user,
    DOUBLE '0.0',
    (s, x) -> x.value * x.value + s,
    s -> s)
from time_series_view
```

Example

构造主机CPU利用率的时间序列，并确定超过CPU阈值的样本比例：

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
```

```
SELECT ROUND(
    REDUCE(cpu_user,
      -- initial state
      CAST(ROW(0, 0) AS ROW(count_high BIGINT, count_total BIGINT)),
      -- function to count the total points and points above a certain threshold
      (s, x) -> CAST(ROW(s.count_high + IF(x.value > 70.0, 1, 0), s.count_total + 1) AS
 ROW(count_high BIGINT, count_total BIGINT)),
      -- output function converting the counts to fraction above threshold
      s -> IF(s.count_total = 0, NULL, CAST(s.count_high AS DOUBLE) / s.count_total)),
    4) AS fraction_cpu_above_threshold
from time_series_view
```

SQL支持

Timestream fo LiveAnalytics r 支持一些常见的SQL结构。你可以在下面阅读更多内容。

主题

- SELECT
- 子查询支持
- SHOW声明
- DESCRIBE声明
- UNLOAD

SELECT

SELECT语句可用于从一个或多个表中检索数据。Timestream 的查询语言支持以下语SELECT句语法：

```
[ WITH with_query [, ...] ]
            SELECT [ ALL | DISTINCT ] select_expr [, ...]
            [ function (expression) OVER (
            [ PARTITION BY partition_expr_list ]
            [ ORDER BY order_list ]
            [ frame_clause ] )
            [ FROM from_item [, ...] ]
            [ WHERE condition ]
            [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
            [ HAVING condition]
            [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
```

```
            [ ORDER BY order_list ]
            [ LIMIT [ count | ALL ] ]
```

其中

- function (expression)是支持的窗口功能之一。
- partition_expr_list 为：

```
expression | column_name [, expr_list ]
```

- order_list 为：

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

- frame_clause 为：

```
ROWS | RANGE
{ UNBOUNDED PRECEDING | expression PRECEDING | CURRENT ROW } |
{BETWEEN
{ UNBOUNDED PRECEDING | expression { PRECEDING | FOLLOWING } |
CURRENT ROW}
AND
{ UNBOUNDED FOLLOWING | expression { PRECEDING | FOLLOWING } |
CURRENT ROW }}
```

- from_item是以下之一：

```
table_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
from_item join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]
```

- join_type是以下之一：

```
[ INNER ] JOIN
LEFT [ OUTER ] JOIN
RIGHT [ OUTER ] JOIN
FULL [ OUTER ] JOIN
```

- grouping_element是以下之一：

```
()
```

```
expression
```

子查询支持

Timestream 支持中的子查询EXISTS和谓词。INEXISTS谓词决定子查询是否返回任何行。IN谓词确定子查询生成的值是否与 IN 子句中的值或表达式相匹配。Timestream 查询语言支持关联子查询和其他子查询。

```
SELECT t.c1
FROM (VALUES 1, 2, 3, 4, 5) AS t(c1)
WHERE EXISTS
(SELECT t.c2
 FROM (VALUES 1, 2, 3) AS t(c2)
 WHERE t.c1= t.c2
)
ORDER BY t.c1
```

c1
1
2
3

```
SELECT t.c1
FROM (VALUES 1, 2, 3, 4, 5) AS t(c1)
WHERE t.c1 IN
(SELECT t.c2
 FROM (VALUES 2, 3, 4) AS t(c2)
)
ORDER BY t.c1
```

c1
2
3

c1
4

SHOW声明

您可以使用该SHOW DATABASES语句查看一个账户中的所有数据库。语法如下：

```
SHOW DATABASES [LIKE pattern]
```

其中子LIKE句可用于筛选数据库名称。

您可以使用对账单查看账户中的所有SHOW TABLES表。语法如下：

```
SHOW TABLES [FROM database] [LIKE pattern]
```

其中，FROM子句可用于筛选数据库名称，LIKE子句可用于筛选表名。

您可以使用SHOW MEASURES语句查看表的所有度量。语法如下：

```
SHOW MEASURES FROM database.table [LIKE pattern]
```

其中，FROM子句将用于指定数据库和表名，该LIKE子句可用于筛选度量名称。

DESCRIBE声明

您可以使用DESCRIBE语句查看表的元数据。语法如下：

```
DESCRIBE database.table
```

其中table包含表名。describe 语句返回表的列名和数据类型。

UNLOAD

Timestream for LiveAnalytics 支持将UNLOAD命令作为其SQL支持的扩展。中描述了UNLOAD支持的数据类型支持的数据类型。time和unknown类型不适用于UNLOAD。

```
UNLOAD (SELECT statement)
 TO 's3://bucket-name/folder'
 WITH ( option = expression [, ...] )
```

选项在哪里

```
{ partitioned_by = ARRAY[ col_name[,…] ]
 | format = [ '{ CSV | PARQUET }' ]
 | compression = [ '{ GZIP | NONE }' ]
 | encryption = [ '{ SSE_KMS | SSE_S3 }' ]
 | kms_key = '<string>'
 | field_delimiter ='<character>'
 | escaped_by = '<character>'
 | include_header = ['{true, false}']
 | max_file_size = '<value>'
}
```

SELECT声明

用于从一个或多个 Timestream 中为 LiveAnalytics 表选择和检索数据的查询语句。

```
(SELECT column 1, column 2, column 3 from database.table
      where measure_name = "ABC" and timestamp between ago (1d) and now() )
```

收件人条款

```
TO 's3://bucket-name/folder'
```

或者

```
TO 's3://access-point-alias/folder'
```

语TO句中的子UNLOAD句指定查询结果输出的目的地。您需要提供完整路径，包括 Amazon S3 存储桶名称或 Amazon S3，以及在 Amazon S3 access-point-alias 上 LiveAnalytics写入输出文件对象的 Timestream 上的文件夹位置。S3 存储桶应归同一账户所有，且位于同一区域。除了查询结果集之外，Timestream 还会将清单和元数据文件 LiveAnalytics 写入指定的目标文件夹。

PARTITIONED_BY 子句

```
partitioned_by = ARRAY [col_name[,…] , (default: none)
```

该partitioned_by子句用于查询中，用于对数据进行精细分组和分析。将查询结果导出到 S3 存储桶时，您可以选择根据选择查询中的一列或多列对数据进行分区。对数据进行分区时，会根据分区列将导出的数据分成子集，每个子集存储在单独的文件夹中。在包含导出数据的结果文件夹中，

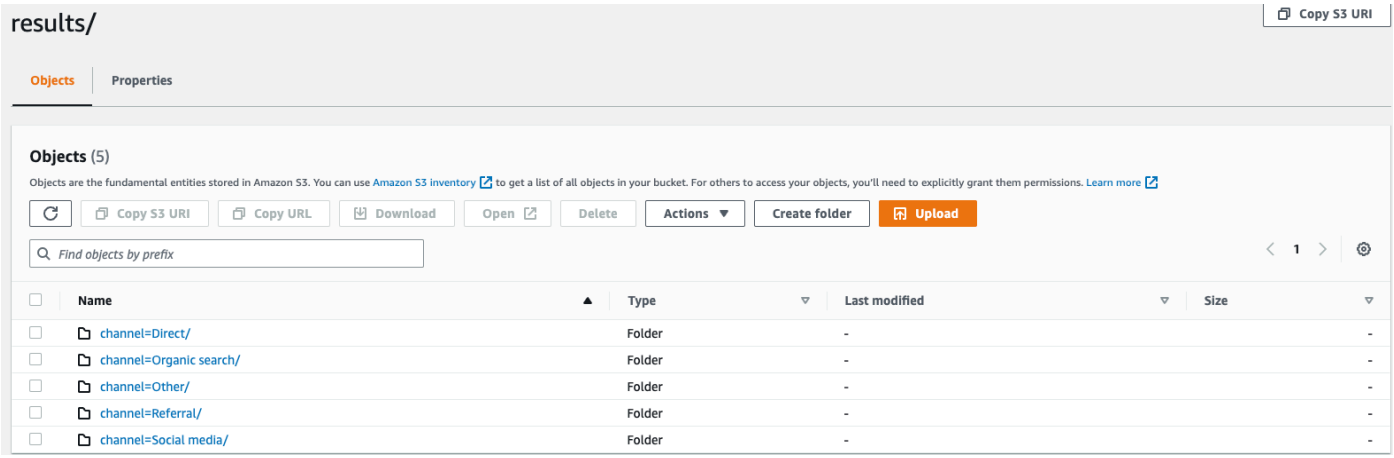
将自动创建一个folder/results/partition column = partition value/子文件夹。但是，请注意，分区列不包含在输出文件中。

partitioned_by不是语法中的必备子句。如果您选择在不进行任何分区的情况下导出数据，则可以在语法中排除该子句。

Example

假设您正在监控网站的点击流数据，并且有 5 个流量渠道direct，即Social Media、Organic SearchOther、和Referral。导出数据时，您可以选择使用列对数据进行分区Channel。在您的数据文件夹中s3://bucketname/results，您将有五个文件夹，每个文件夹都有各自的频道名称，例如，s3://bucketname/results/channel=Social Media/.在此文件夹中，您将找到通过该Social Media渠道登陆您网站的所有客户的数据。同样，剩下的频道也将有其他文件夹。

按频道列分区的导出数据



FORMAT

```
format = [ '{ CSV | PARQUET }' , default: CSV
```

用于指定写入您的 S3 存储桶的查询结果格式的关键字。您可以使用逗号 (,) 作为默认分隔符将数据导出为逗号分隔值 (CSV)，也可以以 Apache Parquet 格式（一种用于分析的高效开放列式存储格式）导出。

COMPRESSION

```
compression = [ '{ GZIP | NONE }' ], default: GZIP
```

您可以使用压缩算法压缩导出的数据，GZIP也可以通过指定NONE选项将其解压缩。

ENCRYPTION

```
encryption = [ '{ SSE_KMS | SSE_S3 }' ], default: SSE_S3
```

Amazon S3 上的输出文件使用您选择的加密选项进行加密。除了您的数据外，还会根据您选择的加密选项对清单文件和元数据文件进行加密。我们目前支持 SSE _S3 和 SSE _ KMS 加密。SSE_S3 是一种服务器端加密，Amazon S3 使用 256 位高级加密标准 () 加密对数据进行加密。AESSSE_ KMS 是一种服务器端加密，用于使用客户管理的密钥对数据进行加密。

KMS_KEY

```
kms_key = '<string>'
```

KMS密钥是客户定义的密钥，用于加密导出的查询结果。KMS密钥由密 AWS 钥管理服务 (AWS KMS) 安全管理，用于加密 Amazon S3 上的数据文件。

FIELD_DELIMITER

```
field_delimiter ='<character>' , default: (,)
```

以CSV格式导出数据时，此字段指定一个ASCII字符用于分隔输出文件中的字段，例如竖线字符 (|)、逗号 (,) 或制表符 (/t)。CSV文件的默认分隔符是逗号字符。如果数据中的值包含选定的分隔符，则分隔符将用引号字符引用。例如，如果您的数据中的值包含Time,stream，则该值将像在导出数据"Time,stream"中一样被引用。Timestream 使用的引号字符 LiveAnalytics 是双引号 (“)。

FIELD_DELIMITER如果要在中包含标题，请避免将回车符（ASCII130D，十六进制，文本 '\ r'）或换行符（ASCII10，十六进制 0A，文本'\ n'）指定为CSV，因为这将使许多解析器无法在生成的输出中正确解析标题。CSV

ESCAPED_by

```
escaped_by = '<character>', default: (\)
```

以CSV格式导出数据时，此字段指定写入 S3 存储桶的数据文件中应被视为转义字符的字符。逃跑发生在以下场景中：

1. 如果值本身包含引号字符 (“)，则将使用转义字符对其进行转义。例如，如果值为Time"stream，其中 (\) 是配置的转义字符，则将其转义为Time\"stream。
2. 如果该值包含配置的转义字符，则会对其进行转义。例如，如果值为Time\stream，则将其转义为Time\\stream。

> **Note**
>
> 如果导出的输出包含诸如数组、行或时间序列之类的复杂数据类型，则会将其序列化为JSON字符串。以下为示例。

数据类型	实际价值	如何CSV以 [序列化JSON字符串] 格式对值进行转义
数组	`[ 23,24,25 ]`	`"[23,24,25]"`
行	`( x=23.0, y=hello )`	`"{\"x\":23.0,\"y\":\"hello\"}"`
时间序列	`[ ( time=1970-01-01 00:00:00.000000010, value=100.0 ), ( time=1970-01-01 00:00:00.000000012, value=120.0 ) ]`	`"[{\"time\":\"1970-01-01 00:00:00.000000010Z\",\"value\":100.0},{\"time\":\"1970-01-01 00:00:00.000000012Z\",\"value\":120.0}]"`

INCLUDE_HEADER

```
include_header = 'true' , default: 'false'
```

以CSV格式导出数据时，此字段允许您将列名作为导出CSV数据文件的第一行。

可接受的值为 “真” 和 “假”，默认值为 “假”。诸如`escaped_by`和之类的文本转换选项也`field_delimiter`适用于标题。

> **Note**
>
> 包括标题时，请务必不要选择回车符（ASCII13，十六进制 0D，文本 '\r'）或换行符（ASCII10，十六进制 0A，文本'\n'）作为标题`FIELD_DELIMITER`，因为这将使许多解析器无法在生成的输出中正确解析标题。CSV

MAX_FILE_SIZE

```
max_file_size = 'X[MB|GB]' , default: '78GB'
```

此字段指定该UNLOAD语句在 Amazon S3 中创建的最大文件大小。该UNLOAD语句可以创建多个文件，但是写入 Amazon S3 的每个文件的最大大小将与该字段中指定的大小差不多。

该字段的值必须介于 16 MB 到 78 GB 之间（含）。你可以用整数（例如）或小数（如0.5GB或24.7MB）来指定。12GB默认值为 78 GB。

实际文件大小是写入文件时的近似值，因此实际最大大小可能不完全等于您指定的数字。

逻辑运算符

Timestream fo LiveAnalytics r 支持以下逻辑运算符。

运算符	描述	示例
AND	如果两个值都为真，则为真	a AND b
或	如果任一值为真，则为真	a 或 b
NOT	如果值为假，则为真	NOTa

- AND比较的结果可能是NULL表达式的一边还是两边都是NULL。
- 如果AND运算符的至少一侧是，则表达式FALSE的计算结果为。FALSE
- OR比较的结果可能是NULL表达式的一边还是两边都是NULL。
- 如果OR运算符的至少一侧是，则表达式TRUE的计算结果为。TRUE
- 的逻辑补语NULL是NULL。

以下真值表演示了如何处理 NULL in an AND dOR：

A	B	A 和 b	A 或 b
null	null	null	null

A	B	A 和 b	A 或 b
false	null	false	null
null	false	false	null
true	null	null	true
null	true	null	true
false	false	false	false
true	false	false	true
false	true	false	true
true	true	true	true

以下真值表演示了如何处理 NULL inNOT：

A	不是
null	null
true	false
false	true

比较运算符

Timestream for LiveAnalytics 支持以下比较运算符。

运算符	描述
<	Less than
>	Greater than
<=	小于或等于

运算符	描述
>=	大于或等于
=	Equal
<>	Not equal
!=	Not equal

Note

- 运BETWEEN算符测试一个值是否在指定范围内。语法如下：

  ```
  BETWEEN min AND max
  ```

  BETWEEN或NOT BETWEEN语句NULL中存在将导致该语句的计算结果为NULL。
- IS NULL 运IS NOT NULL算符测试一个值是否为空（未定义）。NULL与 with 一起使用IS NULL计算结果为 true。
- 在中SQL，NULL值表示未知值。

比较函数

的 Timestream LiveAnalytics 支持以下比较函数。

主题

- 最棒的 ()
- 至少 ()
- ALL()、ANY () 和 SOME ()

最棒的 ()

g reatest () 函数返回所提供值中的最大值。NULL如果提供的值中存在任何值，则返回NULL。语法如下所示。

```
greatest(value1, value2, ..., valueN)
```

至少 ()

le ast () 函数返回所提供值中的最小值。NULL如果提供的值中存在任何值，则返回NULL。语法如下所示。

```
least(value1, value2, ..., valueN)
```

ALL()、ANY () 和 SOME ()

ALL、ANY和SOME量词可以通过以下方式与比较运算符一起使用。

Expression	含义
A = ALL (...)	当 A 等于所有值时，计算结果为 true。
A <> ALL (...)	当 A 与任何值都不匹配时，计算结果为 true。
A < ALL (...)	当 A 小于最小值时，计算结果为 true。
A = ANY (...)	当 A 等于任一值时，计算结果为 true。
A <> ANY (...)	当 A 与一个或多个值不匹配时，计算结果为 true。
A < ANY (...)	当 A 小于最大值时，计算结果为 true。

示例和用法说明

> **Note**
>
> 使用ALL、ANY或时SOME，如果比较值是文字列表，则VALUES应使用关键字。

例如：**ANY()**

查询语句ANY()中的示例如下。

```
SELECT 11.7 = ANY (VALUES 12.0, 13.5, 11.7)
```

同一操作的替代语法如下。

```
SELECT 11.7 = ANY (SELECT 12.0 UNION ALL SELECT 13.5 UNION ALL SELECT 11.7)
```

在本例中，ANY()计算为。True

例如：**ALL()**

查询语句ALL()中的示例如下。

```
SELECT 17 < ALL (VALUES 19, 20, 15);
```

同一操作的替代语法如下。

```
SELECT 17 < ALL (SELECT 19 UNION ALL SELECT 20 UNION ALL SELECT 15);
```

在本例中，ALL()计算为。False

例如：**SOME()**

查询语句SOME()中的示例如下。

```
SELECT 50 >= SOME (VALUES 53, 77, 27);
```

同一操作的替代语法如下。

```
SELECT 50 >= SOME (SELECT 53 UNION ALL SELECT 77 UNION ALL SELECT 27);
```

在本例中，SOME()计算为。True

条件表达式

Timestream LiveAnalytics 支持以下条件表达式。

主题

- 该CASE声明

- IF 语句
- 该COALESCE声明
- 该NULLIF声明
- 该TRY声明

该CASE声明

该CASE语句从左到右搜索每个值表达式，直到找到一个等于的表达式。expression如果找到匹配项，则返回匹配值的结果。如果未找到匹配项，则返回ELSE子句的结果（如果存在）；否则返回null该子句的结果。语法如下：

```
CASE expression
    WHEN value THEN result
    [ WHEN ... ]
    [ ELSE result ]
END
```

Timestream 还支持以下语CASE句语法。在这种语法中，“searched” 表单从左到右计算每个布尔条件，直到有一个布尔条件，true然后返回匹配的结果。如果没有条件true，则返回ELSE子句的结果（如果存在）；否则返回null该子句的结果。备用语法见下文：

```
CASE
    WHEN condition THEN result
    [ WHEN ... ]
    [ ELSE result ]
END
```

IF 语句

I F 语句将条件评估为真或假，并返回相应的值。Timestream 支持 I F 的以下两种语法表示形式：

```
if(condition, true_value)
```

true_value如果条件为，则此语法计算并返回true；否则返回nulltrue_value且不进行求值。

```
if(condition, true_value, false_value)
```

true_value如果条件为，则此语法计算并返回true，否则计算并返回。false_value

示例

```
SELECT
  if(true, 'example 1'),
  if(false, 'example 2'),
  if(true, 'example 3 true', 'example 3 false'),
  if(false, 'example 4 true', 'example 4 false')
```

_col0	_col1	_col2	_col3
example 1	- null	example 3 true	example 4 false

该COALESCE声明

COALESCE返回参数列表中的第一个非空值。语法如下：

```
coalesce(value1, value2[,...])
```

该NULLIF声明

I F 语句将条件评估为真或假，并返回相应的值。Timestream 支持 I F 的以下两种语法表示形式：

NULLIF如果value1等于，则返回 nullvalue2；否则返回value1。语法如下：

```
nullif(value1, value2)
```

该TRY声明

该TRY函数计算表达式并通过返回null来处理某些类型的错误。语法如下：

```
try(expression)
```

转换函数

Timestream LiveAnalytics 支持以下转换函数。

主题

cast()

将值显式转换为类型的强制转换函数的语法如下。

```
cast(value AS type)
```

try_cast ()

Timestream LiveAnalytics 还支持 try_cast 函数，该函数与强制转换类似，但如果强制转换失败则返回 null。语法如下所示。

```
try_cast(value AS type)
```

数学运算

Timestream fo LiveAnalytics r 支持以下数学运算符。

运算符	描述
+	加
-	减
*	乘
/	除法（整数除法执行截断）
%	模量（余数）

数学函数

Timestream LiveAnalytics 支持以下数学函数。

函数	输出数据类型	描述
腹肌 (x)	[与输入相同]	返回 x 的绝对值。
cbrt (x)	double	返回 x 的立方根。
天花板 (x) 或天花板 (x)	[与输入相同]	返回 x 四舍五入到最接近的整数。
度 (x)	double	将以弧度为单位的角度 x 转换为度。
e ()	double	返回常量欧拉数。
exp (x)	double	返回以 x 为底的欧拉数。
楼层 (x)	[与输入相同]	返回 x 向下舍入到最接近的整数。
from_base（字符串，基数）	bigint	返回解释为基数的字符串的值。
ln (x)	double	返回 x 的自然对数。
log2 (x)	double	返回 x 以 2 为底的对数。
log10 (x)	double	返回 x 以 10 为底的对数。
模组 (n, m)	[与输入相同]	返回 n 除以 m 的模数（余数）。
pi ()	double	返回常量 Pi。
pow (x, p) 或功率 (x, p)	double	返回 x 以 p 为底的次方。
弧度 (x)	double	将以度为单位的角度 x 转换为弧度。
rand () 或随机 ()	double	返回 0.0 1.0 范围内的伪随机值。

函数	输出数据类型	描述
随机 (n)	[与输入相同]	返回 0 到 n 之间的伪随机数（不包括）。
圆形 (x)	[与输入相同]	返回 x 四舍五入到最接近的整数。
圆形 (x, d)	[与输入相同]	返回 x 四舍五入到 d 个小数位。
标志 (x)	[与输入相同]	返回 x 的符号函数，即： • 如果参数为 0 则为 0 • 如果参数大于 0 则为 1 • 如果参数小于 0，则为 -1。 对于双重参数，该函数还会返回： • 如果参数是 NaN，则为 NaN • 如果参数为 +Infinity，则为 1 • 如果参数是-Infinity，则为 -1。
sqrt (x)	double	返回 x 的平方根。
to_base (x, radix)	varchar	返回 x 的基数表示形式。
截断 (x)	double	通过在小数点后删除数字来返回 x 四舍五入为整数。
acos (x)	double	返回 x 的反余弦值。
asin (x)	double	返回 x 的反正弦值。
atan (x)	double	返回 x 的反正切。
atan2 (y, x)	double	返回 y/x 的反正切值。

函数	输出数据类型	描述
cos (x)	double	返回 x 的余弦值。
cosh (x)	double	返回 x 的双曲余弦值。
罪恶 (x)	double	返回 x 的正弦值。
棕褐色 (x)	double	返回 x 的正切值。
tanh (x)	double	返回 x 的双曲正切值。
无穷大 ()	double	返回表示正无穷大的常数。
is_finite (x)	布尔值	确定 x 是否有限。
is_infinite (x)	布尔值	确定 x 是否为无穷大。
is_nan (x)	布尔值	确定 x 是否为 not-a-number。
nan ()	double	返回表示的常量 not-a-num ber。

字符串运算符

Timestream for LiveAnalytics 支持用于连接一个或多个字符串的 | | 运算符。

字符串函数

> **Note**
>
> 除非另有说明，否则假定这些函数的输入数据类型为 varchar。

函数	输出数据类型	描述
chr (n)	varchar	以 varchar 的形式返回 Unicode 码点 n。

函数	输出数据类型	描述
代码点 (x)	整数	返回 str 中唯一字符的 Unicode 码点。
concat (x1,..., xN)	varchar	返回 x1、x2、...、xN 的串联。
hamming_distance (x1, x2)	bigint	返回 x1 和 x2 的汉明距离，即相应字符不同的位置数。请注意，两个 varchar 输入的长度必须相同。
长度 (x)	bigint	返回 x 的长度（以字符为单位）。
levenshtein_distance (x1, x2)	bigint	返回 x1 和 x2 的 Levenshtein 编辑距离，即将 x1 更改为 x2 所需的最小单字符编辑次数（插入、删除或替换）。
更低 (x)	varchar	将 x 转换为小写。
加载 (x1，bigint size，x2)	varchar	用左键盘 x1 调整字符大小 x2。如果 size 小于 x1 的长度，则结果将被截断为大小字符。size 不能为负数，x2 必须为非空。
ltrim (x)	varchar	从 x 中移除前导空格。
替换 (x1, x2)	varchar	从 x1 中移除 x2 的所有实例。
替换 (x1、x2、x3)	varchar	用 x1 中的 x3 替换 x2 的所有实例。
反向 (x)	varchar	返回字符顺序相反的 x。

函数	输出数据类型	描述
rpad (x1，bigint size，x2)	varchar	右键填充 x1，用 x2 调整字符的大小。如果 size 小于 x1 的长度，则结果将被截断为大小字符。size 不能为负数，x2 必须为非空。
rtrim (x)	varchar	删除 x 的尾随空格。
split (x1, x2)	array(varchar)	在分隔符 x2 上拆分 x1 并返回一个数组。
分割（x1、x2、bigint 限制）	array(varchar)	在分隔符 x2 上拆分 x1 并返回一个数组。数组中的最后一个元素始终包含 x1 中剩下的所有内容。限制必须为正数。
split_part (x1、x2、bigint pos)	varchar	在分隔符 x2 上拆分 x1，然后在 pos 处返回 varchar 字段。字段索引以 1 开头。如果 pos 大于字段数，则返回 null。
strpos (x1, x2)	bigint	返回 x1 中 x2 的第一个实例的起始位置。位置从 1 开始。如果未找到，则返回 0。
strpos（x1、x2、bigint 实例）	bigint	返回 x2 的第 N 个实例在 x1 中的位置。实例必须为正数。位置从 1 开始。如果未找到，则返回 0。
strrpos (x1, x2)	bigint	返回 x1 中最后一个 x2 实例的起始位置。位置从 1 开始。如果未找到，则返回 0。

函数	输出数据类型	描述
strrpos（x1、x2、bigint 实例）	bigint	返回 x1 中 x2 的第 N 个实例从 x1 的末尾开始的位置。实例必须为正数。位置从 1 开始。如果未找到，则返回 0。
位置 (x2 英寸 x1)	bigint	返回 x1 中 x2 的第一个实例的起始位置。位置从 1 开始。如果未找到，则返回 0。
substr（x，bigint 开头）	varchar	从起始位置开始返回 x 的其余部分。位置从 1 开始。负起始位置被解释为相对于 x 的结尾。
substr (x、bigint start、bigint len)	varchar	从起始位置开始返回长度为 len 的 x 的子字符串。位置从 1 开始。负起始位置被解释为相对于 x 的结尾。
修剪 (x)	varchar	删除 x 的前导和尾随空格。
上方 (x)	varchar	将 x 转换为大写。

数组运算符

的 Timestream LiveAnalytics 支持以下数组运算符。

运算符	描述
[]	访问数组中第一个索引从 1 开始的元素。
||	将一个数组与另一个数组或相同类型的元素连接起来。

数组函数

的 Timestream LiveAnalytics 支持以下数组函数。

函数	输出数据类型	描述
array_distinct (x)	数组	从数组 x 中删除重复的值。 `SELECT array_distinct(ARRAY[1,2,2,3])` 结果示例：`[ 1,2,3 ]`
array_intersect (x, y)	数组	返回 x 和 y 交叉处的元素的数组，没有重复项。 `SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5])` 结果示例：`[ 3 ]`
array_union (x, y)	数组	返回 x 和 y 并集元素的数组，没有重复项。 `SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5])` 结果示例：`[ 1,2,3,4,5 ]`
array_except (x, y)	数组	返回 x 中但不是 y 中的元素的数组，没有重复项。 `SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5])` 结果示例：`[ 1,2 ]`

函数	输出数据类型	描述
array_join（x，分隔符，空值替换）	varchar	使用分隔符和可选字符串连接给定数组的元素以替换空值。 `SELECT array_join(ARRAY[1,2,3], ';', '')` 结果示例：1;2;3
array_max (x)	与数组元素相同	返回输入数组的最大值。 `SELECT array_max(ARRAY[1,2,3])` 结果示例：3
array_min (x)	与数组元素相同	返回输入数组的最小值。 `SELECT array_min(ARRAY[1,2,3])` 结果示例：1
数组位置（x，元素）	bigint	返回该元素在数组 x 中首次出现的位置（如果未找到，则返回 0）。 `SELECT array_position(ARRAY[3,4,5,9], 5)` 结果示例：3

函数	输出数据类型	描述
array_remove (x, 元素)	数组	从数组 x 中移除所有等于元素的元素。 `SELECT array_remove(ARRAY[3,4,5,9], 4)` 结果示例：`[ 3,5,9 ]`
数组排序 (x)	数组	对数组 x 进行排序并返回。x 的元素必须是可排序的。Null 元素将放在返回数组的末尾。 `SELECT array_sort(ARRAY[6,8,2,9,3])` 结果示例：`[ 2,3,6,8,9 ]`
arrays_overlay (x, y)	布尔值	测试数组 x 和 y 是否有共同的非空元素。如果没有共同的非空元素，但任何一个数组都包含空值，则返回 null。 `SELECT arrays_overlap(ARRAY[6,8,2,9,3], ARRAY[6,8])` 结果示例：`true`
基数 (x)	bigint	返回数组 x 的大小。 `SELECT cardinality(ARRAY[6,8,2,9,3])` 结果示例：5

函数	输出数据类型	描述
concat（array1、array2、...、arrayN）	数组	连接数组 array1、array2、...、arrayN。 `SELECT concat(ARRAY[6,8,2,9,3], ARRAY[11,32], ARRAY[6,8,2,0,14])` 结果示例：`[ 6,8,2,9,3,11,32,6,8,2,0,14 ]`
元素_at（数组 (E)，索引）	E	返回给定索引处的数组元素。如果索引 < 0，则 element_at 访问从最后一个到第一个的元素。 `SELECT element_at(ARRAY[6,8,2,9,3], 1)` 结果示例：6
重复（元素，计数）	数组	重复元素计数次数。 `SELECT repeat(1, 3)` 结果示例：`[ 1,1,1 ]`
反向 (x)	数组	返回一个与数组 x 的顺序相反的数组。 `SELECT reverse(ARRAY[6,8,2,9,3])` 结果示例：`[ 3,9,2,8,6 ]`

函数	输出数据类型	描述
顺序（开始、停止）	数组（大整数）	从头到尾生成一个整数序列，如果 start 小于或等于 stop，则以 1 为增量，否则为 -1。 `SELECT sequence(3, 8)` 结果示例：`[ 3,4,5,6,7,8 ]`
顺序（开始、停止、步进）	数组（大整数）	从头到尾生成一个整数序列，逐渐递增。 `SELECT sequence(3, 15, 2)` 结果示例：`[ 3,5,7,9,11,13,15 ]`

函数	输出数据类型	描述
顺序（开始、停止）	数组（时间戳）	生成从开始日期到停止日期的时间戳序列，以 1 天为增量。 `SELECT sequence( '2023-04-02 19:26:12.941000000', '2023-04-06 19:26:12.941000000', 1d)` 结果示例：[2023-04-02 19:26:12.941000000 ,2023-04-03 19:26:12.941000000 ,2023-04-04 19:26:12.941000000 ,2023-04-05 19:26:12.941000000 ,2023-04-06 19:26:12.941000000]

函数	输出数据类型	描述
顺序（开始、停止、步进）	数组（时间戳）	生成从开始到停止的时间戳序列，逐渐递增。步进的数据类型是间隔。 `SELECT sequence( '2023-04-02 19:26:12.941000000', '2023-04-10 19:26:12.941000000', 2d)` 结果示例：[2023-04-02 19:26:12.941000000 ,2023-04-04 19:26:12.941000000 ,2023-04-06 19:26:12.941000000 ,2023-04-08 19:26:12.941000000 ,2023-04-10 19:26:12.941000000]
随机播放 (x)	数组	生成给定数组 x 的随机排列。 `SELECT shuffle(ARRAY[6,8,2,9,3])` 结果示例：[6,3,2,9,8]
切片（x、起点、长度）	数组	子集数组 x 从索引开头开始（如果起始为负数，则从结尾开始），长度为。 `SELECT slice(ARRAY[6,8,2,9,3], 1, 3)` 结果示例：[6,8,2]

函数	输出数据类型	描述
zip（数组 1，数组 2 [，...] ）	数组（行）	按元素将给定数组合并成单个行数组。如果参数的长度不均匀，则用缺失值填充NULL。 `SELECT zip(ARRAY [6,8,2,9,3], ARRAY[15, 24])` 结果示例：`[ ( 6, 15 ), ( 8, 24 ),( 2, - ), ( 9, - ),( 3, - ) ]`

按位函数

的 Timestream LiveAnalytics 支持以下按位函数。

函数	输出数据类型	描述

bit_count（bigint、bigint）	bigint（二进制补码）	返回第一个 bigint 参数中的位数，其中第二个参数是位有符号整数，例如 8 或 64。
		`SELECT bit_count(19, 8)`
		结果示例：3
		`SELECT bit_count(19, 2)`
		结果示例：`Number must be representable with the bits specified . 19 can not be represented with 2 bits`

函数	输出数据类型	描述
bitwise_and (bigint、bigint)	bigint（二进制补码）	返回 bigint AND 参数的按位值。 `SELECT bitwise_and(12, 7)` 结果示例：4
bitwise_not (bigint)	bigint（二进制补码）	返回 bigint NOT 参数的按位值。 `SELECT bitwise_not(12)` 结果示例：-13
bitwise_or (bigint、bigint)	bigint（二进制补码）	返回 bigint 参数的按位或。 `SELECT bitwise_or(12, 7)` 结果示例：15
bitwise_xor（bigint、bigint）	bigint（二进制补码）	返回 bigint XOR 参数的按位值。 `SELECT bitwise_xor(12, 7)` 结果示例：11

正则表达式函数

Timestream 中的正则表达式函数 LiveAnalytics 支持 Java 模式语法。Timestream LiveAnalytics 支持以下正则表达式函数。

函数	输出数据类型	描述
regexp_extract_all（字符串、模式）	array(varchar)	返回字符串中与正则表达式模式匹配的子字符串。 `SELECT regexp_extract_all('example expect complex', 'ex\w')` 结果示例：`[ exa,exp ]`
regexp_extract_all（字符串、模式、组）	array(varchar)	在字符串中查找所有出现的正则表达式模式并返回捕获组号组。 `SELECT regexp_extract_all('example expect complex', '(ex)(\w)', 2)` 结果示例：`[ a,p ]`
regexp_extract（字符串、模式）	varchar	返回字符串中与正则表达式模式匹配的第一个子字符串。 `SELECT regexp_extract('example expect', 'ex\w')` 结果示例：`exa`
regexp_extract（字符串、模式、组）	varchar	查找字符串中第一次出现的正则表达式模式并返回捕获组号组。 `SELECT regexp_extract('example`

函数	输出数据类型	描述
		expect', '(ex)(\w)', 2) 结果示例：a
regexp_like（字符串、模式）	布尔值	评估正则表达式模式并确定它是否包含在字符串中。此函数与LIKE运算符类似，不同之处在于模式只需要包含在字符串中，而不必匹配所有字符串。换句话说，它执行的是包含操作而不是匹配操作。您可以通过使用 ^ 和 $ 锚定模式来匹配整个字符串。 SELECT regexp_like('example', 'ex') 结果示例：true
regexp_replace（字符串、模式）	varchar	从字符串中删除与正则表达式模式匹配的子字符串的所有实例。 SELECT regexp_replace('example expect', 'expect') 结果示例：example

函数	输出数据类型	描述
regexp_replace（字符串、模式、替换）	varchar	用替换替换字符串中与正则表达式模式匹配的子字符串的每个实例。可以在替换中引用捕获组，使用 $g 表示带编号的组，使用 $ {name} 表示命名的组。如果用反斜杠 (\ $) 转义美元符号 ($)，则可以在替换项中包含美元符号 ($)。 `SELECT regexp_replace('example expect', 'expect', 'surprise')` 结果示例：`example surprise`
regexp_replace（字符串、模式、函数）	varchar	使用 function 替换字符串中与正则表达式模式匹配的子字符串的每个实例。每次匹配都会调用 lambda 表达式函数，捕获组作为数组传递。捕获组号从 1 开始；整个匹配项没有分组（如果需要，请用括号将整个表达式括起来）。 `SELECT regexp_replace('example', '(\w)', x -> upper(x[1]))` 结果示例：EXAMPLE

函数	输出数据类型	描述
regexp_split（字符串、模式）	array(varchar)	使用正则表达式模式拆分字符串并返回一个数组。保留尾随的空字符串。 `SELECT regexp_split('example', 'x')` 结果示例：`[ e,ample ]`

日期/时间运算符

> **Note**
>
> 的时间流 LiveAnalytics 不支持负时间值。任何导致负时间的操作都会导致错误。

Timestream LiveAnalytics 支持对`timestamps`、`dates`和`intervals`进行以下操作。

运算符	描述
+	加
-	减

主题

- 操作
- 加
- 减

操作

运算的结果类型基于操作数。`3s`可以使用间隔文字`1day`，例如和。

```
SELECT date '2022-05-21' + interval '2' day
```

```
SELECT date '2022-05-21' + 2d
```

```
SELECT date '2022-05-21' + 2day
```

每种结果的示例：2022-05-23

间隔单位包括second、minute、hour、day、week、month、和year。但在某些情况下，并非所有情况都适用。例如，不能将秒、分钟和小时与日期相加，也不能从日期中减去。

```
SELECT interval '4' year + interval '2' month
```

结果示例：4-2

```
SELECT typeof(interval '4' year + interval '2' month)
```

结果示例：interval year to month

间隔运算的结果类型可能是'interval year to month'或'interval day to second'取决于操作数。间隔可以与和相加，也可以从dates和timestamps中减去。但是，timestamp不能在date或中添加或减去 a date。timestamp要查找与日期或时间戳相关的间隔或持续时间，请参阅中的date_diff和相关函数。[间隔和持续时间]

加

Example

```
SELECT date '2022-05-21' + interval '2' day
```

结果示例：2022-05-23

Example

```
SELECT typeof(date '2022-05-21' + interval '2' day)
```

结果示例：date

Example

```
SELECT interval '2' year + interval '4' month
```

结果示例：2-4

Example

```
SELECT typeof(interval '2' year + interval '4' month)
```

结果示例：interval year to month

减

Example

```
SELECT timestamp '2022-06-17 01:00' - interval '7' hour
```

结果示例：2022-06-16 18:00:00.000000000

Example

```
SELECT typeof(timestamp '2022-06-17 01:00' - interval '7' hour)
```

结果示例：timestamp

Example

```
SELECT interval '6' day - interval '4' hour
```

结果示例：5 20:00:00.000000000

Example

```
SELECT typeof(interval '6' day - interval '4' hour)
```

结果示例：interval day to second

日期/时间函数

> **Note**
>
> 的时间流 LiveAnalytics 不支持负时间值。任何导致负时间的操作都会导致错误。

timestream LiveAnalytics 使用UTC时区表示日期和时间。Timestream 支持以下日期和时间函数。

主题

一般和转换

Timestream for LiveAnalytics 支持以下日期和时间的常规函数和转换函数。

函数	输出数据类型	描述
当前日期	date	返回当前日期UTC。不使用括号。 `SELECT current_date` 结果示例：`2022-07-07` **Note** 这也是一个保留关键字。有关保留关键字的列表，请参阅保留关键字。

函数	输出数据类型	描述
当前时间	时间	返回当前时间UTC。不使用括号。 `SELECT current_time` 结果示例：`17:41:52.827000000` **Note** 这也是一个保留关键字。有关保留关键字的列表，请参阅保留关键字。
当前时间戳或现在 ()	时间戳	返回当前的时间戳。UTC `SELECT current_timestamp` 结果示例：`2022-07-07 17:42:32.939000000` **Note** 这也是一个保留关键字。有关保留关键字的列表，请参阅保留关键字。

函数	输出数据类型	描述
当前时区 ()	varchar 该值将是 'UTC。'	Timestream 使用UTC时区作为日期和时间。 `SELECT current_timezone()` 结果示例：UTC
日期（varchar (x)）、日期（时间戳）	date	`SELECT date(TIMESTAMP '2022-07-07 17:44:43.771000000')` 结果示例：2022-07-07
本月的最后一天（时间戳）、当月的最后一天（日期）	date	`SELECT last_day_of_month(TIMESTAMP '2022-07-07 17:44:43.771000000')` 结果示例：2022-07-31
from_iso8601_timestamp（字符串）	时间戳	将 ISO 8601 时间戳解析为内部时间戳格式。 `SELECT from_iso8601_timestamp('2022-06-17T08:04:05.000000000+05:00')` 结果示例：2022-06-17 03:04:05.000000000

函数	输出数据类型	描述
from_iso8601_date（字符串）	date	将 ISO 8601 日期字符串解析为指定日期 UTC 00:00:00 的内部时间戳格式。 `SELECT from_iso8601_date('2022-07-17')` 结果示例：2022-07-17
to_iso8601（时间戳）、to_iso8601（日期）	varchar	返回一个 ISO 8601 格式的输入字符串。 `SELECT to_iso8601(from_iso8601_date('2022-06-17'))` 结果示例：2022-06-17
from_mliseconds (bigint)	时间戳	`SELECT from_milliseconds(1)` 结果示例：1970-01-01 00:00:00.001000000
from_nanoseconds (bigint)	时间戳	`select from_nanoseconds(300000001)` 结果示例：1970-01-01 00:00:00.300000001

函数	输出数据类型	描述
from_unixtime（双精度）	时间戳	返回与所提供的 unixtime 相对应的时间戳。 `SELECT from_unixtime(1)` 结果示例：`1970-01-01 00:00:01.000000000`
当地时间	时间	返回当前时间UTC。不使用括号。 `SELECT localtime` 结果示例：`17:58:22.654000000` **Note** 这也是一个保留关键字。有关保留关键字的列表，请参阅保留关键字。

函数	输出数据类型	描述
本地时间戳	时间戳	返回当前的时间戳。UTC不使用括号。 `SELECT localtimestamp` 结果示例：2022-07-07 17:59:04.368000000 **Note** 这也是一个保留关键字。有关保留关键字的列表，请参阅保留关键字。

to_mliseconds（间隔日到秒）、to_mliseconds（时间戳）	bigint	`SELECT to_milliseconds(INTERVAL '2' DAY + INTERVAL '3' HOUR)`
		结果示例：183600000
		`SELECT to_milliseconds(TIMESTAMP '2022-06-17 17:44:43.771000000')`
		结果示例：1655487883771

函数	输出数据类型	描述

to_nanoseconds（间隔日到秒）、to_nanoseconds（时间戳）	bigint	`SELECT to_nanoseconds(INTERVAL '2' DAY + INTERVAL '3' HOUR)`
		
		结果示例：183600000000000
		
		`SELECT to_nanoseconds(TIMESTAMP '2022-06-17 17:44:43.771000678')`
		
		结果示例：1655487883771000678

to_unixtime（时间戳）	double	返回所提供时间戳的 unixtime。 `SELECT to_unixtime('2022-06-17 17:44:43.771000000')` 结果示例：1.655487883771E9
date_trunc（单位，时间戳）	时间戳	返回截断为单位的时间戳，其中单位为 [秒、分钟、小时、日、周、月、季或年] 之一。 `SELECT date_trunc('minute', TIMESTAMP '2022-06-17 17:44:43.771000000')` 结果示例：2022-06-17 17:44:00.000000000

间隔和持续时间

Timestream for LiveAnalytics 支持以下日期和时间间隔和持续时间函数。

函数	输出数据类型	描述
date_add（单位、大整数、日期）、date_add（单位、大整数、时间）、date_add（varchar (x)、bigint、timestamp）	时间戳	添加一个 bigint 单位，其中单位是 [秒、分钟、小时、日、周、月、季度或年] 之一。 `SELECT date_add('hour', 9, TIMESTAMP '2022-06-17 00:00:00')` 结果示例：`2022-06-17 09:00:00.000000000`
date_diff（单位、日期、日期）、日期差异（单位、时间、时间）、日期差异（单位、时间戳、时间戳）	bigint	返回差值，其中单位为 [秒、分钟、小时、日、周、月、季度或年] 之一。 `SELECT date_diff('day', DATE '2020-03-01', DATE '2020-03-02')` 结果示例：`1`
解析持续时间（字符串）	interval	解析输入字符串以返回`interval`等效的字符串。 `SELECT parse_duration('42.8ms')` 结果示例：`0 00:00:00.042800000`

函数	输出数据类型	描述
		`SELECT typeof(parse_duration('42.8 ms'))` 结果示例：`interval day to second`

函数	输出数据类型	描述

bin（时间戳、间隔）	时间戳	将timestamp 参数的整数值向下舍入到interval参数整数值的最接近的倍数。
		
		这个返回值的含义可能并不明显。它是使用整数算术计算的，首先是将时间戳整数除以间隔整数，然后将结果乘以间隔整数。
		
		请记住，时间戳将时间UTC点指定为自POSIX纪元（1970年1月1日）以来经过的几分之一秒，因此返回值很少与日历单位一致。例如，如果您将 30 天的时间间隔指定为 30 天，则将自纪元以来的所有天数划分为 30 天增量，并返回最近 30 天增量的起始日期，这与日历月无关。
		
		下面是一些示例：
		
		`bin(TIMESTAMP '2022-06-17 10:15:20', 5m) ==> 2022-06-17 10:15:00.000000000`
		`bin(TIMESTAMP '2022-06-17 10:15:20', 1d) ==> 2022-06-17 00:00:00.000000000`
		`bin(TIMESTAMP '2022-06-17 10:15:20', 10day) ==> 2022-06-17 00:00:00.000000000`
		`bin(TIMESTAMP '2022-06-17 10:15:20', 30day)`

函数	输出数据类型	描述
		==> 2022-05-28 00:00:00.000000000
之前（间隔）	时间戳	返回与当前时间戳对应的值。interval `SELECT ago(1d)` 结果示例：2022-07-06 21:08:53.245000000
间隔文字，例如 1h、1d 和 30m	interval	间隔字面值便于 parse_duration（字符串）。例如，1d 与 parse_duration('1d') 相同。这允许在任何使用间隔的地方使用文字。例如，ago(1d) 和 bin(*<timestamp>* , 1m)。

一些间隔文字充当 parse_duration 的简写。例如，parse_duration('1day')、1dayparse_duration('1d')、，1d每个都返回类型100:00:00.000000000所在的位置interval day to second。允许以提供给的格式留出空间parse_duration。例如，parse_duration('1day')还会返回00:00:00.000000000。但1day不是间隔字面意思。

与之相关的单位interval day to second是 ns、纳秒、us、微秒、毫秒、s、秒、m、分钟、h、小时、d 和天。

还有interval year to month。与年与月间隔相关的单位是 y、年和月。例如，SELECT 1year退货1-0。 SELECT 12month还会返回1-0。 SELECT 8month返回0-8。

尽管单位quarter也可用于某些函数，例如date_trunc和date_add，quarter但不能作为区间文字的一部分使用。

格式化和解析

Timestream for LiveAnalytics 支持以下日期和时间的格式化和解析函数。

函数	输出数据类型	描述
日期格式（时间戳，变量字符 (x)）	varchar	有关此函数使用的格式说明符的更多信息，请参见 # https://trino.io/docs/current/functions/datetime.html mysql-date-functions `SELECT date_format(TIMESTAMP '2019-10-20 10:20:20', '%Y-%m-%d %H:%i:%s')` 结果示例：2019-10-20 10:20:20
date_parse（varchar (x)、varchar (y)）	时间戳	有关此函数使用的格式说明符的更多信息，请参见 # https://trino.io/docs/current/functions/datetime.html mysql-date-functions `SELECT date_parse('2019-10-20 10:20:20', '%Y-%m-%d %H:%i:%s')` 结果示例：2019-10-20 10:20:20.000000000
format_datetime（时间戳，varchar (x)）	varchar	有关此函数使用的格式字符串的更多信息，请参阅 http://joda-time.sourceforge。 net/apidocs/org/joda/time/format/DateTimeFormat.html `SELECT format_datetime(parse_datet`

函数	输出数据类型	描述
		`ime('1968-01-13 12', 'yyyy-MM-dd HH'), 'yyyy-MM-dd HH')` 结果示例：`1968-01-13 12`
parse_datetime (varchar (x)、varchar (y))	时间戳	有关此函数使用的格式字符串的更多信息，请参阅 http://joda-time.sourceforge。 net/apidocs/org/joda/time/format/DateTimeFormat.html `SELECT parse_datetime('2019-12-29 10:10 PST', 'uuuu-LL-dd HH:mm z')` 结果示例：`2019-12-29 18:10:00.000000000`

提取

Timestream for LiveAnalytics 支持以下日期和时间提取函数。提取函数是其余便捷函数的基础。

函数	输出数据类型	描述
extract	bigint	从时间戳中提取一个字段，其中字段是 [、、、、_OF_ YEAR QUARTER、_ OF_ MONTH WEEK DAY、DAY _OF_、MONTH、DAY _OF_、_OF_ WEEK DOW、、、DAY YEAR、、DOY、YEAR或] 之一。WEEK YOW HOUR MINUTE SECOND

函数	输出数据类型	描述
		`SELECT extract(YEAR FROM '2019-10-12 23:10:34.000000000')` 结果示例：2019
日（时间戳）、日（日期）、日（日到秒的间隔）	bigint	`SELECT day('2019-10-12 23:10:34.000000000')` 结果示例：12
day_of_month（时间戳）、月日_of_month（日到秒的间隔）	bigint	`SELECT day_of_month('2019-10-12 23:10:34.000000000')` 结果示例：12
星期_of_week（时间戳）、星期_of_week（日期）	bigint	`SELECT day_of_week('2019-10-12 23:10:34.000000000')` 结果示例：6
年日_年（时间戳）、年日_年（日期）	bigint	`SELECT day_of_year('2019-10-12 23:10:34.000000000')` 结果示例：285
向下（时间戳）、向下（日期）	bigint	day_of_week 的别名
doy（时间戳）、doy（日期）	bigint	day_of_year 的别名

函数	输出数据类型	描述
小时（时间戳）、小时（时间）、小时（日到秒间隔）	bigint	`SELECT hour('2019-10-12 23:10:34.000000000')` 结果示例：23
毫秒（时间戳）、毫秒（时间）、毫秒（间隔日到秒）	bigint	`SELECT millisecond('2019-10-12 23:10:34.000000000')` 结果示例：0
分钟（时间戳）、分钟（时间）、分钟（日到秒间隔）	bigint	`SELECT minute('2019-10-12 23:10:34.000000000')` 结果示例：10
月（时间戳）、月（日期）、月（年到月的间隔）	bigint	`SELECT month('2019-10-12 23:10:34.000000000')` 结果示例：10
纳秒（时间戳）、纳秒（时间）、纳秒（间隔日到秒）	bigint	`SELECT nanosecond(current_timestamp)` 结果示例：162000000
季度（时间戳）、季度（日期）	bigint	`SELECT quarter('2019-10-12 23:10:34.000000000')` 结果示例：4

函数	输出数据类型	描述
秒（时间戳）、秒（时间）、秒（间隔日到秒）	bigint	`SELECT second('2019-10-12 23:10:34.000000000')` 结果示例：34
周（时间戳）、周（日期）	bigint	`SELECT week('2019-10-12 23:10:34.000000000')` 结果示例：41
年中的周（时间戳）、年中的周（日期）	bigint	本周的别名
年（时间戳）、年（日期）、年（年与月的间隔）	bigint	`SELECT year('2019-10-12 23:10:34.000000000')` 结果示例：2019
一周中的一年（时间戳）、一周中的一年（日期）	bigint	`SELECT year_of_week('2019-10-12 23:10:34.000000000')` 结果示例：2019
yow（时间戳），yow（日期）	bigint	本周的别名

聚合函数

Timestream LiveAnalytics 支持以下聚合函数。

函数	输出数据类型	描述
任意 (x)	[与输入相同]	返回 x 的任意非空值（如果存在）。

函数	输出数据类型	描述
		`SELECT arbitrary(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：1
array_agg (x)	数组< [与输入相同]	返回由输入 x 个元素创建的数组。 `SELECT array_agg(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：[1,2,3,4]
平均值 (x)	double	返回所有输入值的平均值（算术平均值）。 `SELECT avg(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：2.5
bool_and（布尔值）每个（布尔值）	布尔值	TRUE如果每个输入值都是TRUE，则返回，否则返回FALSE。 `SELECT bool_and(t.c) FROM (VALUES true, true, false, true) AS t(c)` 结果示例：false

函数	输出数据类型	描述
bool_or（布尔值）	布尔值	TRUE如果有任何输入值为，则返回TRUE，否则返回FALSE。 `SELECT bool_or(t.c) FROM (VALUES true, true, false, true) AS t(c)` 结果示例：`true`
计数 (*) 计数 (x)	bigint	count (*) 返回输入行的数量。 count (x) 返回非空输入值的数量。 `SELECT count(t.c) FROM (VALUES true, true, false, true) AS t(c)` 结果示例：4
count_if (x)	bigint	返回TRUE输入值的数量。 `SELECT count_if(t.c) FROM (VALUES true, true, false, true) AS t(c)` 结果示例：3

函数	输出数据类型	描述
几何平均值 (x)	double	返回所有输入值的几何平均值。 `SELECT geometric_mean(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：2.213363839400643
max_by (x, y)	[与 x 相同]	返回所有输入值中与 y 的最大值关联的 x 值。 `SELECT max_by(t.c1, t.c2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 结果示例：d
max_by (x, y, n)	数组< [same as x] >	按照 y 的降序返回与 y 的所有输入值中最大的 n 个关联的 x 值。 `SELECT max_by(t.c1, t.c2, 2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 结果示例：[d,c]

函数	输出数据类型	描述
min_by (x, y)	[与 x 相同]	返回所有输入值中与 y 的最小值关联的 x 值。 `SELECT min_by(t.c1, t.c2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 结果示例：a
min_by (x, y, n)	数组< [same as x] >	按照 y 的升序返回与 y 的所有输入值中最小的 n 个相关联的 x 值。 `SELECT min_by(t.c1, t.c2, 2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 结果示例：[a,b]
最大值 (x)	[与输入相同]	返回所有输入值的最大值。 `SELECT max(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：4

函数	输出数据类型	描述
最大值 (x, n)	数组< [same as x] >	返回 x 的所有输入值中的 n 个最大值。 `SELECT max(t.c, 2) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：[4,3]
最小 (x)	[与输入相同]	返回所有输入值的最小值。 `SELECT min(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：1
最小值 (x, n)	数组< [same as x] >	返回 x 的所有输入值中的 n 个最小值。 `SELECT min(t.c, 2) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：[1,2]
总和 (x)	[与输入相同]	返回所有输入值的总和。 `SELECT sum(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：10

函数	输出数据类型	描述
bitwise_and_agg (x)	bigint	以 2s 补码AND表示形式返回所有输入值的位数。 `SELECT bitwise_and_agg(t.c) FROM (VALUES 1, -3) AS t(c)` 结果示例：1
bitwise_or_agg (x)	bigint	以 2s 补码表示形式返回所有输入值的按位或。 `SELECT bitwise_or_agg(t.c) FROM (VALUES 1, -3) AS t(c)` 结果示例：-3
approx_distinct (x)	bigint	返回不同输入值的大致数量。此函数提供计数 (DISTINCT x) 的近似值。如果所有输入值均为空，则返回零。此函数应生成 2.3% 的标准差，这是所有可能集合中（近似正态）误差分布的标准差。它不能保证任何特定输入集的错误都有上限。 `SELECT approx_distinct(t.c) FROM (VALUES 1, 2, 3, 4, 8) AS t(c)` 结果示例：5

函数	输出数据类型	描述
approx_distinct (x, e)	bigint	返回不同输入值的大致数量。此函数提供计数 (DISTINCTx) 的近似值。如果所有输入值均为空，则返回零。此函数产生的标准误应不大于 e，即（近似正态）误差分布在所有可能集合上的标准差。它不能保证任何特定输入集的错误都有上限。此函数的当前实现要求 e 在 [0.0040625，0.26000] 的范围内。 `SELECT approx_distinct(t.c, 0.2) FROM (VALUES 1, 2, 3, 4, 8) AS t(c)` 结果示例：5
approx_percentile (x，百分比)	[与 x 相同]	按给定百分比返回 x 的所有输入值的近似百分位数。百分比的值必须介于零和一之间，并且所有输入行的百分比值必须保持不变。 `SELECT approx_percentile(t.c, 0.4) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：2

函数	输出数据类型	描述
近似百分位数（x，百分比）	数组< [same as x] >	返回 x 的所有输入值在每个指定百分比处的近似百分位数。百分比数组的每个元素都必须介于零和一之间，并且所有输入行的数组都必须是恒定的。 `SELECT approx_percentile(t.c, ARRAY[0.1, 0.8, 0.8]) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：`[ 1,4,4 ]`
approx_percentile（x、w、百分比）	[与 x 相同]	使用每件商品的权重 w 和百分比 p，返回 x 的所有输入值的近似加权百分位数。权重必须是至少为 1 的整数值。它实际上是百分位数集中值 x 的复制计数。p 的值必须介于零和一之间，并且对于所有输入行都必须是恒定的。 `SELECT approx_percentile(t.c, 1, 0.1) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：`1`

函数	输出数据类型	描述
approx_percentile（x、w、百分比）	数组< [same as x] >	使用数组中指定的每个给定百分比处的每项权重 w 返回 x 的所有输入值的近似加权百分位数。权重必须是至少为 1 的整数值。它实际上是百分位数集中值 x 的复制计数。数组的每个元素都必须介于零和一之间，并且所有输入行的数组都必须是恒定的。 `SELECT approx_percentile(t.c, 1, ARRAY[0.1, 0.8, 0.8]) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：`[ 1,4,4 ]`
approx_percentile（x、w、百分比、精度）	[与 x 相同]	返回所有输入值 x 的近似加权百分位数，使用每件商品的权重 w 以百分比 p 为准，最大等级误差为精度。权重必须是至少为 1 的整数值。它实际上是百分位数集中值 x 的复制计数。p 的值必须介于零和一之间，并且对于所有输入行都必须是恒定的。精度必须是大于零且小于一的值，并且所有输入行的精度必须是恒定的。 `SELECT approx_percentile(t.c, 1, 0.1, 0.5) FROM (VALUES 1, 2, 3, 4) AS t(c)` 结果示例：1

函数	输出数据类型	描述
corr (y, x)	double	返回输入值的相关系数。 `SELECT corr(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 结果示例：`1.0`
covar_pop (y, x)	double	返回输入值的总体协方差。 `SELECT covar_pop(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 结果示例：`1.25`
covar_samp (y, x)	double	返回输入值的样本协方差。 `SELECT covar_samp(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 结果示例：`1.6666666666666667`

函数	输出数据类型	描述
regr_intercept (y, x)	double	返回输入值的线性回归截距。y 是因值。x 是独立值。 `SELECT regr_intercept(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 结果示例：`0.0`
regr_slope (y, x)	double	返回输入值的线性回归斜率。y 是因值。x 是独立值。 `SELECT regr_slope(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 结果示例：`1.0`
偏度 (x)	double	返回所有输入值的偏度。 `SELECT skewness(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 结果示例：`0.8978957037987335`

函数	输出数据类型	描述
stddev_pop (x)	double	返回所有输入值的总体标准差。 `SELECT stddev_pop(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 结果示例：2.4166091947189146
stddev_samp (x) stddev (x)	double	返回所有输入值的样本标准差。 `SELECT stddev_samp(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 结果示例：2.701851217221259
var_pop (x)	double	返回所有输入值的总体方差。 `SELECT var_pop(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 结果示例：5.840000000000001

函数	输出数据类型	描述
var_samp (x) 方差 (x)	double	返回所有输入值的样本方差。 `SELECT var_samp(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 结果示例：7.300000000000001

窗口函数

窗口函数对查询结果的各行进行计算。它们在HAVING子句之后但在 BY 子句ORDER之前运行。调用窗口函数需要特殊语法，使用OVER子句来指定窗口。窗口有三个组成部分：

- 分区规范，它将输入行分成不同的分区。这类似于 BY 子句如何将聚合函数GROUP的行分成不同的组。
- 排序规范，它决定了窗口函数处理输入行的顺序。
- 窗口框架，它指定给定行的滑动窗口，由函数处理给定行。如果未指定框架，则默认为 RANGE UNBOUNDEDPRECEDING，与相同RANGEBETWEENUNBOUNDEDPRECEDINGANDCURRENTROW。此框架包含从分区开头到当前行的最后一个对等体的所有行。

通过添加OVER子句，所有聚合函数都可以用作窗口函数。聚合函数是针对当前行的窗口框架内各行计算的。除了聚合函数外，Timestream 还 LiveAnalytics 支持以下排名和值函数。

函数	输出数据类型	描述
cume_dist ()	bigint	返回一组值中某个值的累积分布。结果是窗口分区窗口顺序中位于该行之前或与之对应的行数除以窗口分区中的总行数。因此，排序中的任何平局值都将计算为相同的分布值。

函数	输出数据类型	描述
dense_rank ()	bigint	返回一组值中某个值的排名。这与 rank () 类似，不同之处在于平局值不会在序列中产生间隙。
intile (n)	bigint	将每个窗口分区的行划分为 n 个存储桶，范围从 1 到最多 n。存储桶值最多相差 1。如果分区中的行数未均匀地划分为存储桶数，则从第一个存储桶开始，余数值将按每个存储桶分配一个。
百分比排名 ()	double	返回一组值中某个值的百分比排名。结果是 (r-1)/(n-1)，其中 r 是行的等级 ()，n 是窗口分区中的总行数。
等级 ()	bigint	返回一组值中某个值的排名。排名等于一加上该行前面与该行不对等的行数。因此，排序中的平局值将在序列中产生间隙。对每个窗口分区进行排名。
行号 ()	bigint	根据窗口分区中行的顺序，为每行返回一个唯一的序号，从一个开始。
第一个值 (x)	[与输入相同]	返回窗口的第一个值。此函数的作用域仅限于窗口框架。该函数采用表达式或目标作为其参数。

函数	输出数据类型	描述
最后一个值 (x)	[与输入相同]	返回窗口的最后一个值。此函数的作用域仅限于窗口框架。该函数采用表达式或目标作为其参数。
第 n 个值（x，偏移量）	[与输入相同]	返回从窗口开头开始的指定偏移量处的值。偏移量从 1 开始。偏移量可以是任何标量表达式。如果偏移量为空或大于窗口中的值数，则返回 null。偏移量为零或负是错误的。该函数将表达式或目标作为其第一个参数。
lead (x [，偏移量 [，默认值]])	[与输入相同]	返回窗口中当前行之后的偏移行处的值。偏移量从 0 开始，即当前行。偏移量可以是任何标量表达式。默认偏移量为 1。如果偏移量为空或大于窗口，则返回 default_value；如果未指定，则返回 null。该函数将表达式或目标作为其第一个参数。
延迟（x [，偏移量 [，默认值]]）	[与输入相同]	返回窗口中当前行之前偏移行的值 Offsets 从 0（即当前行）开始。偏移量可以是任何标量表达式。默认偏移量为 1。如果偏移量为空或大于窗口，则返回 default_value；如果未指定，则返回 null。该函数将表达式或目标作为其第一个参数。

示例查询

本节包括 Timestream 的查询语言 LiveAnalytics的示例用例。

主题

简单查询

以下是最近为表添加的 10 个数据点。

```
SELECT * FROM <database_name>.<table_name>
ORDER BY time DESC
LIMIT 10
```

以下是特定度量的 5 个最早的数据点。

```
SELECT * FROM <database_name>.<table_name>
WHERE measure_name = '<measure_name>'
ORDER BY time ASC
LIMIT 5
```

以下内容适用于纳秒粒度时间戳。

```
SELECT now() AS time_now
, now() - (INTERVAL '12' HOUR) AS twelve_hour_earlier -- Compatibility with ANSI SQL
, now() - 12h AS also_twelve_hour_earlier -- Convenient time interval literals
, ago(12h) AS twelve_hours_ago -- More convenience with time functionality
, bin(now(), 10m) AS time_binned -- Convenient time binning support
, ago(50ns) AS fifty_ns_ago -- Nanosecond support
, now() + (1h + 50ns) AS hour_fifty_ns_future
```

多度量记录的度量值由列名标识。单度量记录的度量值由标识measure_value::*<data_type>*，其中*<data_type>*是double、bigintboolean、或之一，varchar如中所支持的数据类型述。有关如何对度量值进行建模的更多信息，请参见单表与多表。

以下内容检索speed从多度量记录中调用的度量的值，其值为measure_name为。IoTMulti-stats

```
SELECT speed FROM <database_name>.<table_name> where measure_name = 'IoTMulti-stats'
```

以下内容从单度记录中检索double值为measure_name的值。load

```
SELECT measure_value::double FROM <database_name>.<table_name> WHERE measure_name =
 'load'
```

使用时间序列函数的查询

主题

- 示例数据集和查询

示例数据集和查询

您可以使用 Timestream LiveAnalytics 来了解和提高您的服务和应用程序的性能和可用性。以下是一个示例表，以及在该表上运行的示例查询。

该表ec2_metrics存储遥测数据，例如来自EC2实例的CPU利用率和其他指标。您可以查看下表。

时间	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1a	frontend01	CPU_利用率	35.1	null
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1a	frontend01	memory_utilization	55.3	null
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1a	frontend01	network_bytes_in	null	1500

时间	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1a	frontend01	网络字节数输出	null	6,700
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1b	frontend02	CPU_利用率	38.5	null
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1b	frontend02	memory_utilization	58.4	null
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1b	frontend02	network_bytes_in	null	23,000
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1b	frontend02	网络字节数输出	null	12000
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1c	frontend03	CPU_利用率	45.0	null
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1c	frontend03	memory_utilization	65.8	null

时间	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1c	frontend03	network_bytes_in	null	15000
2019-12-04 19:00:00.000 000000000	us-east-1	us–east–1c	frontend03	网络字节数输出	null	836,000
2019-12-04 19:00:05.000 000000 000	us-east-1	us–east–1a	frontend01	CPU_利用率	55.2	null
2019-12-04 19:00:05.000 000000 000	us-east-1	us–east–1a	frontend01	memory_utilization	75.0	null
2019-12-04 19:00:05.000 000000 000	us-east-1	us–east–1a	frontend01	network_bytes_in	null	1,245
2019-12-04 19:00:05.000 000000 000	us-east-1	us–east–1a	frontend01	网络字节数输出	null	68,432
2019-12-04 19:00:08.000 000000 000	us-east-1	us–east–1b	frontend02	CPU_利用率	65.6	null

时间	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:08.000 000 000 000	us-east-1	us–east–1b	frontend02	memory_utilization	85.3	null
2019-12-04 19:00:08.000 000 000 000	us-east-1	us–east–1b	frontend02	network_bytes_in	null	1,245
2019-12-04 19:00:08.000 000 000 000	us-east-1	us–east–1b	frontend02	网络字节数输出	null	68,432
2019-12-04 19:00:20.000 000 000 000	us-east-1	us–east–1c	frontend03	CPU_利用率	12.1	null
2019-12-04 19:00:20.000 000 000 000	us-east-1	us–east–1c	frontend03	memory_utilization	32.0	null
2019-12-04 19:00:20.000 000 000 000	us-east-1	us–east–1c	frontend03	network_bytes_in	null	1,400
2019-12-04 19:00:20.000 000 000 000	us-east-1	us–east–1c	frontend03	网络字节数输出	null	345

时间	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:10.000 000 000	us-east-1	us–east–1a	frontend01	CPU_利用率	15.3	null
2019-12-04 19:00:10.000 000 000	us-east-1	us–east–1a	frontend01	memory_utilization	35.4	null
2019-12-04 19:00:10.000 000 000	us-east-1	us–east–1a	frontend01	network_bytes_in	null	23
2019-12-04 19:00:10.000 000 000	us-east-1	us–east–1a	frontend01	网络字节数输出	null	0
2019-12-04 19:00:16.000 000 000	us-east-1	us–east–1b	frontend02	CPU_利用率	44.0	null
2019-12-04 19:00:16.000 000 000	us-east-1	us–east–1b	frontend02	memory_utilization	64.2	null
2019-12-04 19:00:16.000 000 000	us-east-1	us–east–1b	frontend02	network_bytes_in	null	1,450

时间	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:16.000 000 000	us-east-1	us–east–1b	frontend02	网络字节数输出	null	200
2019-12-04 19:00:40.000 000 000 000	us-east-1	us–east–1c	frontend03	CPU_利用率	66.4	null
2019-12-04 19:00:40.000 000 000 000	us-east-1	us–east–1c	frontend03	memory_utilization	86.3	null
2019-12-04 19:00:40.000 000 000 000	us-east-1	us–east–1c	frontend03	network_bytes_in	null	300
2019-12-04 19:00:40.000 000 000 000	us-east-1	us–east–1c	frontend03	网络字节数输出	null	423

查找过去 2 小时内特定EC2主机的平均CPU利用率、p90、p95 和 p99：

```
SELECT region, az, hostname, BIN(time, 15s) AS binned_timestamp,
    ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.9), 2) AS p90_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.95), 2) AS p95_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.99), 2) AS p99_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
```

```
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY region, hostname, az, BIN(time, 15s)
ORDER BY binned_timestamp ASC
```

确定与CPU过去 2 小时内整个机群的平均CPU利用率相比，其利用率高出 10% 或以上的EC2主机：

```
WITH avg_fleet_utilization AS (
    SELECT COUNT(DISTINCT hostname) AS total_host_count, AVG(measure_value::double) AS
 fleet_avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND time > ago(2h)
), avg_per_host_cpu AS (
    SELECT region, az, hostname, AVG(measure_value::double) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND time > ago(2h)
    GROUP BY region, az, hostname
)
SELECT region, az, hostname, avg_cpu_utilization, fleet_avg_cpu_utilization
FROM avg_fleet_utilization, avg_per_host_cpu
WHERE avg_cpu_utilization > 1.1 * fleet_avg_cpu_utilization
ORDER BY avg_cpu_utilization DESC
```

查找过去 2 小时内特定EC2主机以 30 秒为间隔的平均分箱CPU利用率：

```
SELECT BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double), 2) AS
 avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
ORDER BY binned_timestamp ASC
```

找出过去 2 小时内特定EC2主机以 30 秒为间隔分箱的平均CPU利用率，使用线性插值填充缺失值：

```
WITH binned_timeseries AS (
    SELECT hostname, BIN(time, 30s) AS binned_timestamp,
 ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
```

```
    WHERE measure_name = 'cpu_utilization'
        AND hostname = 'host-Hovjv'
        AND time > ago(2h)
    GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
    SELECT hostname,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
                SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
    FROM binned_timeseries
    GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

找出过去 2 小时内特定EC2主机以 30 秒为间隔分箱的平均CPU利用率，并根据上次执行的观测值使用插值填充缺失值：

```
WITH binned_timeseries AS (
    SELECT hostname, BIN(time, 30s) AS binned_timestamp,
 ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND hostname = 'host-Hovjv'
        AND time > ago(2h)
    GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
    SELECT hostname,
        INTERPOLATE_LOCF(
            CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
                SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
    FROM binned_timeseries
    GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

使用聚合函数的查询

以下是物联网场景示例数据集的示例，用于说明使用聚合函数的查询。

主题

示例数据

Timestream 使您能够存储和分析物联网传感器数据，例如一个或多个卡车车队的位置、油耗、速度和负载能力，从而实现有效的车队管理。以下是 iot_trucks 表的架构和一些数据，该表存储了卡车的位置、油耗、速度和装载能力等遥测数据。

时间	卡车_ID	Make	模型	实例集	燃料容量	负载容量	measure_name	measure_value::double	measure_value::varchar
2019-12-4 19:00:00.000000000	1234567	GMC	天文	Alpha	100	500	fuel_reading	65.2	null
2019-12-4 19:00:00.000000000	1234567	GMC	天文	Alpha	100	500	负载	400.0	null
2019-12-4 19:00:00.000	1234567	GMC	天文	Alpha	100	500	speed	90.2	null

时间	卡车_ID	Make	模型	实例集	燃料容量	负载容量	measure_name	measure_value::double	measure_value::varchar
00000000									
2019-12-4 19:00:00.000000000	1234567	GMC	天文	Alpha	100	500	location	null	47.6062 N，122.3321 W
2019-12-4 19:00:00.000000000	1234567	肯沃思	W900	Alpha	150	1000	fuel_reading	10.1	null
2019-12-4 19:00:00.000000000	1234567	肯沃思	W900	Alpha	150	1000	负载	950.3	null
2019-12-4 19:00:00.000000000	1234567	肯沃思	W900	Alpha	150	1000	speed	50.8	null

时间	卡车_ID	Make	模型	实例集	燃料容量	负载容量	measure_name	measure_value::double	measure_value::varchar
2019-12-4 19:00:00.000000000	1234567	肯沃思	W900	Alpha	150	1000	location	null	40.7128 北 40.7128 度，西 74.0060 度

示例查询

获取车队中每辆卡车正在监控的所有传感器属性和值的列表。

```
SELECT
    truck_id,
    fleet,
    fuel_capacity,
    model,
    load_capacity,
    make,
    measure_name
FROM "sampleDB".IoT
GROUP BY truck_id, fleet, fuel_capacity, model, load_capacity, make, measure_name
```

获取过去 24 小时内车队中每辆卡车的最新燃油读数。

```
WITH latest_recorded_time AS (
    SELECT
        truck_id,
        max(time) as latest_time
    FROM "sampleDB".IoT
    WHERE measure_name = 'fuel-reading'
    AND time >= ago(24h)
    GROUP BY truck_id
)
SELECT
    b.truck_id,
    b.fleet,
```

```
    b.make,
    b.model,
    b.time,
    b.measure_value::double as last_reported_fuel_reading
FROM
latest_recorded_time a INNER JOIN "sampleDB".IoT b
ON a.truck_id = b.truck_id AND b.time = a.latest_time
WHERE b.measure_name = 'fuel-reading'
AND b.time > ago(24h)
ORDER BY b.truck_id
```

找出在过去 48 小时内低油耗（低于 10%）的卡车：

```
WITH low_fuel_trucks AS (
    SELECT time, truck_id, fleet, make, model, (measure_value::double/
cast(fuel_capacity as double)*100) AS fuel_pct
    FROM "sampleDB".IoT
    WHERE time >= ago(48h)
    AND (measure_value::double/cast(fuel_capacity as double)*100) < 10
    AND measure_name = 'fuel-reading'
),
other_trucks AS (
SELECT time, truck_id, (measure_value::double/cast(fuel_capacity as double)*100) as
 remaining_fuel
    FROM "sampleDB".IoT
    WHERE time >= ago(48h)
    AND truck_id IN (SELECT truck_id FROM low_fuel_trucks)
    AND (measure_value::double/cast(fuel_capacity as double)*100) >= 10
    AND measure_name = 'fuel-reading'
),
trucks_that_refuelled AS (
    SELECT a.truck_id
    FROM low_fuel_trucks a JOIN other_trucks b
    ON a.truck_id = b.truck_id AND b.time >= a.time
)
SELECT DISTINCT truck_id, fleet, make, model, fuel_pct
FROM low_fuel_trucks
WHERE truck_id NOT IN (
    SELECT truck_id FROM trucks_that_refuelled
)
```

查找过去一周每辆卡车的平均负载量和最大速度：

```
SELECT
    bin(time, 1d) as binned_time,
    fleet,
    truck_id,
    make,
    model,
    AVG(
        CASE WHEN measure_name = 'load' THEN measure_value::double ELSE NULL END
    ) AS avg_load_tons,
    MAX(
        CASE WHEN measure_name = 'speed' THEN measure_value::double ELSE NULL END
    ) AS max_speed_mph
FROM "sampleDB".IoT
WHERE time >= ago(7d)
AND measure_name IN ('load', 'speed')
GROUP BY fleet, truck_id, make, model, bin(time, 1d)
ORDER BY truck_id
```

获取过去一周每辆卡车的装载效率：

```
WITH average_load_per_truck AS (
    SELECT
        truck_id,
        avg(measure_value::double)  AS avg_load
    FROM "sampleDB".IoT
    WHERE measure_name = 'load'
    AND time >= ago(7d)
    GROUP BY truck_id, fleet, load_capacity, make, model
),
truck_load_efficiency AS (
    SELECT
        a.truck_id,
        fleet,
        load_capacity,
        make,
        model,
        avg_load,
        measure_value::double,
        time,
        (measure_value::double*100)/avg_load as load_efficiency -- ,
 approx_percentile(avg_load_pct, DOUBLE '0.9')
    FROM "sampleDB".IoT a JOIN average_load_per_truck b
    ON a.truck_id = b.truck_id
```

```
    WHERE a.measure_name = 'load'
)
SELECT
    truck_id,
    time,
    load_efficiency
FROM truck_load_efficiency
ORDER BY truck_id, time
```

API参考

本节包含亚马逊 Timestream 的API参考文档。

Timestream 有两个APIs：查询和写入。

- 写入API允许您执行诸如表创建、资源标记和将记录写入 Timestream 之类的操作。
- 查询API允许您执行查询操作。

> **Note**
>
> 两者都APIs包括动 DescribeEndpoints 作。对于 “查询” 和 “写入”， DescribeEndpoints 操作是相同的。

您可以在API下面阅读有关每种内容的更多信息，以及数据类型、常见错误和参数。

> **Note**
>
> 有关所有 AWS 服务的通用错误代码，请参阅 Supp AWS ort 部分。

主题

- 操作
- 数据类型
- 常见错误
- 常见参数

操作

Amazon Timestream Write 支持以下操作：

- CreateBatchLoadTask
- CreateDatabase
- CreateTable
- DeleteDatabase
- DeleteTable
- DescribeBatchLoadTask
- DescribeDatabase
- DescribeEndpoints
- DescribeTable
- ListBatchLoadTasks
- ListDatabases
- ListTables
- ListTagsForResource
- ResumeBatchLoadTask
- TagResource
- UntagResource
- UpdateDatabase
- UpdateTable
- WriteRecords

亚马逊 Timestream 查询支持以下操作：

- CancelQuery
- CreateScheduledQuery
- DeleteScheduledQuery
- DescribeAccountSettings
- DescribeEndpoints
- DescribeScheduledQuery

- ExecuteScheduledQuery
- ListScheduledQueries
- ListTagsForResource
- PrepareQuery
- Query
- TagResource
- UntagResource
- UpdateAccountSettings
- UpdateScheduledQuery

亚马逊 Timestream Write

Amazon Timestream Write 支持以下操作：

- CreateBatchLoadTask
- CreateDatabase
- CreateTable
- DeleteDatabase
- DeleteTable
- DescribeBatchLoadTask
- DescribeDatabase
- DescribeEndpoints
- DescribeTable
- ListBatchLoadTasks
- ListDatabases
- ListTables
- ListTagsForResource
- ResumeBatchLoadTask
- TagResource
- UntagResource
- UpdateDatabase
- UpdateTable

- WriteRecords

CreateBatchLoadTask

服务: Amazon Timestream Write

创建新的 Timestream 批量加载任务。批量加载任务处理来自 S3 位置CSV源的数据并写入 Timestream 表。从源到目标的映射是在批量加载任务中定义的。错误和事件将写入 S3 位置的报告中。对于报告，如果未指定 AWS KMS 密钥，则在可选的情况下SSE_S3，将使用 S3 托管密钥对报告进行加密。否则，将引发错误。有关更多信息，请参阅 AWS 托管式密钥。适用服务配额。有关详细信息，请参阅代码示例。

请求语法

```
{
   "ClientToken": "string",
   "DataModelConfiguration": {
      "DataModel": {
         "DimensionMappings": [
            {
               "DestinationColumn": "string",
               "SourceColumn": "string"
            }
         ],
         "MeasureNameColumn": "string",
         "MixedMeasureMappings": [
            {
               "MeasureName": "string",
               "MeasureValueType": "string",
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "SourceColumn": "string",
               "TargetMeasureName": "string"
            }
         ],
         "MultiMeasureMappings": {
            "MultiMeasureAttributeMappings": [
               {
                  "MeasureValueType": "string",
                  "SourceColumn": "string",
                  "TargetMultiMeasureAttributeName": "string"
```

```
                }
            ],
            "TargetMultiMeasureName": "string"
         },
         "TimeColumn": "string",
         "TimeUnit": "string"
      },
      "DataModelS3Configuration": {
         "BucketName": "string",
         "ObjectKey": "string"
      }
   },
   "DataSourceConfiguration": {
      "CsvConfiguration": {
         "ColumnSeparator": "string",
         "EscapeChar": "string",
         "NullValue": "string",
         "QuoteChar": "string",
         "TrimWhiteSpace": boolean
      },
      "DataFormat": "string",
      "DataSourceS3Configuration": {
         "BucketName": "string",
         "ObjectKeyPrefix": "string"
      }
   },
   "RecordVersion": number,
   "ReportConfiguration": {
      "ReportS3Configuration": {
         "BucketName": "string",
         "EncryptionOption": "string",
         "KmsKeyId": "string",
         "ObjectKeyPrefix": "string"
      }
   },
   "TargetDatabaseName": "string",
   "TargetTableName": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ClientToken

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

必需：否

DataModelConfiguration

类型：DataModelConfiguration 对象

必需：否

DataSourceConfiguration

定义有关批量加载任务的数据源的配置详细信息。

类型：DataSourceConfiguration 对象

必需：是

RecordVersion

类型：长整型

必需：否

ReportConfiguration

报告批量加载任务的配置。其中包含有关错误报告的存储位置的详细信息。

类型：ReportConfiguration 对象

必需：是

TargetDatabaseName

批量加载任务的目标时间流数据库。

类型：字符串

模式：`[a-zA-Z0-9_.-]+`

必需：是

TargetTableName

批量加载任务的目标时间流表。

类型：字符串

模式：[a-zA-Z0-9_.-]+

必需：是

响应语法

```
{
   "TaskId": "string"
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

TaskId

批量加载任务的 ID。

类型：字符串

长度约束：最小长度为 3。最大长度为 32。

模式：[A-Z0-9]+

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

ConflictException

Timestream 无法处理此请求，因为它包含已经存在的资源。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ServiceQuotaExceededException

已超过该账户的资源实例配额。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面

- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

CreateDatabase

服务: Amazon Timestream Write

创建新的 Timestream 数据库。如果未指定 AWS KMS 密钥，则将使用位于您账户中的 Timestream 托管 AWS KMS 密钥对数据库进行加密。有关更多信息，请参阅 AWS 托管式密钥。适用服务配额。有关详细信息，请参阅代码示例。

请求语法

```
{
   "DatabaseName": "string",
   "KmsKeyId": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

DatabaseName

Timestream 数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

模式：[a-zA-Z0-9_.-]+

必需：是

KmsKeyId

数据库的 AWS KMS 密钥。如果未指定 AWS KMS 密钥，则将使用位于您账户中的 Timestream 托管 AWS KMS 密钥对数据库进行加密。有关更多信息，请参阅 AWS 托管式密钥。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

Tags

用于标记表的键值对列表。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

必需：否

响应语法

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Database

新创建的时间流数据库。

类型：Database 对象

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

ConflictException

Timestream 无法处理此请求，因为它包含已经存在的资源。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ServiceQuotaExceededException

已超过该账户的资源实例配额。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

CreateTable

服务: Amazon Timestream Write

向账户中的现有数据库添加新表。在中 AWS 账户，如果表名位于同一个数据库中，则每个区域内的表名必须至少是唯一的。如果表位于不同的数据库中，则同一区域中的表名可能相同。创建表时，必须指定表名称、数据库名称和保留属性。适用服务配额。请参阅代码示例，了解详细信息。

请求语法

```
{
   "DatabaseName": "string",
   "MagneticStoreWriteProperties": {
      "EnableMagneticStoreWrites": boolean,
      "MagneticStoreRejectedDataLocation": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      }
   },
   "RetentionProperties": {
      "MagneticStoreRetentionPeriodInDays": number,
      "MemoryStoreRetentionPeriodInHours": number
   },
   "Schema": {
      "CompositePartitionKey": [
         {
            "EnforcementInRecord": "string",
            "Name": "string",
            "Type": "string"
         }
      ]
   },
   "TableName": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

DatabaseName

Timestream 数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

模式：[a-zA-Z0-9_.-]+

必需：是

MagneticStoreWriteProperties

包含启用磁存储写入时要在表上设置的属性。

类型：MagneticStoreWriteProperties 对象

必需：否

RetentionProperties

您的时间序列数据必须存储在存储器和磁性存储器中的持续时间。

类型：RetentionProperties 对象

必需：否

Schema

表的架构。

类型：Schema 对象

必需：否

TableName

Timestream 表的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

模式：[a-zA-Z0-9_.-]+

必需：是

Tags

用于标记表的键值对列表。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

必需：否

响应语法

```
{
   "Table": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
```

```
                "EnforcementInRecord": "string",
                "Name": "string",
                "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Table

新创建的时间流表。

类型：Table 对象

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

ConflictException

Timestream 无法处理此请求，因为它包含已经存在的资源。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ServiceQuotaExceededException

已超过该账户的资源实例配额。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2

- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DeleteDatabase

服务: Amazon Timestream Write

删除给定的时间流数据库。这是一项不可逆的操作。删除数据库后，无法恢复其表中的时间序列数据。

> **Note**
>
> 必须先删除数据库中的所有表，否则将引发 ValidationException 错误。
> 由于分布式重试的性质，该操作可以返回成功或。 ResourceNotFoundException客户应将它们视为等同的。

请参阅代码示例，了解详细信息。

请求语法

```
{
   "DatabaseName": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

DatabaseName

要删除的时间流数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2

- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DeleteTable
服务: Amazon Timestream Write

删除给定的时间流表。这是一次不可逆的手术。删除 Timestream 数据库表后，存储在该表中的时间序列数据将无法恢复。

> **Note**
>
> 由于分布式重试的性质，该操作可以返回 success 或。 ResourceNotFoundException客户应将它们视为等同的。

请参阅代码示例，了解详细信息。

请求语法

```
{
   "DatabaseName": "string",
   "TableName": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

DatabaseName

要从中删除 Timestream 数据库的数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

TableName

要删除的时间流表的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DescribeBatchLoadTask

服务: Amazon Timestream Write

返回有关批量加载任务的信息，包括配置、映射、进度和其他详细信息。适用服务配额。请参阅代码示例，了解详细信息。

请求语法

```
{
   "TaskId": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

TaskId

批量加载任务的 ID。

类型：字符串

长度约束：最小长度为 3。最大长度为 32。

模式：[A-Z0-9]+

必需：是

响应语法

```
{
   "BatchLoadTaskDescription": {
      "CreationTime": number,
      "DataModelConfiguration": {
         "DataModel": {
            "DimensionMappings": [
               {
                  "DestinationColumn": "string",
                  "SourceColumn": "string"
               }
            ],
```

```
            "MeasureNameColumn": "string",
            "MixedMeasureMappings": [
               {
                  "MeasureName": "string",
                  "MeasureValueType": "string",
                  "MultiMeasureAttributeMappings": [
                     {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                     }
                  ],
                  "SourceColumn": "string",
                  "TargetMeasureName": "string"
               }
            ],
            "MultiMeasureMappings": {
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "TargetMultiMeasureName": "string"
            },
            "TimeColumn": "string",
            "TimeUnit": "string"
         },
         "DataModelS3Configuration": {
            "BucketName": "string",
            "ObjectKey": "string"
         }
      },
      "DataSourceConfiguration": {
         "CsvConfiguration": {
            "ColumnSeparator": "string",
            "EscapeChar": "string",
            "NullValue": "string",
            "QuoteChar": "string",
            "TrimWhiteSpace": boolean
         },
         "DataFormat": "string",
         "DataSourceS3Configuration": {
```

```
            "BucketName": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "ErrorMessage": "string",
      "LastUpdatedTime": number,
      "ProgressReport": {
         "BytesMetered": number,
         "FileFailures": number,
         "ParseFailures": number,
         "RecordIngestionFailures": number,
         "RecordsIngested": number,
         "RecordsProcessed": number
      },
      "RecordVersion": number,
      "ReportConfiguration": {
         "ReportS3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "ResumableUntil": number,
      "TargetDatabaseName": "string",
      "TargetTableName": "string",
      "TaskId": "string",
      "TaskStatus": "string"
   }
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

BatchLoadTaskDescription

批量加载任务的描述。

类型：BatchLoadTaskDescription 对象

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2

- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DescribeDatabase

服务: Amazon Timestream Write

返回有关数据库的信息，包括数据库名称、创建数据库的时间以及在数据库中找到的表总数。适用服务配额。请参阅代码示例，了解详细信息。

请求语法

```
{
   "DatabaseName": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

DatabaseName

Timestream 数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

响应语法

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Database

Timestream 表的名称。

类型：Database 对象

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DescribeEndpoints
服务: Amazon Timestream Write

返回用于API调用 Timestream 的可用端点列表。此API操作可通过 “写入” 和 “查询” 进行APIs。

由于 Timestream SDKs 旨在透明地与服务的架构（包括服务端点的管理和映射）配合使用，因此我们不建议您使用此API操作，除非：

- 您正在使用带有 T imestrea AWS PrivateLink m 的VPC端点 ()
- 您的应用程序使用的编程语言尚不SDK支持
- 您需要更好地控制客户端实现

有关如何以及何时使用和实施的详细信息 DescribeEndpoints，请参阅端点发现模式。

响应语法

```
{
   "Endpoints": [
      {
         "Address": "string",
         "CachePeriodInMinutes": number
      }
   ]
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Endpoints

DescribeEndpoints发出请求时会返回一个Endpoints对象。

类型：Endpoint 对象数组

错误

有关所有操作的常见错误的信息，请参阅常见错误。

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DescribeTable
服务: Amazon Timestream Write

返回有关表的信息，包括表名、数据库名称、内存存储和磁性存储的保留时间。适用服务配额。请参阅代码示例，了解详细信息。

请求语法

```
{
   "DatabaseName": "string",
   "TableName": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

DatabaseName

Timestream 数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

TableName

Timestream 表的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

响应语法

```
{
   "Table": {
```

```
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
               "EnforcementInRecord": "string",
               "Name": "string",
               "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Table

时间流表。

类型：Table 对象

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET

- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

ListBatchLoadTasks
服务: Amazon Timestream Write

提供批量加载任务的列表，以及名称、状态、任务可恢复到何时以及其他详细信息。请参阅代码示例，了解详细信息。

请求语法

```
{
   "MaxResults": number,
   "NextToken": "string",
   "TaskStatus": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

MaxResults

要在输出中返回的项目总数。如果可用项目总数大于指定值，则输出中会提供 a NextToken 。要恢复分页，请提供该 NextToken 值作为后续API调用的参数。

类型：整数

有效范围：最小值为 1。最大值为 100。

必需：否

NextToken

指定从何处开始分页的令牌。这是之前截 NextToken 断的响应。

类型：字符串

必需：否

TaskStatus

批量加载任务的状态。

类型：字符串

有效值：CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME

必需：否

响应语法

```
{
   "BatchLoadTasks": [
      {
         "CreationTime": number,
         "DatabaseName": "string",
         "LastUpdatedTime": number,
         "ResumableUntil": number,
         "TableName": "string",
         "TaskId": "string",
         "TaskStatus": "string"
      }
   ],
   "NextToken": "string"
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

BatchLoadTasks

批量加载任务详细信息列表。

类型：BatchLoadTask 对象数组

NextToken

指定从何处开始分页的令牌。提供下一个 ListBatchLoadTasksRequest。

类型：字符串

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python

- AWS SDK适用于 Ruby V3

ListDatabases
服务: Amazon Timestream Write

返回您的时间流数据库列表。适用服务配额。请参阅代码示例，了解详细信息。

请求语法

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

MaxResults

要在输出中返回的项目总数。如果可用项目总数大于指定值，则输出中会提供 a NextToken 。要恢复分页，请提供该 NextToken 值作为后续API调用的参数。

类型：整数

有效范围：最小值为 1。最大值为 20。

必需：否

NextToken

分页标记。要恢复分页，请提供该 NextToken 值作为后续API调用的参数。

类型：字符串

必需：否

响应语法

```
{
   "Databases": [
      {
```

```
            "Arn": "string",
            "CreationTime": number,
            "DatabaseName": "string",
            "KmsKeyId": "string",
            "LastUpdatedTime": number,
            "TableCount": number
        }
    ],
    "NextToken": "string"
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Databases

数据库名称列表。

类型：Database 对象数组

NextToken

分页标记。当响应被截断时，将返回此参数。

类型：字符串

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

ListTables
服务: Amazon Timestream Write

提供表列表以及每个表的名称、状态和保留属性。请参阅代码示例，了解详细信息。

请求语法

```
{
   "DatabaseName": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

DatabaseName

Timestream 数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：否

MaxResults

要在输出中返回的项目总数。如果可用项目总数大于指定值，则输出中会提供 a NextToken 。要恢复分页，请提供该 NextToken 值作为后续API调用的参数。

类型：整数

有效范围：最小值为 1。最大值为 20。

必需：否

NextToken

分页标记。要恢复分页，请提供该 NextToken 值作为后续API调用的参数。

类型：字符串

必需：否

响应语法

```
{
   "NextToken": "string",
   "Tables": [
      {
         "Arn": "string",
         "CreationTime": number,
         "DatabaseName": "string",
         "LastUpdatedTime": number,
         "MagneticStoreWriteProperties": {
            "EnableMagneticStoreWrites": boolean,
            "MagneticStoreRejectedDataLocation": {
               "S3Configuration": {
                  "BucketName": "string",
                  "EncryptionOption": "string",
                  "KmsKeyId": "string",
                  "ObjectKeyPrefix": "string"
               }
            }
         },
         "RetentionProperties": {
            "MagneticStoreRetentionPeriodInDays": number,
            "MemoryStoreRetentionPeriodInHours": number
         },
         "Schema": {
            "CompositePartitionKey": [
               {
                  "EnforcementInRecord": "string",
                  "Name": "string",
                  "Type": "string"
               }
            ]
         },
         "TableName": "string",
         "TableStatus": "string"
      }
   ]
```

```
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

NextToken

指定从何处开始分页的令牌。这是之前截 NextToken 断的响应。

类型：字符串

Tables

表格列表。

类型：Table 对象数组

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

ListTagsForResource
服务: Amazon Timestream Write

列出时间流资源上的所有标签。

请求语法

```
{
   "ResourceARN": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ResourceARN

带有待列出标签的时间流资源。此值是 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：长度下限为 1。最大长度为 1011。

必需：是

响应语法

```
{
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Tags

当前与 Timestream 资源关联的标签。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET

- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

ResumeBatchLoadTask
服务: Amazon Timestream Write

请求语法

```
{
   "TaskId": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

TaskId

要恢复的批量加载任务的 ID。

类型：字符串

长度约束：最小长度为 3。最大长度为 32。

模式：[A-Z0-9]+

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python

- AWS SDK适用于 Ruby V3

TagResource
服务: Amazon Timestream Write

将一组标签与时间流资源相关联。然后，您可以激活这些用户定义的标签，使它们显示在账单和成本管理控制台上，用于跟踪成本分配。

请求语法

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ResourceARN

标识应向其添加标签的时间流资源。此值是 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：长度下限为 1。最大长度为 1011。

必需：是

Tags

要分配给 Timestream 资源的标签。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ServiceQuotaExceededException

已超过该账户的资源实例配额。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++

- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

UntagResource
服务: Amazon Timestream Write

从 Timestream 资源中移除标签的关联。

请求语法

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ResourceARN

将从中移除标签的时间流资源。此值是 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：长度下限为 1。最大长度为 1011。

必需：是

TagKeys

标签密钥列表。密钥属于此列表成员的资源的现有标签将从 Timestream 资源中删除。

类型：字符串数组

数组成员：最少 0 个物品。最多 200 项。

长度限制：长度下限为 1。长度上限为 128。

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ServiceQuotaExceededException

已超过该账户的资源实例配额。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2

- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

UpdateDatabase
服务: Amazon Timestream Write

修改现有数据库的 AWS KMS 密钥。更新数据库时，必须指定数据库名称和要使用的新 AWS KMS 密钥的标识符 (KmsKeyId)。如果有任何并发UpdateDatabase请求，则第一个写入者获胜。

请参阅代码示例，了解详细信息。

请求语法

```
{
   "DatabaseName": "string",
   "KmsKeyId": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

DatabaseName

数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

KmsKeyId

用于加密存储在数据库中的数据的新 AWS KMS 密钥 (KmsKeyId) 的标识符。如果KmsKeyId当前在数据库中注册的与请求KmsKeyId中的相同，则不会有任何更新。

您可以使用以下任一KmsKeyId方法来指定：

- 密钥 ID：1234abcd-12ab-34cd-56ef-1234567890ab
- 密钥ARN：arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
- 别名：alias/ExampleAlias

- 别名ARN：`arn:aws:kms:us-east-1:111122223333:alias/ExampleAlias`

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

响应语法

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Database

表格的顶级容器。数据库和表是 Amazon Timestream 中的基本管理概念。数据库中的所有表都使用相同的 AWS KMS 密钥进行加密。

类型：Database 对象

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ServiceQuotaExceededException

已超过该账户的资源实例配额。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++

- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

UpdateTable

服务: Amazon Timestream Write

修改 Timestream 表的内存存储和磁性存储的保留时间。请注意，保留期限的更改会立即生效。例如，如果内存存储的保留期最初设置为 2 小时，然后更改为 24 小时，则内存存储将能够保存 24 小时的数据，但在进行此更改后 22 小时将填充 24 小时的数据。Timestream 不会从磁性存储中检索数据来填充内存存储。

请参阅代码示例，了解详细信息。

请求语法

```
{
   "DatabaseName": "string",
   "MagneticStoreWriteProperties": {
      "EnableMagneticStoreWrites": boolean,
      "MagneticStoreRejectedDataLocation": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      }
   },
   "RetentionProperties": {
      "MagneticStoreRetentionPeriodInDays": number,
      "MemoryStoreRetentionPeriodInHours": number
   },
   "Schema": {
      "CompositePartitionKey": [
         {
            "EnforcementInRecord": "string",
            "Name": "string",
            "Type": "string"
         }
      ]
   },
   "TableName": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

DatabaseName

Timestream 数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

MagneticStoreWriteProperties

包含启用磁存储写入时要在表上设置的属性。

类型：MagneticStoreWriteProperties 对象

必需：否

RetentionProperties

存储器和磁性存储器的保留时间。

类型：RetentionProperties 对象

必需：否

Schema

表的架构。

类型：Schema 对象

必需：否

TableName

Timestream 表的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

响应语法

```
{
   "Table": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
               "EnforcementInRecord": "string",
               "Name": "string",
               "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Table

更新后的时间流表。

类型：Table 对象

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

WriteRecords
服务: Amazon Timestream Write

使您能够将时间序列数据写入 Timestream。您可以指定要插入到系统的单个数据点或一批数据点。Timestream 为您提供了一个灵活的架构，它可以根据您在调用写入数据库时指定的数据点的维度名称和数据类型，自动检测 Timestream 表的列名和数据类型。

Timestream 支持最终一致性读取语义。这意味着，当您在将一批数据写入 Timestream 后立即查询数据时，查询结果可能无法反映最近完成的写入操作的结果。结果可能还包括一些陈旧的数据。如果您在短时间后重复查询请求，则结果应返回最新数据。适用服务配额。

请参阅代码示例，了解详细信息。

Upserts

您可以在`WriteRecords`请求中使用`Version`参数来更新数据点。Timestream 会跟踪每条记录的版本号。 `Version`1当请求中没有为记录指定时，默认为。Timestream 会更新现有记录的度量值以及该记录的度量值更高的写入请求`Version`时该记录的度`Version`量值。当 Timestream 收到的度量值与现有记录的度量值相同的更新请求时，如果 Timestream 的现有值大于的`Version`现有值`Version`，它仍会更新。只要的值`Version`不断增加，就可以根据需要多次更新数据点。

例如，假设你写了一条新记录，但没有在请求`Version`中注明。Timestream 存储此记录，并将其设置`Version`为。1现在，假设您尝试使用具有不同度量值的相同记录的`WriteRecords`请求来更新此记录，但与以前一样，没有提供`Version`。在这种情况下，Timestream 将拒绝此更新，`RejectedRecordsException`因为更新的记录的版本不大于 “版本” 的现有值。

但是，如果您要在`Version`设置为的情况下重新发送更新请求2，Timestream 将成功更新记录的值，`Version`并将设置为。2接下来，假设您发送的`WriteRecords`请求具有相同的记录和相同的度量值，但`Version`设置为3。在这种情况下，Timestream 只会更新`Version`为。3任何进一步的更新都需要发送大于的版本号3，否则更新请求将收到`RejectedRecordsException`。

请求语法

```
{
   "CommonAttributes": {
      "Dimensions": [
         {
            "DimensionValueType": "string",
            "Name": "string",
            "Value": "string"
         }
      ],
```

```
      "MeasureName": "string",
      "MeasureValue": "string",
      "MeasureValues": [
         {
            "Name": "string",
            "Type": "string",
            "Value": "string"
         }
      ],
      "MeasureValueType": "string",
      "Time": "string",
      "TimeUnit": "string",
      "Version": number
   },
   "DatabaseName": "string",
   "Records": [
      {
         "Dimensions": [
            {
               "DimensionValueType": "string",
               "Name": "string",
               "Value": "string"
            }
         ],
         "MeasureName": "string",
         "MeasureValue": "string",
         "MeasureValues": [
            {
               "Name": "string",
               "Type": "string",
               "Value": "string"
            }
         ],
         "MeasureValueType": "string",
         "Time": "string",
         "TimeUnit": "string",
         "Version": number
      }
   ],
   "TableName": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

CommonAttributes

包含请求中所有记录共享的常用度量、维度、时间和版本属性的记录。当数据写入 Timestream 时，指定的度量和维度属性将与记录对象中的度量和维度属性合并。尺寸不得重叠，否则ValidationException会被抛出。换句话说，记录必须包含具有唯一名称的维度。

类型：Record 对象

必需：否

DatabaseName

Timestream 数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

Records

包含每个时间序列数据点的唯一度量、维度、时间和版本属性的记录数组。

类型：Record 对象数组

数组成员：最少 1 个项目。最多 100 个项目。

必需：是

TableName

Timestream 表的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：是

响应语法

```
{
   "RecordsIngested": {
      "MagneticStore": number,
      "MemoryStore": number,
      "Total": number
   }
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

RecordsIngested

有关此请求获取的记录的信息。

类型：RecordsIngested 对象

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，Timestream 无法完全处理此请求。

HTTP状态码：500

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

RejectedRecordsException

WriteRecords 在以下情况下会抛出这个异常：

- 具有重复数据的记录，其中有多个具有相同维度、时间戳和度量名称的记录，但是：
  - 测量值不同
  - 请求中不存在版本，或者新记录中版本的值等于或小于现有值

  在这种情况下，如果 Timestream 拒绝数据，则`RejectedRecords`响应中的`ExistingVersion`字段将指示当前记录的版本。要强制更新，您可以重新发送请求，并将该记录的版本设置为大于。`ExistingVersion`
- 时间戳超出内存存储保留期限的记录。
- 维度或度量超过时间流定义限制的记录。

有关更多信息，请参阅《亚马逊 Timestream 开发者指南》中的配额。

HTTP状态码：400

ResourceNotFoundException

该操作试图访问一个不存在的资源。可能未正确指定资源，或者其状态可能不正确ACTIVE。

HTTP状态码：400

ThrottlingException

用户发出的请求太多，超过了服务配额。请求已被阻止。

HTTP状态码：400

ValidationException

无效或格式错误的请求。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++

- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

亚马逊 Timestream 查询

亚马逊 Timestream 查询支持以下操作：

- CancelQuery
- CreateScheduledQuery
- DeleteScheduledQuery
- DescribeAccountSettings
- DescribeEndpoints
- DescribeScheduledQuery
- ExecuteScheduledQuery
- ListScheduledQueries
- ListTagsForResource
- PrepareQuery
- Query
- TagResource
- UntagResource
- UpdateAccountSettings
- UpdateScheduledQuery

CancelQuery

服务: Amazon Timestream Query

取消已发出的查询。只有在发出取消请求之前查询尚未完成运行时，才会提供取消功能。由于取消是一个等值操作，因此后续的取消请求将返回CancellationMessage，表示查询已被取消。请参阅代码示例，了解详细信息。

请求语法

```
{
   "QueryId": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

QueryId

需要取消的查询的 ID。 QueryID作为查询结果的一部分返回。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：[a-zA-Z0-9]+

必需：是

响应语法

```
{
   "CancellationMessage": "string"
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

CancellationMessage

当QueryId已发出指定的查询CancelQuery请求时，将返回 A。CancellationMessage

类型：字符串

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面

- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

CreateScheduledQuery

服务: Amazon Timestream Query

创建一个计划查询，该查询将按照配置的计划代表您运行。Timestream 担任执行角色（作为 ScheduledQueryExecutionRoleArn 参数的一部分提供）以运行查询。您可以使用 NotificationConfiguration 参数为计划的查询操作配置通知。

请求语法

```
{
   "ClientToken": "string",
   "ErrorReportConfiguration": {
      "S3Configuration": {
         "BucketName": "string",
         "EncryptionOption": "string",
         "ObjectKeyPrefix": "string"
      }
   },
   "KmsKeyId": "string",
   "Name": "string",
   "NotificationConfiguration": {
      "SnsConfiguration": {
         "TopicArn": "string"
      }
   },
   "QueryString": "string",
   "ScheduleConfiguration": {
      "ScheduleExpression": "string"
   },
   "ScheduledQueryExecutionRoleArn": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ],
   "TargetConfiguration": {
      "TimestreamConfiguration": {
         "DatabaseName": "string",
         "DimensionMappings": [
            {
               "DimensionValueType": "string",
               "Name": "string"
```

```
            }
         ],
         "MeasureNameColumn": "string",
         "MixedMeasureMappings": [
            {
               "MeasureName": "string",
               "MeasureValueType": "string",
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "SourceColumn": "string",
               "TargetMeasureName": "string"
            }
         ],
         "MultiMeasureMappings": {
            "MultiMeasureAttributeMappings": [
               {
                  "MeasureValueType": "string",
                  "SourceColumn": "string",
                  "TargetMultiMeasureAttributeName": "string"
               }
            ],
            "TargetMultiMeasureName": "string"
         },
         "TableName": "string",
         "TimeColumn": "string"
      }
   }
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ClientToken

使用 a ClientToken 会调用 CreateScheduledQuery idempotent，换句话说，重复发出相同的请求将产生相同的结果。发出多个相同的 CreateScheduledQuery 请求与发出单个请求的效果相同。

- CreateScheduledQuery 如果调用时不带有ClientToken，则查询SDK会ClientToken代表您生成。
- 8 小时后，具有相同 ClientToken 的任何请求都将视为新请求。

类型：字符串

长度限制：最小长度为 32。长度上限为 128。

必需：否

ErrorReportConfiguration

错误报告的配置。如果写入查询结果时遇到问题，将生成错误报告。

类型：ErrorReportConfiguration 对象

必需：是

KmsKeyId

用于加密预定查询资源的 Amazon 密KMS钥（静态）。如果未指定亚马逊KMS密钥，则计划查询资源将使用 Timestream 拥有的亚马逊KMS密钥进行加密。要指定KMS密钥，请使用密钥 ID、密钥 ARN、别名或别名ARN。使用别名时，应为名称加上 alias/前缀

如果SSE_KMS用作加密类型，则 ErrorReportConfiguration 使用相同的 KmsKeyId 加密类型对错误报告进行静态加密。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

Name

计划查询的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+

必需：是

NotificationConfiguration

计划查询的通知配置。查询运行完成、状态更新或您将其删除时，Timestream 将发送通知。

类型：NotificationConfiguration 对象

必需：是

QueryString

要运行的查询字符串。参数名称可以采用查询字符串 @ 字符后跟标识符的形式指定。指定的参数 @scheduled_runtime 将保留，并可在查询中使用，以获取查询计划运行的时间。

根据 ScheduleConfiguration 参数计算的时间戳将是每次查询运行的 p @scheduled_runtime aramater 的值。例如，考虑在 2021-12-01 00:00:00 执行的计划查询的实例。对于此实例，@scheduled_runtime 参数在调用查询时初始化的时间戳为 2021-12-01 00:00:00。

类型：字符串

长度限制：长度下限为 1。长度上限为 262144。

必需：是

ScheduleConfiguration

查询的计划配置。

类型：ScheduleConfiguration 对象

必需：是

ScheduledQueryExecutionRoleArn

表示运行计划查询时 Timestream 将扮演的IAM角色。ARN

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

Tags

用于标记计划查询的键值对列表。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

必需：否

TargetConfiguration

用于写入查询结果的配置。

类型：TargetConfiguration 对象

必需：否

响应语法

```
{
   "Arn": "string"
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Arn

ARN用于创建的计划查询。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

ConflictException

无法对已取消的查询进行轮询结果。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ServiceQuotaExceededException

您已超过服务配额。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2

- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DeleteScheduledQuery
服务: Amazon Timestream Query

删除给定的计划查询。这是一项不可逆的操作。

请求语法

```
{
   "ScheduledQueryArn": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ScheduledQueryArn

计划查询的 ARN。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

找不到请求的资源。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DescribeAccountSettings
服务: Amazon Timestream Query

描述您的账户的设置，包括查询定价模型以及TCUs该服务可用于您的查询工作负载的配置的最大值。

您只需按用于工作负载的计算单位的持续时间付费。

响应语法

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

MaxQueryTCU

该服务在任何时间点为你的查询提供服务时将使用的最大 Timestream 计算单元 (TCUs) 数。

类型：整数

QueryPricingModel

您的账户中查询的定价模式。

类型：字符串

有效值：BYTES_SCANNED | COMPUTE_UNITS

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DescribeEndpoints
服务: Amazon Timestream Query

DescribeEndpoints 返回用于API调用 Timestream 的可用端点列表。API这可通过 “写入” 和 “查询” 获得。

由于 Timestream SDKs 旨在透明地与服务的架构配合使用，包括服务端点的管理和映射，因此不建议您使用它，除非：API

- 您正在将VPC端点 (AWS PrivateLink) 与 Timestream 配合使用
- 您的应用程序使用的编程语言尚不SDK支持
- 您需要更好地控制客户端实现

有关如何以及何时使用和实施的详细信息 DescribeEndpoints，请参阅端点发现模式。

响应语法

```
{
   "Endpoints": [
      {
         "Address": "string",
         "CachePeriodInMinutes": number
      }
   ]
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Endpoints

DescribeEndpoints发出请求时会返回一个Endpoints对象。

类型：Endpoint 对象数组

错误

有关所有操作的常见错误的信息，请参阅常见错误。

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

DescribeScheduledQuery
服务: Amazon Timestream Query

提供有关计划查询的详细信息。

请求语法

```
{
   "ScheduledQueryArn": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ScheduledQueryArn

计划查询的 ARN。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

响应语法

```
{
   "ScheduledQuery": {
      "Arn": "string",
      "CreationTime": number,
      "ErrorReportConfiguration": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "KmsKeyId": "string",
      "LastRunSummary": {
         "ErrorReportLocation": {
```

```
            "S3ReportLocation": {
               "BucketName": "string",
               "ObjectKey": "string"
            }
         },
         "ExecutionStats": {
            "BytesMetered": number,
            "CumulativeBytesScanned": number,
            "DataWrites": number,
            "ExecutionTimeInMillis": number,
            "QueryResultRows": number,
            "RecordsIngested": number
         },
         "FailureReason": "string",
         "InvocationTime": number,
         "QueryInsightsResponse": {
            "OutputBytes": number,
            "OutputRows": number,
            "QuerySpatialCoverage": {
               "Max": {
                  "PartitionKey": [ "string" ],
                  "TableArn": "string",
                  "Value": number
               }
            },
            "QueryTableCount": number,
            "QueryTemporalRange": {
               "Max": {
                  "TableArn": "string",
                  "Value": number
               }
            }
         },
         "RunStatus": "string",
         "TriggerTime": number
      },
      "Name": "string",
      "NextInvocationTime": number,
      "NotificationConfiguration": {
         "SnsConfiguration": {
            "TopicArn": "string"
         }
      },
      "PreviousInvocationTime": number,
```

```
      "QueryString": "string",
      "RecentlyFailedRuns": [
         {
            "ErrorReportLocation": {
               "S3ReportLocation": {
                  "BucketName": "string",
                  "ObjectKey": "string"
               }
            },
            "ExecutionStats": {
               "BytesMetered": number,
               "CumulativeBytesScanned": number,
               "DataWrites": number,
               "ExecutionTimeInMillis": number,
               "QueryResultRows": number,
               "RecordsIngested": number
            },
            "FailureReason": "string",
            "InvocationTime": number,
            "QueryInsightsResponse": {
               "OutputBytes": number,
               "OutputRows": number,
               "QuerySpatialCoverage": {
                  "Max": {
                     "PartitionKey": [ "string" ],
                     "TableArn": "string",
                     "Value": number
                  }
               },
               "QueryTableCount": number,
               "QueryTemporalRange": {
                  "Max": {
                     "TableArn": "string",
                     "Value": number
                  }
               }
            },
            "RunStatus": "string",
            "TriggerTime": number
         }
      ],
      "ScheduleConfiguration": {
         "ScheduleExpression": "string"
      },
```

```
      "ScheduledQueryExecutionRoleArn": "string",
      "State": "string",
      "TargetConfiguration": {
         "TimestreamConfiguration": {
            "DatabaseName": "string",
            "DimensionMappings": [
               {
                  "DimensionValueType": "string",
                  "Name": "string"
               }
            ],
            "MeasureNameColumn": "string",
            "MixedMeasureMappings": [
               {
                  "MeasureName": "string",
                  "MeasureValueType": "string",
                  "MultiMeasureAttributeMappings": [
                     {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                     }
                  ],
                  "SourceColumn": "string",
                  "TargetMeasureName": "string"
               }
            ],
            "MultiMeasureMappings": {
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "TargetMultiMeasureName": "string"
            },
            "TableName": "string",
            "TimeColumn": "string"
         }
      }
   }
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

ScheduledQuery

计划查询。

类型：ScheduledQueryDescription 对象

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

找不到请求的资源。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

ExecuteScheduledQuery
服务: Amazon Timestream Query

您可以使用它API来手动运行计划查询。

如果您启用QueryInsights，API它还会返回与您在 Amazon SNS 通知中执行的查询相关的见解和指标。 QueryInsights有助于调整查询的性能。有关更多信息QueryInsights，请参阅使用查询见解来优化 Amazon Timestream 中的查询。

请求语法

```
{
   "ClientToken": "string",
   "InvocationTime": number,
   "QueryInsights": {
      "Mode": "string"
   },
   "ScheduledQueryArn": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ClientToken

未使用。

类型：字符串

长度限制：最小长度为 32。长度上限为 128。

必需：否

InvocationTime

中的时间戳。UTC查询将像在此时间戳被调用一样运行。

类型：时间戳

必需：是

QueryInsights

封装用于启用的设置。QueryInsights

在您执行的查询的 Amazon SNS 通知中启用QueryInsights退货见解和指标。您可以使用QueryInsights来调整查询性能和成本。

类型：ScheduledQueryInsights 对象

必需：否

ScheduledQueryArn

ARN的计划查询。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

找不到请求的资源。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

示例

ENABLED_ WITH RATE _ CONTROL 模式的预设查询通知消息

以下示例显示了QueryInsights参数ENABLED_WITH_RATE_CONTROL模式的成功计划查询通知消息。

```
"SuccessNotificationMessage": {
    "type": "MANUAL_TRIGGER_SUCCESS",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-49c6ed55-
c2e7-4cc2-9956-4a0ecea13420-80e05b035236a4c3",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1723710546,
        "triggerTimeMillis": 1723710547490,
        "runStatus": "MANUAL_TRIGGER_SUCCESS",
        "executionStats": {
            "executionTimeInMillis": 17343,
            "dataWrites": 1024,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 600,
            "recordsIngested": 1,
            "queryResultRows": 1
        },
        "queryInsightsResponse": {
            "querySpatialCoverage": {
                "max": {
                    "value": 1.0,
```

```
                    "tableArn": "arn:aws:timestream:<Region>:<Account>:database/BaseDb/
table/BaseTable",
                    "partitionKey": [
                        "measure_name"
                    ]
                }
            },
            "queryTemporalRange": {
                "max": {
                    "value": 2399999999999,
                    "tableArn": "arn:aws:timestream:<Region>:<Account>:database/BaseDb/
table/BaseTable"
                }
            },
            "queryTableCount": 1,
            "outputRows": 1,
            "outputBytes": 59
        }
    }
}
```

该DISABLED模式的预设查询通知消息

以下示例显示了QueryInsights参数DISABLED模式的成功计划查询通知消息。

```
"SuccessNotificationMessage": {
    "type": "MANUAL_TRIGGER_SUCCESS",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
fa109d9e-6528-4a0d-ac40-482fa05e657f-140faaeecdc5b2a7",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1723711401,
        "triggerTimeMillis": 1723711402144,
        "runStatus": "MANUAL_TRIGGER_SUCCESS",
        "executionStats": {
            "executionTimeInMillis": 17992,
            "dataWrites": 1024,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 600,
            "recordsIngested": 1,
            "queryResultRows": 1
        }
    }
}
```

ENABLED_ WITH RATE _ CONTROL 模式的失败通知消息

以下示例显示了QueryInsights参数ENABLED_WITH_RATE_CONTROL模式的计划查询失败通知消息。

```
"FailureNotificationMessage": {
    "type": "MANUAL_TRIGGER_FAILURE",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
b261670d-790c-4116-9db5-0798071b18b1-b7e27a1d79be226d",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1727915513,
        "triggerTimeMillis": 1727915513894,
        "runStatus": "MANUAL_TRIGGER_FAILURE",
        "executionStats": {
            "executionTimeInMillis": 10777,
            "dataWrites": 0,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 0,
            "recordsIngested": 0,
            "queryResultRows": 4
        },
        "errorReportLocation": {
            "s3ReportLocation": {
                "bucketName": "my-amzn-s3-demo-bucket",
                "objectKey": "4my-organization-f7a3c5d065a1a95e/1727915513/
MANUAL/1727915513894/5e14b3df-b147-49f4-9331-784f749b68ae"
            }
        },
        "failureReason": "Schedule encountered some errors and is incomplete. Please
 take a look at error report for further details"
    }
}
```

该DISABLED模式的失败通知消息

以下示例显示了QueryInsights参数DISABLED模式的计划查询失败通知消息。

```
"FailureNotificationMessage": {
    "type": "MANUAL_TRIGGER_FAILURE",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
b261670d-790c-4116-9db5-0798071b18b1-b7e27a1d79be226d",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1727915194,
```

```
        "triggerTimeMillis": 1727915195119,
        "runStatus": "MANUAL_TRIGGER_FAILURE",
        "executionStats": {
            "executionTimeInMillis": 10777,
            "dataWrites": 0,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 0,
            "recordsIngested": 0,
            "queryResultRows": 4
        },
        "errorReportLocation": {
            "s3ReportLocation": {
                "bucketName": "my-amzn-s3-demo-bucket",
                "objectKey": "4my-organization-b7e27a1d79be226d/1727915194/
MANUAL/1727915195119/08dea9f5-9a0a-4e63-a5f7-ded23247bb98"
            }
        },
        "failureReason": "Schedule encountered some errors and is incomplete. Please
 take a look at error report for further details"
    }
}
```

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

ListScheduledQueries
服务: Amazon Timestream Query

获取来电者的 Amazon 账户和地区中所有预定查询的列表。 `ListScheduledQueries`最终是一致的。

请求语法

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

MaxResults

输出中要返回的最大项目数。如果可用项目总数大于指定值，则输出中会提供 a `NextToken`。要恢复分页，请提供该`NextToken`值作为后续调用的参数。`ListScheduledQueriesRequest`

类型：整数

有效范围：最小值为 1。最大值为 1000。

必需：否

NextToken

用于恢复分页的分页令牌。

类型：字符串

必需：否

响应语法

```
{
```

```
   "NextToken": "string",
   "ScheduledQueries": [
      {
         "Arn": "string",
         "CreationTime": number,
         "ErrorReportConfiguration": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "ObjectKeyPrefix": "string"
            }
         },
         "LastRunStatus": "string",
         "Name": "string",
         "NextInvocationTime": number,
         "PreviousInvocationTime": number,
         "State": "string",
         "TargetDestination": {
            "TimestreamDestination": {
               "DatabaseName": "string",
               "TableName": "string"
            }
         }
      }
   ]
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

NextToken

指定从何处开始分页的令牌。这是之前截 NextToken 断的响应。

类型：字符串

ScheduledQueries

计划查询列表。

类型：ScheduledQuery 对象数组

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2

- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

ListTagsForResource
服务: Amazon Timestream Query

列出 Timestream 查询资源上的所有标签。

请求语法

```
{
   "MaxResults": number,
   "NextToken": "string",
   "ResourceARN": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

MaxResults

要返回的最大标签数。

类型：整数

有效范围：最小值为 1。最大值为 200。

必需：否

NextToken

用于恢复分页的分页令牌。

类型：字符串

必需：否

ResourceARN

带有待列出标签的时间流资源。此值是 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

响应语法

```
{
   "NextToken": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

NextToken

分页令牌，用于在随后调用时恢复分页。ListTagsForResourceResponse

类型：字符串

Tags

当前与 Timestream 资源关联的标签。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

找不到请求的资源。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式不正确。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

PrepareQuery
服务: Amazon Timestream Query

一种同步操作，允许您提交带有参数的查询，这些参数将由 Timestream 存储以供日后运行。Timestream 仅支持在`ValidateOnly`设置为的情况下使用此操作。`true`

请求语法

```
{
   "QueryString": "string",
   "ValidateOnly": boolean
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

QueryString

要用作预准备语句的 Timestream 查询字符串。参数名称可以采用查询字符串 @ 字符后跟标识符的形式指定。

类型：字符串

长度限制：长度下限为 1。长度上限为 262144。

必需：是

ValidateOnly

通过将此值设置为`true`，Timestream 将仅验证查询字符串是否为有效的 Timestream 查询，而不会存储准备好的查询以供日后使用。

类型：布尔值

必需：否

响应语法

```
{
   "Columns": [
      {
```

```
         "Aliased": boolean,
         "DatabaseName": "string",
         "Name": "string",
         "TableName": "string",
         "Type": {
            "ArrayColumnInfo": {
               "Name": "string",
               "Type": "Type"
            },
            "RowColumnInfo": [
               {
                  "Name": "string",
                  "Type": "Type"
               }
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": {
               "Name": "string",
               "Type": "Type"
            }
         }
      }
   ],
   "Parameters": [
      {
         "Name": "string",
         "Type": {
            "ArrayColumnInfo": {
               "Name": "string",
               "Type": "Type"
            },
            "RowColumnInfo": [
               {
                  "Name": "string",
                  "Type": "Type"
               }
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": {
               "Name": "string",
               "Type": "Type"
            }
         }
      }
```

```
    ],
    "QueryString": "string"
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

Columns

已提交的查询字符串的SELECT子句列列表。

类型：SelectColumn 对象数组

Parameters

提交的查询字符串中使用的参数列表。

类型：ParameterMapping 对象数组

QueryString

要准备的查询字符串。

类型：字符串

长度限制：长度下限为 1。长度上限为 262144。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式不正确。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

Query

服务: Amazon Timestream Query

`Query`是一种同步操作，可让您对自己的 Amazon Timestream 数据运行查询。

如果您启用`QueryInsights`，API它还会返回与您执行的查询相关的见解和指标。`QueryInsights`有助于调整查询的性能。有关更多信息`QueryInsights`，请参阅使用查询见解来优化 Amazon Timestream 中的查询。

> **Note**
>
> `QueryInsights`启用后，允许您`Query`API发出的最大请求数为每秒 1 个查询 (QPS)。如果超过此查询速率，则可能会导致限制。

`Query`将在 60 秒后超时。您必须更新中的默认超时时间SDK才能支持 60 秒的超时。有关详细信息，请参阅代码示例。

在以下情况下，您的查询请求将失败：

- 如果您在 5 分钟等性窗口之外使用相同的客户端令牌提交`Query`请求。
- 如果您使用相同的客户端令牌提交`Query`请求，但更改了其他参数，则在 5 分钟的等性窗口内。
- 如果行的大小（包括查询元数据）超过 1 MB，则查询将失败并显示以下错误消息：

  ```
  Query aborted as max page response size has been exceeded by the output result row
  ```

- 如果查询发起者和结果读取器的IAM主体不一样和/或查询发起者和结果读取器在查询请求中没有相同的查询字符串，则查询将失败并出现错误。`Invalid pagination token`

请求语法

```
{
   "ClientToken": "string",
   "MaxRows": number,
   "NextToken": "string",
   "QueryInsights": {
      "Mode": "string"
   },
   "QueryString": "string"
```

```
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ClientToken

Query发出请求时指定的、区分大小写的唯一ASCII字符串，最多 64 个字符。提供 a ClientToken 会使调用Query等性。这意味着重复运行相同的查询将产生相同的结果。换句话说，发出多个相同的Query请求与发出单个请求的效果相同。在查询ClientToken中使用时，请注意以下几点：

- 如果在不带的情况下实例化查询ClientToken，则查询将ClientToken代表您SDK生成。API
- 如果Query调用仅包含ClientToken但不包含NextToken，则假定该调用Query是新运行的查询。
- 如果调用包含NextToken，则假定该特定调用是对先前一次查询调用的后续调用API，并返回结果集。
- 4 小时后，任何具有相同请求的请求ClientToken都将被视为新请求。

类型：字符串

长度限制：最小长度为 32。长度上限为 128。

必需：否

MaxRows

要在Query输出中返回的总行数。在两种情况下，Query使用指定MaxRows值的初始运行将返回查询的结果集：

- 结果的大小小于1MB。
- 结果集中的行数小于的值maxRows。

否则，的初始调用Query只会返回 aNextToken，然后可以在后续调用中使用它来获取结果集。要恢复分页，请在后续命令中提供该NextToken值。

如果行大小很大（例如，一行有许多列），Timestream 可能会返回更少的行，以防止响应大小超过 1 MB 的限制。如果MaxRows未提供，Timestream 将发送必要的行数以满足 1 MB 的限制。

类型：整数

有效范围：最小值为 1。最大值为 1000。

必需：否

NextToken

用于返回一组结果的分页标记。使用调用时NextToken，假定该特定调用是对先前调用的后续调用Query，并返回结果集。Query API但是，如果Query调用仅包含ClientToken，则假定该调用Query是新运行的查询。

在查询 NextToken 中使用时，请注意以下几点：

- 分页令牌最多可用于五Query次调用，或者持续时间长达 1 小时（以先到者为准）。
- 使用相同NextToken值将返回相同的记录集。要继续对结果集进行分页，必须使用最新nextToken的。
- 假设Query调用返回两个NextToken值，TokenA和。TokenB如果TokenB在后续Query调用中使用，TokenA则失效且无法重复使用。
- 要在分页开始后从查询中请求先前的结果集，必须重新调用该查询。API
- 在返回之前null，NextToken应使用最新的分页，此时NextToken应使用新的分页。
- 如果查询发起者和结果读取器的IAM主体不一样和/或查询发起者和结果读取器在查询请求中没有相同的查询字符串，则查询将失败并出现错误。Invalid pagination token

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

QueryInsights

封装用于启用的设置。QueryInsights

启用后，除了您执行的查询的查询结果外，还会QueryInsights返回见解和指标。您可以使用QueryInsights来调整查询性能。

类型：QueryInsights 对象

必需：否

QueryString

将由 Timestream 运行的查询。

类型：字符串

长度限制：长度下限为 1。长度上限为 262144。

必需：是

响应语法

```
{
   "ColumnInfo": [
      {
         "Name": "string",
         "Type": {
            "ArrayColumnInfo": "ColumnInfo",
            "RowColumnInfo": [
               "ColumnInfo"
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": "ColumnInfo"
         }
      }
   ],
   "NextToken": "string",
   "QueryId": "string",
   "QueryInsightsResponse": {
      "OutputBytes": number,
      "OutputRows": number,
      "QuerySpatialCoverage": {
         "Max": {
            "PartitionKey": [ "string" ],
            "TableArn": "string",
            "Value": number
         }
      },
      "QueryTableCount": number,
      "QueryTemporalRange": {
         "Max": {
            "TableArn": "string",
            "Value": number
         }
      },
      "UnloadPartitionCount": number,
      "UnloadWrittenBytes": number,
```

```
      "UnloadWrittenRows": number
   },
   "QueryStatus": {
      "CumulativeBytesMetered": number,
      "CumulativeBytesScanned": number,
      "ProgressPercentage": number
   },
   "Rows": [
      {
         "Data": [
            {
               "ArrayValue": [
                  "Datum"
               ],
               "NullValue": boolean,
               "RowValue": "Row",
               "ScalarValue": "string",
               "TimeSeriesValue": [
                  {
                     "Time": "string",
                     "Value": "Datum"
                  }
               ]
            }
         ]
      }
   ]
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

ColumnInfo

返回的结果集的列数据类型。

类型：ColumnInfo 对象数组

NextToken

一种分页令牌，可以在Query调用时再次使用以获取下一组结果。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

QueryId

给定查询的唯一 ID。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：`[a-zA-Z0-9]+`

QueryInsightsResponse

封装`QueryInsights`包含与您执行的查询相关的见解和指标。

类型：QueryInsightsResponse 对象

QueryStatus

有关查询状态的信息，包括进度和扫描的字节。

类型：QueryStatus 对象

Rows

查询返回的结果集行。

类型：Row 对象数组

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

ConflictException

无法对已取消的查询进行轮询结果。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

QueryExecutionException

Timestream 无法成功运行查询。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式不正确。

HTTP状态码：400

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python

- AWS SDK适用于 Ruby V3

TagResource
服务: Amazon Timestream Query

将一组标签与时间流资源相关联。然后，您可以激活这些用户定义的标签，使它们显示在账单和成本管理控制台上，用于跟踪成本分配。

请求语法

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ResourceARN

标识应向其添加标签的时间流资源。此值是 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

Tags

要分配给 Timestream 资源的标签。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

找不到请求的资源。

HTTP状态码：400

ServiceQuotaExceededException

您已超过服务配额。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++

- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

UntagResource
服务: Amazon Timestream Query

从 Timestream 查询资源中移除标签的关联。

请求语法

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ResourceARN

将从中移除标签的时间流资源。此值是 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

TagKeys

标签密钥列表。密钥属于此列表成员的资源的现有标签将从 Timestream 资源中删除。

类型：字符串数组

数组成员：最少 0 个物品。最多 200 项。

长度限制：长度下限为 1。长度上限为 128。

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

找不到请求的资源。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

UpdateAccountSettings
服务: Amazon Timestream Query

转换您的账户以TCUs用于查询定价，并修改您配置的最大查询计算单位。如果将的值减少MaxQueryTCU到所需的配置，则新值最多可能需要 24 小时才能生效。

> **Note**
>
> 在将账户过渡到TCUs用于查询定价之后，您就无法过渡到使用扫描字节来进行查询定价。

请求语法

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

MaxQueryTCU

该服务在任何时间点为你的查询提供服务所使用的最大计算单元数。要运行查询，必须将最小容量设置为 4 TCU。您可以以 4 TCU 的倍数设置最大数量，例如 4、8、16、32 等。

支持的最大值MaxQueryTCU为 1000。要申请提高此软限制，请联系 Su AWS pport。有关默认配额的信息 maxQueryTCU，请参阅默认配额。

类型：整数

必需：否

QueryPricingModel

账户中查询的定价模型。

> **Note**
>
> 该QueryPricingModel参数由多个 Timestream 操作使用；但是，该UpdateAccountSettingsAPI操作无法识别除COMPUTE_UNITS之外的任何值。

类型：字符串

有效值：BYTES_SCANNED | COMPUTE_UNITS

必需：否

响应语法

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

响应元素

如果操作成功，服务将发回 HTTP 200 响应。

以下数据由服务以JSON格式返回。

MaxQueryTCU

该服务在任何时间点为你的查询提供服务的已配置的最大计算单元数。

类型：整数

QueryPricingModel

账户的定价模式。

类型：字符串

有效值：BYTES_SCANNED | COMPUTE_UNITS

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式错误。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2
- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python

- AWS SDK适用于 Ruby V3

UpdateScheduledQuery
服务: Amazon Timestream Query

更新计划查询。

请求语法

```
{
   "ScheduledQueryArn": "string",
   "State": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅通用参数。

该请求接受以下JSON格式的数据。

ScheduledQueryArn

ARN计划查询的。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

State

计划查询的状态。

类型：字符串

有效值：ENABLED | DISABLED

必需：是

响应元素

如果操作成功，服务将发回 HTTP 200 响应，HTTP正文为空。

错误

有关所有操作的常见错误的信息，请参阅常见错误。

AccessDeniedException

您无权执行此操作。

HTTP状态码：400

InternalServerException

由于内部服务器错误，该服务无法完全处理此请求。

HTTP状态码：400

InvalidEndpointException

请求的端点无效。

HTTP状态码：400

ResourceNotFoundException

找不到请求的资源。

HTTP状态码：400

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationException

请求无效或格式不正确。

HTTP状态码：400

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS SDK对于。 NET
- AWS SDK对于 C++
- AWS SDK适用于 Go v2

- AWS SDK适用于 Java V2
- AWS SDK适用于 JavaScript V3
- AWS SDK适用于 PHP V3
- AWS SDK适用于 Python
- AWS SDK适用于 Ruby V3

数据类型

Amazon Timestream Write 支持以下数据类型：

- BatchLoadProgressReport
- BatchLoadTask
- BatchLoadTaskDescription
- CsvConfiguration
- Database
- DataModel
- DataModelConfiguration
- DataModelS3Configuration
- DataSourceConfiguration
- DataSourceS3Configuration
- Dimension
- DimensionMapping
- Endpoint
- MagneticStoreRejectedDataLocation
- MagneticStoreWriteProperties
- MeasureValue
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- PartitionKey
- Record
- RecordsIngested

- RejectedRecord
- ReportConfiguration
- ReportS3Configuration
- RetentionProperties
- S3Configuration
- Schema
- Table
- Tag

Amazon Timestream 查询支持以下数据类型：

- ColumnInfo
- Datum
- DimensionMapping
- Endpoint
- ErrorReportConfiguration
- ErrorReportLocation
- ExecutionStats
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- NotificationConfiguration
- ParameterMapping
- QueryInsights
- QueryInsightsResponse
- QuerySpatialCoverage
- QuerySpatialCoverageMax
- QueryStatus
- QueryTemporalRange
- QueryTemporalRangeMax
- Row

- S3Configuration
- S3ReportLocation
- ScheduleConfiguration
- ScheduledQuery
- ScheduledQueryDescription
- ScheduledQueryInsights
- ScheduledQueryInsightsResponse
- ScheduledQueryRunSummary
- SelectColumn
- SnsConfiguration
- Tag
- TargetConfiguration
- TargetDestination
- TimeSeriesDataPoint
- TimestreamConfiguration
- TimestreamDestination
- Type

亚马逊 Timestream Write

Amazon Timestream Write 支持以下数据类型：

- BatchLoadProgressReport
- BatchLoadTask
- BatchLoadTaskDescription
- CsvConfiguration
- Database
- DataModel
- DataModelConfiguration
- DataModelS3Configuration
- DataSourceConfiguration
- DataSourceS3Configuration

- Dimension
- DimensionMapping
- Endpoint
- MagneticStoreRejectedDataLocation
- MagneticStoreWriteProperties
- MeasureValue
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- PartitionKey
- Record
- RecordsIngested
- RejectedRecord
- ReportConfiguration
- ReportS3Configuration
- RetentionProperties
- S3Configuration
- Schema
- Table
- Tag

BatchLoadProgressReport
服务: Amazon Timestream Write

有关批量加载任务进度的详细信息。

内容

BytesMetered

类型：长整型

必需：否

FileFailures

类型：长整型

必需：否

ParseFailures

类型：长整型

必需：否

RecordIngestionFailures

类型：长整型

必需：否

RecordsIngested

类型：长整型

必需：否

RecordsProcessed

类型：长整型

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

BatchLoadTask
服务: Amazon Timestream Write

有关批量加载任务的详细信息。

内容

CreationTime

Timestream 批量加载任务的创建时间。

类型：时间戳

必需：否

DatabaseName

批量加载任务将数据加载到的数据库的数据库名称。

类型：字符串

必需：否

LastUpdatedTime

Timestream 批量加载任务的上次更新时间。

类型：时间戳

必需：否

ResumableUntil

类型：时间戳

必需：否

TableName

批量加载任务将数据加载到的表的表名。

类型：字符串

必需：否

TaskId

批量加载任务的 ID。

类型：字符串

长度约束：最小长度为 3。最大长度为 32。

模式：[A-Z0-9]+

必需：否

TaskStatus

批量加载任务的状态。

类型：字符串

有效值：CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

BatchLoadTaskDescription
服务: Amazon Timestream Write

有关批量加载任务的详细信息。

内容

CreationTime

Timestream 批量加载任务的创建时间。

类型：时间戳

必需：否

DataModelConfiguration

批量加载任务的数据模型配置。其中包含有关批量加载任务的数据模型存储位置的详细信息。

类型：DataModelConfiguration 对象

必需：否

DataSourceConfiguration

有关批量加载任务的数据源的配置详细信息。

类型：DataSourceConfiguration 对象

必需：否

ErrorMessage

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

LastUpdatedTime

Timestream 批量加载任务的上次更新时间。

类型：时间戳

必需：否

ProgressReport

类型：BatchLoadProgressReport 对象

必需：否

RecordVersion

类型：长整型

必需：否

ReportConfiguration

报告批量加载任务的配置。其中包含有关错误报告的存储位置的详细信息。

类型：ReportConfiguration 对象

必需：否

ResumableUntil

类型：时间戳

必需：否

TargetDatabaseName

类型：字符串

必需：否

TargetTableName

类型：字符串

必需：否

TaskId

批量加载任务的 ID。

类型：字符串

长度约束：最小长度为 3。最大长度为 32。

模式：[A-Z0-9]+

必需：否

TaskStatus

批量加载任务的状态。

类型：字符串

有效值：CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

CsvConfiguration
服务: Amazon Timestream Write

一种分隔的数据格式，其中列分隔符可以是逗号，记录分隔符可以是换行符。

内容

ColumnSeparator

列分隔符可以是逗号 (','）、竖线 ('|)、分号 (';')、制表符 ('/t') 或空格 (") 之一。

类型：字符串

长度限制：固定长度为 1。

必需：否

EscapeChar

转义字符可以是其中之一

类型：字符串

长度限制：固定长度为 1。

必需：否

NullValue

可以是空格 (")。

类型：字符串

长度限制：最小长度为 1。最大长度为 256。

必需：否

QuoteChar

可以是单引号 (') 或双引号 (“)。

类型：字符串

长度限制：固定长度为 1。

必需：否

TrimWhiteSpace

指定修剪前导和尾随的空白。

类型：布尔值

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Database
服务: Amazon Timestream Write

表格的顶级容器。数据库和表是 Amazon Timestream 中的基本管理概念。数据库中的所有表都使用相同的 AWS KMS 密钥进行加密。

内容

Arn

唯一标识此数据库的 Amazon 资源名称。

类型：字符串

必需：否

CreationTime

数据库的创建时间，从 Unix 纪元时间开始计算。

类型：时间戳

必需：否

DatabaseName

Timestream 数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：否

KmsKeyId

用于加密存储在数据库中的数据的密 AWS KMS 钥的标识符。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

LastUpdatedTime

上次更新此数据库的时间。

类型：时间戳

必需：否

TableCount

在 Timestream 数据库中找到的表的总数。

类型：长整型

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

DataModel
服务: Amazon Timestream Write

批量加载任务的数据模型。

内容

DimensionMappings

维度的源到目标映射。

类型：DimensionMapping 对象数组

数组成员：最少 1 个物品。

必需：是

MeasureNameColumn

类型：字符串

长度限制：最小长度为 1。最大长度为 256。

必需：否

MixedMeasureMappings

测量的源到目标映射。

类型：MixedMeasureMapping 对象数组

数组成员：最少 1 个物品。

必需：否

MultiMeasureMappings

多度量记录的源到目标映射。

类型：MultiMeasureMappings 对象

必需：否

TimeColumn

要映射到时间的源列。

类型：字符串

长度限制：最小长度为 1。最大长度为 256。

必需：否

TimeUnit

时间戳单位的粒度。它指示时间值是以秒、毫秒、纳秒还是其他支持的值为单位。默认值为 `MILLISECONDS`。

类型：字符串

有效值：`MILLISECONDS | SECONDS | MICROSECONDS | NANOSECONDS`

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

DataModelConfiguration
服务: Amazon Timestream Write

内容

DataModel

类型：DataModel 对象

必需：否

DataModelS3Configuration

类型：DataModelS3Configuration 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

DataModelS3Configuration
服务: Amazon Timestream Write

内容

BucketName

类型：字符串

长度约束：最小长度为 3。最大长度为 63。

模式：[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]

必需：否

ObjectKey

类型：字符串

长度限制：长度下限为 1。长度上限为 1024。

模式：[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

DataSourceConfiguration
服务: Amazon Timestream Write

定义有关数据源的配置详细信息。

内容

DataFormat

目前是CSV。

类型：字符串

有效值：CSV

必需：是

DataSourceS3Configuration

为包含要加载的数据的文件配置 S3 位置。

类型：DataSourceS3Configuration 对象

必需：是

CsvConfiguration

一种分隔的数据格式，其中列分隔符可以是逗号，记录分隔符可以是换行符。

类型：CsvConfiguration 对象

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

DataSourceS3Configuration
服务: Amazon Timestream Write

内容

BucketName

客户 S3 存储桶的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 63。

模式：`[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必需：是

ObjectKeyPrefix

类型：字符串

长度限制：长度下限为 1。长度上限为 1024。

模式：`[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Dimension
服务: Amazon Timestream Write

表示时间序列的元数据属性。例如，EC2实例的名称和可用区或风力涡轮机制造商的名称都是尺寸。

内容

Name

维度表示时间序列的元数据属性。例如，EC2实例的名称和可用区或风力涡轮机制造商的名称都是尺寸。

有关维度名称的限制，请参阅命名约束。

类型：字符串

长度限制：长度下限为 1。最大长度为 60。

必需：是

Value

维度的值。

类型：字符串

必需：是

DimensionValueType

时间序列数据点的维度数据类型。

类型：字符串

有效值：VARCHAR

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2

- AWS SDK适用于 Ruby V3

DimensionMapping
服务: Amazon Timestream Write

内容

DestinationColumn

类型：字符串

长度限制：长度下限为 1。

必需：否

SourceColumn

类型：字符串

长度限制：长度下限为 1。

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Endpoint
服务: Amazon Timestream Write

表示用于对其进行API调用的可用端点，以及该端点TTL的。

内容

Address

终端节点地址。

类型：字符串

必需：是

CachePeriodInMinutes

终端节点TTL的，以分钟为单位。

类型：长整型

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

MagneticStoreRejectedDataLocation
服务: Amazon Timestream Write

在磁存储写入期间，为异步拒绝的记录写入错误报告的位置。

内容

S3Configuration

配置 S3 位置在磁存储写入期间，为异步拒绝的记录写入错误报告。

类型：S3Configuration 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

MagneticStoreWriteProperties
服务: Amazon Timestream Write

表上用于配置磁存储写入的一组属性。

内容

EnableMagneticStoreWrites

启用磁性存储写入的标志。

类型：布尔值

必需：是

MagneticStoreRejectedDataLocation

在磁存储写入期间，为异步拒绝的记录写入错误报告的位置。

类型：MagneticStoreRejectedDataLocation 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

MeasureValue
服务: Amazon Timestream Write

表示时间序列的数据属性。例如，EC2实例或风力涡轮机的CPU利用率就是衡量标准。RPM MeasureValue 既有名称又有值。

MeasureValue 仅允许用于 type MULTI。使用 t MULTI ype，可以在单个记录中传递与相同时间序列关联的多个数据属性

内容

Name

的名字 MeasureValue。

有关 MeasureValue 名称的限制，请参阅《亚马逊 Timestream 开发者指南》中的命名限制。

类型：字符串

长度限制：长度下限为 1。

必需：是

Type

包含时间序列数据点的数据类型。 MeasureValue

类型：字符串

有效值：DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI

必需：是

Value

的值 MeasureValue。有关信息，请参阅数据类型。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

MixedMeasureMapping
服务: Amazon Timestream Write

内容

MeasureValueType

类型：字符串

有效值：DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI

必需：是

MeasureName

类型：字符串

长度限制：长度下限为 1。

必需：否

MultiMeasureAttributeMappings

类型：MultiMeasureAttributeMapping 对象数组

数组成员：最少 1 个物品。

必需：否

SourceColumn

类型：字符串

长度限制：长度下限为 1。

必需：否

TargetMeasureName

类型：字符串

长度限制：长度下限为 1。

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

MultiMeasureAttributeMapping
服务: Amazon Timestream Write

内容

SourceColumn

类型：字符串

长度限制：长度下限为 1。

必需：是

MeasureValueType

类型：字符串

有效值：DOUBLE | BIGINT | BOOLEAN | VARCHAR | TIMESTAMP

必需：否

TargetMultiMeasureAttributeName

类型：字符串

长度限制：长度下限为 1。

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

MultiMeasureMappings
服务: Amazon Timestream Write

内容

MultiMeasureAttributeMappings

类型：MultiMeasureAttributeMapping 对象数组

数组成员：最少 1 个物品。

必需：是

TargetMultiMeasureName

类型：字符串

长度限制：长度下限为 1。

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

PartitionKey
服务: Amazon Timestream Write

用于对表中的数据进行分区的属性。维度键使用由维度名称指定的维度值作为分区键对数据进行分区，而度量键则使用度量名称（“measure_name” 列的值）对数据进行分区。

内容

Type

分区键的类型。选项有DIMENSION（维度键）和MEASURE（度量键）。

类型：字符串

有效值：`DIMENSION | MEASURE`

必需：是

EnforcementInRecord

在摄取的记录中指定维度键的强制执行级别。选项包括REQUIRED（必须指定维度键）和OPTIONAL（不必指定维度键）。

类型：字符串

有效值：`REQUIRED | OPTIONAL`

必需：否

Name

用于维度键的属性的名称。

类型：字符串

长度限制：长度下限为 1。

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++

- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Record
服务: Amazon Timestream Write

表示正在写入 Timestream 的时间序列数据点。每条记录都包含一个维度数组。维度表示时间序列数据点的元数据属性，例如实例名称或EC2实例的可用区。记录还包含度量名称，即正在收集的度量的名称（例如，EC2实例的CPU利用率）。此外，记录还包含度量值和值类型，即测量值的数据类型。此外，该记录还包含收集度量的时间戳和时间戳单位，后者表示时间戳的粒度。

记录中有一个 64 位`Version`字段`long`，可用于更新数据点。只有在写入请求中记录的属性高于现有记录的`Version`属性时，才会成功写入具有相同维度、时间戳和度量名称但度量值不同的重复记录。1对于没有该`Version`字段`Version`的记录，时间流默认为 a of。

内容

Dimensions

包含时间序列数据点的维度列表。

类型：Dimension 对象数组

数组成员：最大数量为 128 项。

必需：否

MeasureName

度量表示时间序列的数据属性。例如，EC2实例或风力涡轮机的CPU利用率就是衡量标准。RPM

类型：字符串

长度限制：最小长度为 1。最大长度为 256。

必需：否

MeasureValue

包含时间序列数据点的度量值。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

MeasureValues

包含时间序列数据点 MeasureValue 的列表。

这仅允许用于类型MULTI。对于标量值，请直接使用记录的MeasureValue属性。

类型：MeasureValue 对象数组

必需：否

MeasureValueType

包含时间序列数据点的度量值的数据类型。默认类型为DOUBLE。有关更多信息，请参阅数据类型。

类型：字符串

有效值：DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI

必需：否

Time

包含收集数据点测量值的时间。时间值加上单位提供了自纪元以来经过的时间。例如，如果时间值为，单位为 12345ms，则自纪元以来12345 ms已经过去了。

类型：字符串

长度限制：最小长度为 1。最大长度为 256。

必需：否

TimeUnit

时间戳单位的粒度。它指示时间值是以秒、毫秒、纳秒还是其他支持的值为单位。默认值为MILLISECONDS。

类型：字符串

有效值：MILLISECONDS | SECONDS | MICROSECONDS | NANOSECONDS

必需：否

Version

用于更新记录的 64 位属性。对具有更高版本号的重复数据的写入请求将更新现有的度量值和版本。如果测量值相同，则仍Version会更新。默认值为 1。

> **Note**
>
> `Version`必须等于1或更大，否则您将收到`ValidationException`错误消息。

类型：长整型

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

RecordsIngested
服务: Amazon Timestream Write

有关此请求获取的记录的信息。

内容

MagneticStore

磁性存储器中摄入的记录数。

类型：整数

必需：否

MemoryStore

已摄入内存存储的记录数。

类型：整数

必需：否

Total

成功摄取的记录总数。

类型：整数

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

RejectedRecord
服务: Amazon Timestream Write

表示由于数据验证问题而未能成功插入到 Timestream 中的记录，在将时间序列数据重新插入系统之前必须解决这些问题。

内容

ExistingVersion

记录的现有版本。如果存在相同记录且版本高于写入请求中的版本，则会填充此值。

类型：长整型

必需：否

Reason

记录未成功插入到 Timestream 的原因。可能的失败原因包括：

- 具有重复数据的记录，其中有多个具有相同维度、时间戳和度量名称的记录，但是：
  - 测量值不同
  - 请求中不存在版本，或者新记录中版本的值等于或小于现有值

  如果 Timestream 拒绝这种情况的数据，则`RejectedRecords`响应中的`ExistingVersion`字段将指示当前记录的版本。要强制更新，您可以重新发送请求，并将该记录的版本设置为大于。`ExistingVersion`
- 时间戳超出内存存储保留期限的记录。

  > **Note**
  >
  > 更新保留期限后，如果您立即尝试在新窗口中提取数据，则会收到`RejectedRecords`异常消息。为避免出现`RejectedRecords`异常，请等到新窗口持续时间后才能摄取新数据。有关更多信息，请参阅配置时间流的最佳实践和对存储在 Timestream 中的工作原理的说明。

- 维度或度量超过时间流定义限制的记录。

有关更多信息，请参阅《Timestream 开发人员指南》中的访问管理。

类型：字符串

必需：否

RecordIndex

的输入请求中记录的索引 WriteRecords。索引以 0 开头。

类型：整数

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ReportConfiguration
服务: Amazon Timestream Write

报告批量加载任务的配置。其中包含有关错误报告的存储位置的详细信息。

内容

ReportS3Configuration

配置 S3 位置以写入批量加载的错误报告和事件。

类型：ReportS3Configuration 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ReportS3Configuration
服务: Amazon Timestream Write

内容

BucketName

类型：字符串

长度约束：最小长度为 3。最大长度为 63。

模式：[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]

必需：是

EncryptionOption

类型：字符串

有效值：SSE_S3 | SSE_KMS

必需：否

KmsKeyId

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

ObjectKeyPrefix

类型：字符串

长度限制：长度下限为 1。最大长度为 928。

模式：[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

RetentionProperties
服务: Amazon Timestream Write

保留属性包含您的时间序列数据必须存储在磁存储和内存存储中的持续时间。

内容

MagneticStoreRetentionPeriodInDays

数据必须存储在磁存储中的持续时间。

类型：长整型

有效范围：最小值为 1。最大值为 73000。

必需：是

MemoryStoreRetentionPeriodInHours

数据必须存储在内存存储中的持续时间。

类型：长整型

有效范围：最小值为 1。最大值为 8766。

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

S3Configuration
服务: Amazon Timestream Write

指定 S3 位置的配置。

内容

BucketName

客户 S3 存储桶的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 63。

模式：`[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必需：否

EncryptionOption

客户 S3 位置的加密选项。选项包括使用 S3 托管密钥或 AWS 托管密钥进行 S3 服务器端加密。

类型：字符串

有效值：`SSE_S3 | SSE_KMS`

必需：否

KmsKeyId

使用 AWS 托管 AWS KMS 密钥加密时客户 S3 位置的密钥 ID。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

ObjectKeyPrefix

客户 S3 位置的对象键预览。

类型：字符串

长度限制：长度下限为 1。最大长度为 928。

模式：`[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Schema

服务: Amazon Timestream Write

架构指定表的预期数据模型。

内容

CompositePartitionKey

一个非空的分区键列表，用于定义用于对表数据进行分区的属性。列表的顺序决定了分区层次结构。创建表后，不能更改每个分区键的名称和类型以及分区键顺序。但是，可以更改每个分区键的强制级别。

类型：PartitionKey 对象数组

数组成员：最少 1 个物品。

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Table
服务: Amazon Timestream Write

表示 Timestream 中的数据库表。表包含一个或多个相关的时间序列。您可以修改表的内存存储和磁存储的保留时间。

内容

Arn

唯一标识此表的 Amazon 资源名称。

类型：字符串

必需：否

CreationTime

时间流表的创建时间。

类型：时间戳

必需：否

DatabaseName

包含此表的 Timestream 数据库的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：否

LastUpdatedTime

上次更新时间流表的时间。

类型：时间戳

必需：否

MagneticStoreWriteProperties

包含启用磁存储写入时要在表上设置的属性。

类型：MagneticStoreWriteProperties 对象

必需：否

RetentionProperties

内存存储和磁性存储的保留持续时间。

类型：RetentionProperties 对象

必需：否

Schema

表的架构。

类型：Schema 对象

必需：否

TableName

Timestream 表的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 256。

必需：否

TableStatus

表格的当前状态：

- DELETING-正在删除表。
- ACTIVE-桌子已准备就绪，可以使用。

类型：字符串

有效值：ACTIVE | DELETING | RESTORING

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Tag
服务: Amazon Timestream Write

标签是您分配给 Timestream 数据库and/or table. Each tag consists of a key and an optional value, both of which you define. With tags, you can categorize databases and/or表的标签，例如按用途、所有者或环境进行分配。

内容

Key

标签的密钥。标签键区分大小写。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

必需：是

Value

标签的值。标签值区分大小写，可以为空。

类型：字符串

长度约束：最小长度为 0。最大长度为 256。

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

亚马逊 Timestream 查询

Amazon Timestream 查询支持以下数据类型：

- ColumnInfo

- Datum
- DimensionMapping
- Endpoint
- ErrorReportConfiguration
- ErrorReportLocation
- ExecutionStats
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- NotificationConfiguration
- ParameterMapping
- QueryInsights
- QueryInsightsResponse
- QuerySpatialCoverage
- QuerySpatialCoverageMax
- QueryStatus
- QueryTemporalRange
- QueryTemporalRangeMax
- Row
- S3Configuration
- S3ReportLocation
- ScheduleConfiguration
- ScheduledQuery
- ScheduledQueryDescription
- ScheduledQueryInsights
- ScheduledQueryInsightsResponse
- ScheduledQueryRunSummary
- SelectColumn
- SnsConfiguration
- Tag

- TargetConfiguration
- TargetDestination
- TimeSeriesDataPoint
- TimestreamConfiguration
- TimestreamDestination
- Type

ColumnInfo
服务: Amazon Timestream Query

包含查询结果的元数据，例如列名、数据类型和其他属性。

内容

Type

结果集列的数据类型。数据类型可以是标量或复数。标量数据类型包括整数、字符串、双精度、布尔值等。复杂数据类型是数组、行等类型。

类型：Type 对象

必需：是

Name

结果集列的名称。结果集的名称适用于除数组之外的所有数据类型的列。

类型：字符串

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Datum

服务: Amazon Timestream Query

Datum 表示查询结果中的单个数据点。

内容

ArrayValue

表示数据点是否为数组。

类型：Datum 对象数组

必需：否

NullValue

指示数据点是否为空。

类型：布尔值

必需：否

RowValue

指示数据点是否为行。

类型：Row 对象

必需：否

ScalarValue

表示数据点是标量值，例如整数、字符串、双精度还是布尔值。

类型：字符串

必需：否

TimeSeriesValue

指示数据点是否为时间序列数据类型。

类型：TimeSeriesDataPoint 对象数组

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

DimensionMapping
服务: Amazon Timestream Query

此类型用于将查询结果中的列映射到目标表中的维度。

内容

DimensionValueType

维度的类型。

类型：字符串

有效值：VARCHAR

必需：是

Name

查询结果中的列名。

类型：字符串

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Endpoint
服务: Amazon Timestream Query

表示用于对其进行API调用的可用端点，以及该端点TTL的。

内容

Address

终端节点地址。

类型：字符串

必需：是

CachePeriodInMinutes

终端节点TTL的，以分钟为单位。

类型：长整型

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ErrorReportConfiguration
服务: Amazon Timestream Query

错误报告所需的配置。

内容

S3Configuration

错误报告的 S3 配置。

类型：S3Configuration 对象

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ErrorReportLocation
服务: Amazon Timestream Query

这包含单个预定查询呼叫的错误报告的位置。

内容

S3ReportLocation

写入错误报告的 S3 位置。

类型：S3ReportLocation 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ExecutionStats
服务: Amazon Timestream Query

单次计划查询运行的统计信息。

内容

BytesMetered

为单次计划查询运行计量的字节数。

类型：长整型

必需：否

CumulativeBytesScanned

为单次计划查询运行扫描的字节数。

类型：长整型

必需：否

DataWrites

按单次计划查询运行中提取的记录计量的数据写入量。

类型：长整型

必需：否

ExecutionTimeInMillis

完成计划查询运行所需的总时间，以毫秒为单位。

类型：长整型

必需：否

QueryResultRows

在提取到目标数据源之前，运行查询的输出中存在的行数。

类型：长整型

必需：否

RecordsIngested

为单次计划查询运行提取的记录数。

类型：长整型

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

MixedMeasureMapping
服务: Amazon Timestream Query

MixedMeasureMappings 是可用于将数据提取到派生表中窄度量和多度量混合的映射。

内容

MeasureValueType

要从中读取的值的类型sourceColumn。如果映射用于MULTI，请使用 MeasureValueType。MULTI。

类型：字符串

有效值：`BIGINT | BOOLEAN | DOUBLE | VARCHAR | MULTI`

必需：是

MeasureName

指结果行中 measure_name 的值。如果已提供，则此字段 MeasureNameColumn 为必填字段。

类型：字符串

必需：否

MultiMeasureAttributeMappings

何时 measureValueType 为必填项MULTI。MULTI值度量的属性映射。

类型：MultiMeasureAttributeMapping 对象数组

数组成员：最少 1 个物品。

必需：否

SourceColumn

此字段指的是要从中读取度量值以实现结果具体化的源列。

类型：字符串

必需：否

TargetMeasureName

要使用的目标度量名称。如果未提供，则默认情况下，目标度量名称将为度量名称（如果提供），否则 sourceColumn 为其他名称。

类型：字符串

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

MultiMeasureAttributeMapping
服务: Amazon Timestream Query

MULTI价值衡量标准的属性映射。

内容

MeasureValueType

要从源列中读取的属性的类型。

类型：字符串

有效值：BIGINT | BOOLEAN | DOUBLE | VARCHAR | TIMESTAMP

必需：是

SourceColumn

要从中读取属性值的源列。

类型：字符串

必需：是

TargetMultiMeasureAttributeName

用于派生表中属性名称的自定义名称。如果未提供，将使用源列名称。

类型：字符串

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

MultiMeasureMappings
服务: Amazon Timestream Query

只能提供 MixedMeasureMappings 或 MultiMeasureMappings 中的一个。 MultiMeasureMappings 可用于在派生表中以多度量形式摄取数据。

内容

MultiMeasureAttributeMappings

必需。用于将查询结果映射到多度量属性提取数据的属性映射。

类型：MultiMeasureAttributeMapping 对象数组

数组成员：最少 1 个物品。

必需：是

TargetMultiMeasureName

派生表中目标多度量名称的名称。如果未提供此输入 measureNameColumn ，则需要此输入。如果 MeasureNameColumn 已提供，则该列中的值将用作多度量名称。

类型：字符串

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

NotificationConfiguration
服务: Amazon Timestream Query

计划查询的通知配置。创建计划查询、更新其状态或删除该查询时，Timestream 将发送通知。

内容

SnsConfiguration

有关SNS配置的详细信息。

类型：SnsConfiguration 对象

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ParameterMapping
服务: Amazon Timestream Query

命名参数的映射。

内容

Name

参数名称。

类型：字符串

必需：是

Type

包含查询结果集中某列的数据类型。数据类型可以是标量或复数。支持的标量数据类型包括整数、布尔值、字符串、双精度、时间戳、日期、时间和间隔。支持的复杂数据类型包括数组、行和时间序列。

类型：Type 对象

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

QueryInsights
服务: Amazon Timestream Query

QueryInsights是一项性能调整功能，可帮助您优化查询、降低成本和提高性能。借QueryInsights助，您可以评估查询的修剪效率，并确定需要改进的领域，以提高查询性能。借QueryInsights助，您还可以分析查询在时间和空间修剪方面的有效性，并找出提高性能的机会。具体而言，您可以评估您的查询如何使用基于时间和基于分区键的索引策略来优化数据检索。要优化查询性能，必须对控制查询执行的时间和空间参数进行微调。

提供的关键指标QueryInsights是QuerySpatialCoverage和QueryTemporalRange。QuerySpatialCoverage表示查询扫描了多少空间轴，值越低效率越高。QueryTemporalRange显示扫描的时间范围，范围越窄性能越高。

的好处 QueryInsights

以下是使用的主要好处QueryInsights：

- 识别效率低下的查询-QueryInsights 提供有关对查询访问的表进行基于时间和基于属性的修剪的信息。此信息可帮助您识别访问效果不佳的表。
- 优化数据模型和分区-您可以使用这些QueryInsights信息来访问和微调您的数据模型和分区策略。
- 调整查询-QueryInsights 突出显示更有效地使用索引的机会。

> **Note**
>
> QueryInsights启用后，允许您QueryAPI发出的最大请求数为每秒 1 个查询 (QPS)。如果超过此查询速率，则可能会导致限制。

内容

Mode

提供以下模式以启用QueryInsights：

- ENABLED_WITH_RATE_CONTROL— QueryInsights 为正在处理的查询启用。此模式还包括速率控制机制，该机制将该QueryInsights功能限制为每秒 1 次查询 (QPS)。
- DISABLED— 禁用。QueryInsights

类型：字符串

有效值：ENABLED_WITH_RATE_CONTROL | DISABLED

必需：是

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

QueryInsightsResponse
服务: Amazon Timestream Query

提供与您执行的查询相关的各种见解和指标。

内容

OutputBytes

表示查询结果集的大小（以字节为单位）。作为查询调整练习的一部分，您可以使用这些数据来验证结果集是否已更改。

类型：长整型

必需：否

OutputRows

表示作为查询结果集一部分返回的总行数。作为查询调整练习的一部分，您可以使用这些数据来验证结果集中的行数是否发生了变化。

类型：长整型

必需：否

QuerySpatialCoverage

提供对查询的空间覆盖范围的见解，包括具有次优化（最大）空间修剪的表。这些信息可以帮助您确定分区策略中需要改进的领域，以增强空间修剪。

类型：QuerySpatialCoverage 对象

必需：否

QueryTableCount

表示查询中表的数量。

类型：长整型

必需：否

QueryTemporalRange

提供对查询时间范围的见解，包括时间范围最大（最大）的表。以下是一些优化基于时间的修剪的潜在选项：

- 添加缺失的时间谓词。
- 移除时间谓词周围的函数。
- 向所有子查询添加时间谓词。

类型：QueryTemporalRange 对象

必需：否

UnloadPartitionCount

表示Unload操作创建的分区。

类型：长整型

必需：否

UnloadWrittenBytes

表示Unload操作写入的大小（以字节为单位）。

类型：长整型

必需：否

UnloadWrittenRows

表示Unload查询写入的行。

类型：长整型

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

QuerySpatialCoverage
服务: Amazon Timestream Query

提供对查询的空间覆盖范围的见解，包括具有次优化（最大）空间修剪的表。这些信息可以帮助您确定分区策略中需要改进的领域，以增强空间修剪

例如，您可以使用这些QuerySpatialCoverage信息执行以下操作：

- 添加 measure_name 或使用客户定义的分区键 () 谓词CDPK。
- 如果您已经完成了上述操作，请移除它们周围的函数或子句，例如LIKE。

内容

Max

提供对已执行查询的空间覆盖范围的见解，以及空间修剪效率最低的表。

- Value— 空间覆盖的最大比率。
- TableArn— 空间修剪效果不佳的表的 Amazon 资源名称 (ARN)。
- PartitionKey— 用于分区的分区键，可以是默认分区键measure_name或。CDPK

类型：QuerySpatialCoverageMax 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

QuerySpatialCoverageMax
服务: Amazon Timestream Query

提供对表格的见解，其中包含您的查询所扫描的最次优空间范围。

内容

PartitionKey

用于分区的分区键，可以是默认分区键，measure_name也可以是客户定义的分区键。

类型：字符串数组

必需：否

TableArn

空间修剪次数最ARN差的表的 Amazon 资源名称 ()。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

Value

空间覆盖的最大比率。

类型：双精度

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

QueryStatus
服务: Amazon Timestream Query

有关查询状态的信息，包括进度和扫描的字节。

内容

CumulativeBytesMetered

查询扫描的数据量（以字节为单位），您需要为此付费。这是累积总和，表示自查询开始以来您要支付的总数据量。该费用仅收取一次，要么在查询完成运行时收取，要么在查询被取消时收取。

类型：长整型

必需：否

CumulativeBytesScanned

查询扫描的数据量（以字节为单位）。这是累积总和，表示自查询开始以来扫描的总字节数。

类型：长整型

必需：否

ProgressPercentage

查询进度，以百分比表示。

类型：双精度

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

QueryTemporalRange
服务: Amazon Timestream Query

提供对查询时间范围的见解，包括时间范围最大（最大）的表。

内容

Max

封装了以下属性，这些属性可以深入了解时间轴上表现最差的表格：

- `Value`— 查询开始和结束之间的最大持续时间（以纳秒为单位）。
- `TableArn`— 以最大时间范围查询的表的 Amazon 资源名称 (ARN)。

类型：QueryTemporalRangeMax 对象

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

QueryTemporalRangeMax
服务: Amazon Timestream Query

通过查询扫描的最次优的时间修剪，提供对表格的见解。

内容

TableArn

使用最大时间范围查询的表的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

Value

查询开始和结束之间的最大持续时间（以纳秒为单位）。

类型：长整型

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Row
服务: Amazon Timestream Query

代表查询结果中的一行。

内容

Data

结果集单行中的数据点列表。

类型：Datum 对象数组

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

S3Configuration

服务: Amazon Timestream Query

有关运行查询导致的错误报告的 S3 位置的详细信息。

内容

BucketName

将在其下创建错误报告的 S3 桶的名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 63。

模式：`[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必需：是

EncryptionOption

错误报告的静态加密选项。如果未指定加密选项，Timestream 将选择 SSE _S3 作为默认值。

类型：字符串

有效值：`SSE_S3 | SSE_KMS`

必需：否

ObjectKeyPrefix

错误报告密钥的前缀。默认情况下，Timestream 将在错误报告路径中添加以下前缀。

类型：字符串

长度限制：长度下限为 1。最大长度为 896。

模式：`[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

S3ReportLocation
服务: Amazon Timestream Query

计划查询运行的 S3 报告位置。

内容

BucketName

S3 桶名称。

类型：字符串

长度约束：最小长度为 3。最大长度为 63。

模式：`[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必需：否

ObjectKey

S3 密钥。

类型：字符串

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ScheduleConfiguration
服务: Amazon Timestream Query

查询计划的配置。

内容

ScheduleExpression

表示何时触发计划查询运行的表达式。此项可以是 cron 表达式或 rate 表达式。

类型：字符串

长度限制：最小长度为 1。最大长度为 256。

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ScheduledQuery
服务: Amazon Timestream Query

计划查询

内容

Arn

亚马逊资源名称。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

Name

计划查询的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+

必需：是

State

计划查询的状态。

类型：字符串

有效值：ENABLED | DISABLED

必需：是

CreationTime

定时查询的创建时间。

类型：时间戳

必需：否

ErrorReportConfiguration

计划查询错误报告的配置。

类型：ErrorReportConfiguration 对象

必需：否

LastRunStatus

上次计划查询运行的状态。

类型：字符串

有效值：AUTO_TRIGGER_SUCCESS | AUTO_TRIGGER_FAILURE | MANUAL_TRIGGER_SUCCESS | MANUAL_TRIGGER_FAILURE

必需：否

NextInvocationTime

下次运行定时查询时。

类型：时间戳

必需：否

PreviousInvocationTime

上次运行定时查询的时间。

类型：时间戳

必需：否

TargetDestination

将写入最终计划查询结果的目标数据源。

类型：TargetDestination 对象

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ScheduledQueryDescription
服务: Amazon Timestream Query

描述计划查询的结构。

内容

Arn

定时查询ARN。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

Name

计划查询的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：`[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必需：是

NotificationConfiguration

通知配置。

类型：NotificationConfiguration 对象

必需：是

QueryString

要运行的查询。

类型：字符串

长度限制：长度下限为 1。长度上限为 262144。

必需：是

ScheduleConfiguration

计划配置。

类型：ScheduleConfiguration 对象

必需：是

State

计划查询的状态。

类型：字符串

有效值：`ENABLED | DISABLED`

必需：是

CreationTime

计划查询的创建时间。

类型：时间戳

必需：否

ErrorReportConfiguration

计划查询的错误报告配置。

类型：ErrorReportConfiguration 对象

必需：否

KmsKeyId

客户提供的用于加密计划查询资源的密KMS钥。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

LastRunSummary

上次计划查询运行的运行时摘要。

类型：ScheduledQueryRunSummary 对象

必需：否

NextInvocationTime

下次计划运行定时查询的时间。

类型：时间戳

必需：否

PreviousInvocationTime

上次运行查询的时间。

类型：时间戳

必需：否

RecentlyFailedRuns

最近五次失败的计划查询运行的运行时摘要。

类型：ScheduledQueryRunSummary 对象数组

必需：否

ScheduledQueryExecutionRoleArn

IAMTimestream 用来运行计划查询的角色。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

TargetConfiguration

计划查询目标存储配置。

类型：TargetConfiguration 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ScheduledQueryInsights
服务: Amazon Timestream Query

封装用于在上启QueryInsights用的设置。ExecuteScheduledQueryRequest

内容

Mode

提供以下模式以启用ScheduledQueryInsights：

- ENABLED_WITH_RATE_CONTROL— ScheduledQueryInsights 为正在处理的查询启用。此模式还包括速率控制机制，该机制将该QueryInsights功能限制为每秒 1 次查询 (QPS)。
- DISABLED— 禁用。ScheduledQueryInsights

类型：字符串

有效值：ENABLED_WITH_RATE_CONTROL | DISABLED

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ScheduledQueryInsightsResponse
服务: Amazon Timestream Query

提供与已执行的相关的各种见解和指标。ExecuteScheduledQueryRequest

内容

OutputBytes

表示查询结果集的大小（以字节为单位）。作为查询调整练习的一部分，您可以使用这些数据来验证结果集是否已更改。

类型：长整型

必需：否

OutputRows

表示作为查询结果集一部分返回的总行数。作为查询调整练习的一部分，您可以使用这些数据来验证结果集中的行数是否发生了变化。

类型：长整型

必需：否

QuerySpatialCoverage

提供对查询的空间覆盖范围的见解，包括具有次优化（最大）空间修剪的表。这些信息可以帮助您确定分区策略中需要改进的领域，以增强空间修剪。

类型：QuerySpatialCoverage 对象

必需：否

QueryTableCount

表示查询中表的数量。

类型：长整型

必需：否

QueryTemporalRange

提供对查询时间范围的见解，包括时间范围最大（最大）的表。以下是一些优化基于时间的修剪的潜在选项：

- 添加缺失的时间谓词。
- 移除时间谓词周围的函数。
- 向所有子查询添加时间谓词。

类型：QueryTemporalRange 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

ScheduledQueryRunSummary
服务: Amazon Timestream Query

计划查询的运行摘要

内容

ErrorReportLocation

错误报告的 S3 位置。

类型：ErrorReportLocation 对象

必需：否

ExecutionStats

计划运行的运行时统计信息。

类型：ExecutionStats 对象

必需：否

FailureReason

失败时计划查询的错误消息。您可能需要查看错误报告才能获得更详细的错误原因。

类型：字符串

必需：否

InvocationTime

InvocationTime 为了这次跑步。这是计划运行查询的时间。`@scheduled_runtime`可以在查询中使用参数来获取值。

类型：时间戳

必需：否

QueryInsightsResponse

提供与计划查询的运行摘要相关的各种见解和指标。

类型：ScheduledQueryInsightsResponse 对象

必需：否

RunStatus

计划查询运行的状态。

类型：字符串

有效值：AUTO_TRIGGER_SUCCESS | AUTO_TRIGGER_FAILURE | MANUAL_TRIGGER_SUCCESS | MANUAL_TRIGGER_FAILURE

必需：否

TriggerTime

查询的实际运行时间。

类型：时间戳

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

SelectColumn
服务: Amazon Timestream Query

查询返回的列的详细信息。

内容

Aliased

如果查询给列名加了别名，则为 true。否则为假。

类型：布尔值

必需：否

DatabaseName

包含此列的数据库。

类型：字符串

必需：否

Name

列的名称。

类型：字符串

必需：否

TableName

数据库中包含此列的表。

类型：字符串

必需：否

Type

包含查询结果集中某列的数据类型。数据类型可以是标量或复数。支持的标量数据类型包括整数、布尔值、字符串、双精度、时间戳、日期、时间和间隔。支持的复杂数据类型包括数组、行和时间序列。

类型：Type 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

SnsConfiguration
服务: Amazon Timestream Query

发送通知时需要提供SNS这方面的详细信息。

内容

TopicArn

SNS计划查询状态通知将发送到的主题ARN。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Tag
服务: Amazon Timestream Query

标签是您分配给 Timestream 数据库and/or table. Each tag consists of a key and an optional value, both of which you define. Tags enable you to categorize databases and/or表的标签，例如按用途、所有者或环境进行分配。

内容

Key

标签的密钥。标签键区分大小写。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

必需：是

Value

标签的值。标签值区分大小写，可以为空值。

类型：字符串

长度约束：最小长度为 0。最大长度为 256。

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

TargetConfiguration
服务: Amazon Timestream Query

用于写入查询输出的配置。

内容

TimestreamConfiguration

将数据写入 Timestream 数据库和表所需的配置。

类型：TimestreamConfiguration 对象

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

TargetDestination
服务: Amazon Timestream Query

为目标数据源写入数据的目标详细信息。当前支持的数据源是 Timestream。

内容

TimestreamDestination

查询 Timestream 数据源的结果目标详细信息。

类型：TimestreamDestination 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

TimeSeriesDataPoint
服务: Amazon Timestream Query

时间序列数据类型表示一段时间内的度量值。时间序列是由时间戳和度量值行组成的数组，行按时间升序排序。A TimeSeriesDataPoint 是时间序列中的单个数据点。它表示时间序列中的一个元组（时间，度量值）。

内容

Time

收集测量值的时间戳。

类型：字符串

必需：是

Value

数据点的测量值。

类型：Datum 对象

必需：是

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

TimestreamConfiguration
服务: Amazon Timestream Query

将数据写入 Timestream 数据库和表的配置。此配置允许用户将查询结果选择列映射到目标表列。

内容

DatabaseName

将写入查询结果的 Timestream 数据库的名称。

类型：字符串

必需：是

DimensionMappings

此项允许将查询结果中的列映射到目标表中的维度。

类型：DimensionMapping 对象数组

必需：是

TableName

将写入查询结果的 Timestream 表的名称。该表应位于 Timestream 配置中提供的同一数据库。

类型：字符串

必需：是

TimeColumn

查询结果中的列，应用作目标表中的时间列。此列的类型应为TIMESTAMP。

类型：字符串

必需：是

MeasureNameColumn

度量列的名称。

类型：字符串

必需：否

MixedMeasureMappings

指定如何将度量映射到多度量记录。

类型：MixedMeasureMapping 对象数组

数组成员：最少 1 个物品。

必需：否

MultiMeasureMappings

多度量映射。

类型：MultiMeasureMappings 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

TimestreamDestination
服务: Amazon Timestream Query

计划查询的目的地。

内容

DatabaseName

时间流数据库名称。

类型：字符串

必需：否

TableName

时间流表名。

类型：字符串

必需：否

另请参阅

有关在特定语言API中使用它的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

Type
服务: Amazon Timestream Query

包含查询结果集中某列的数据类型。数据类型可以是标量或复数。支持的标量数据类型包括整数、布尔值、字符串、双精度、时间戳、日期、时间和间隔。支持的复杂数据类型包括数组、行和时间序列。

内容

ArrayColumnInfo

表示该列是否为数组。

类型：ColumnInfo 对象

必需：否

RowColumnInfo

表示该列是否为行。

类型：ColumnInfo 对象数组

必需：否

ScalarType

表示列的类型是否为字符串、整数、布尔值、双精度、时间戳、日期、时间。有关更多信息，请参阅支持的数据类型。

类型：字符串

有效值：VARCHAR | BOOLEAN | BIGINT | DOUBLE | TIMESTAMP | DATE | TIME | INTERVAL_DAY_TO_SECOND | INTERVAL_YEAR_TO_MONTH | UNKNOWN | INTEGER

必需：否

TimeSeriesMeasureValueColumnInfo

指示该列是否为时间序列数据类型。

类型：ColumnInfo 对象

必需：否

另请参阅

有关API在一种特定语言中使用此功能的更多信息 AWS SDKs，请参阅以下内容：

- AWS SDK对于 C++
- AWS SDK适用于 Java V2
- AWS SDK适用于 Ruby V3

常见错误

本节列出了所有 AWS 服务API操作中常见的错误。有关此服务某项API操作的特定错误，请参阅该API操作的主题。

AccessDeniedException

您没有足够的访问权限，无法执行该操作。

HTTP状态码：400

IncompleteSignature

请求签名不符合 AWS 标准。

HTTP状态码：400

InternalFailure

由于未知错误、异常或故障，请求处理失败。

HTTP状态码：500

InvalidAction

所请求的操作无效。确认正确键入了操作。

HTTP状态码：400

InvalidClientTokenId

我们的记录中不存在提供的 X.509 证书或 AWS 访问密钥 ID。

HTTP状态码：403

NotAuthorized

您无权执行此操作。

HTTP状态码：400

OptInRequired

AWS 访问密钥 ID 需要订阅该服务。

HTTP状态码：403

RequestExpired

请求在请求的日期戳后超过 15 分钟或请求到期日期（例如预签名URLs）超过 15 分钟到达服务，或者请求上的日期戳超过未来 15 分钟。

HTTP状态码：400

ServiceUnavailable

由于服务器发生临时故障而导致请求失败。

HTTP状态码：503

ThrottlingException

由于请求限制而导致请求被拒绝。

HTTP状态码：400

ValidationError

输入无法满足 AWS 服务指定的约束。

HTTP状态码：400

常见参数

以下列表包含所有操作用于使用查询字符串对 Signature Version 4 请求进行签名的参数。任何特定于操作的参数都列在该操作的主题中。有关签名版本 4 的更多信息，请参阅IAM用户指南中的签名 AWS API请求。

Action

要执行的操作。

类型：字符串。

必需：是

Version

请求所针对的API版本，以格式表示 YYYY-MM-DD。

类型：字符串。

必需：是

X-Amz-Algorithm

您用于创建请求签名的哈希算法。

条件：当您在查询字符串而不是HTTP授权标头中包含身份验证信息时，请指定此参数。

类型：字符串

有效值：`AWS4-HMAC-SHA256`

必需：条件

X-Amz-Credential

凭证范围值，该值是一个字符串，其中包含您的访问密钥、日期、您要定位的区域、您请求的服务以及终止字符串（“aws4_request”）。该值用以下格式表示：access_key//region/service YYYYMMDD/aws4_reque st。

有关更多信息，请参阅《IAM用户指南》中的创建已签名 AWS API请求。

条件：当您在查询字符串而不是HTTP授权标头中包含身份验证信息时，请指定此参数。

类型：字符串

必需：条件

X-Amz-Date

用于创建签名的日期。格式必须是 ISO 8601 基本格式（YYYYMMDD'T' 'ZHHMMSS'）。例如，以下日期时间是有效 X-Amz-Date值：`20120325T120000Z`。

条件： X-Amz-Date对于所有请求都是可选的；它可用于覆盖用于签署请求的日期。如果日期标题以 ISO 8601 基本格式指定，则 X-Amz-Date不是必填项。使用 X-Amz-Date时，它总是会覆盖 Date 标题的值。有关更多信息，请参阅《IAM用户指南》中的 AWS API请求签名元素。

类型：字符串

必需：条件

X-Amz-Security-Token

通过调用 AWS Security Token Service (AWS STS) 获得的临时安全令牌。有关支持临时安全证书的服务列表 AWS STS，请参阅《IAM用户指南》AWS 服务 IAM中与之配合使用的服务。

条件：如果您使用的是中的临时安全证书 AWS STS，则必须包含安全令牌。

类型：字符串

必需：条件

X-Amz-Signature

指定从要签名的字符串和派生的签名密钥计算的十六进制编码签名。

条件：当您在查询字符串而不是HTTP授权标头中包含身份验证信息时，请指定此参数。

类型：字符串

必需：条件

X-Amz-SignedHeaders

指定规范HTTP请求中包含的所有标头。有关指定签名标头的更多信息，请参阅IAM用户指南中的创建签名 AWS API请求。

条件：当您在查询字符串而不是HTTP授权标头中包含身份验证信息时，请指定此参数。

类型：字符串

必需：条件

文档历史记录

变更	说明	日期
仅限文档的更新	更新了配额主题以区分默认配额和系统限制。	2024年10月22日
亚马逊 Timestream 现在支持查询见解	Timestream 现在支持查询见解功能，该功能可帮助您优化查询、提高查询性能并降低成本。	2024年10月22日

适用于 InfluxDB 的亚马逊 Timestream 更新了现有政策。	适用于 InfluxDB 的 Amazon Timestream 已将该`ec2:DescribeRouteTables` 操作添加到现有`AmazonTimestreamInfluxDBFullAccess` 托管策略中，用于描述您的路由表	2024年10月8日
`AmazonTimestreamReadOnlyAccess` — 更新现有政策	的 Timestream LiveAnalytics 已向`AmazonTimestreamReadOnlyAccess` 托管策略添加了描述 AWS 账户 设置的`DescribeAccountSettings` 权限。	2024 年 6 月 3 日
亚马逊 Timestream LiveAnalytics 现在支持 Timestream 计算单位 () TCUs	LiveAnalytics 目前，Amazon Timestream 支持时间流计算单位 (TCUs)，用于衡量为满足您的查询需求分配的计算容量。	2024 年 4 月 29 日
添加了新政策	适用于 InfluxDB 的 Amazon Timestream 添加了两项新策略：一项允许该服务管理您账户中的网络接口和安全组。有关更多信息，请参阅AmazonTimestreamInfluxDBServiceRolePolicy。另一个提供创建、更新、删除和列出 Amazon Timestream InfluxDB 实例以及创建和列出参数组的完全管理权限。有关更多信息，请参阅AmazonTimestreamInfluxDBFullAccess。	2024 年 3 月 14 日
适用于 InfluxDB 的亚马逊 Timestream 现已正式上市。	本文档涵盖了适用于 InfluxDB 的 Amazon Timestream 的初始版本。	2024 年 3 月 14 日

Amazon Timestream for LiveAnalytics Query 事件可用于 AWS CloudTrail	Amazon Timestream LiveAnalytics 现在向发布查询API数据事件。 AWS CloudTrail客户可以审核在其 AWS 账户中提出的所有查询API请求，并查看诸如哪个IAM用户/角色发出了请求、何时发出请求、查询了哪些数据库和表以及请求的查询 ID 等信息。	2023 年 9 月 12 日
适用于 Amazon Timestream LiveAnalytics UNLOAD	Amazon Timestream LiveAnalytics 现在支持UNLOAD将查询结果导出到 S3。	2023 年 5 月 12 日
Amazon Timestream 用于 LiveAnalytics 更新现有政策。	Batch 加载权限已添加到托管策略。	2023 年 2 月 24 日
用于 LiveAnalytics 批量加载的 Amazon Timestream。	亚马逊 Timestream LiveAnalytics 目前支持批量加载功能。	2023 年 2 月 24 日
亚马逊 Timestream LiveAnalytics 目前支持。 AWS Backup	亚马逊 Timestream LiveAnalytics 目前支持。 AWS Backup	2022 年 12 月 14 日
Amazon Timestream 获取托 LiveAnalytics 管政策的 AWS 更新	有关 AWS 托管策略和 Amazon Timestream 的新信息 LiveAnalytics，包括对现有托管策略的更新。	2021 年 11 月 29 日
适用于 Amazon Timestream 的 LiveAnalytics 支持预定查询	Amazon Timestream LiveAnalytics 目前支持根据计划代表您运行查询。	2021 年 11 月 29 日
适用于的亚马逊 Timestream LiveAnalytics 支持磁性存储。	Amazon Timestream LiveAnalytics 现在支持使用磁性存储进行表写入。	2021 年 11 月 29 日

用于 LiveAnalytics 多指标记录的 Amazon Timestream。	LiveAnalytics 目前，Amazon Timestream 支持更紧凑的格式来存储您的时间序列数据。	2021 年 11 月 29 日
Amazon Timestream 获取托 LiveAnalytics 管政策的AWS更新	有关 AWS 托管策略和 Amazon Timestream 的新信息 LiveAnalytics，包括对现有托管策略的更新。	2021 年 5 月 24 日
Amazon Timestream LiveAnalytics 版现已在欧洲（法兰克福）地区上市。	Amazon Timestream 版现已在欧洲（法兰克福）地区正式上市（`eu-central-1` ）。LiveAnalytics	2021 年 4 月 23 日
适用于 Amazon Timestream 的 LiveAnalytics 版本现在支持 VPC终端节点 ()AWS PrivateLink。	Amazon Timestream LiveAnalytics 目前支持使用 VPC终端节点 ()AWS PrivateLink。	2021 年 3 月 23 日
亚马逊 Timestream 现在支持跨表查询。	您可以使用 Amazon Timestream LiveAnalytics 来运行跨表查询。	2021 年 2 月 10 日
Amazon Timestream LiveAnalytics 目前支持增强的查询执行统计数据。	Amazon Timestream LiveAnalytics 现在支持增强的查询执行统计数据，例如扫描的数据量。	2021 年 2 月 10 日
Amazon Timestream LiveAnalytics 目前支持高级时间序列函数。	您可以使用 Amazon Timestream LiveAnalytics 来运行带有高级时间序列函数的SQL查询，例如导数、积分和相关性。	2021 年 2 月 10 日
现在HIPAA的亚马逊 Time LiveAnalytics stream 已合ISO规。PCI	现在，您可以将 Amazon Timestream LiveAnalytics 用于需要和PCI合HIPAA规ISO基础设施的工作负载。	2021 年 1 月 27 日

亚马逊 Timestream LiveAnalytics 目前支持开源 Telegraf 和 grafana。	现在，您可以使用开源、插件驱动的服务器代理 Telegraf 来收集和报告指标，也可以使用 Grafana（数据库的开源分析和监控平台）和 Amazon Timestream 的开源分析和监控平台。 LiveAnalytics	2020 年 11 月 25 日
适用于 Amazon Timestream 的 LiveAnalytics 版现已正式上市。	本文档涵盖了 Amazon Timestream 的初始版本。 LiveAnalytics	2020 年 9 月 30 日

InfluxDB 的时间流是什么？

适用于InfluxDB的Amazon Timestream是一个托管的时间序列数据库引擎，它使应用程序开发人员和 DevOps 团队可以轻松地使用开源为实时时间序列应用程序运行InfluxDB数据库。 AWS APIs借助适用于 InfluxDB 的 Amazon Timestream，可以轻松设置、操作和扩展时间序列工作负载，这些工作负载可以用个位数毫秒的查询响应时间来回答查询。

适用于 InfluxDB 的 Amazon Timestream 允许你在其 2.x 分支上访问熟悉的 InfluxDB 开源版本的功能。这意味着你现在已经在现有InfluxDB开源数据库中使用的代码、应用程序和工具应该可以与适用于InfluxDB的Amazon Timestream无缝协作。适用于 InfluxDB 的 Amazon Timestream 可以自动备份您的数据库，并使您的数据库软件保持最新版本。此外，适用于 InfluxDB 的 Amazon Timestream 可以轻松使用复制来增强数据库可用性并提高数据持久性。与所有 AWS 服务一样，无需前期投资，您只需为使用的资源付费。

数据库实例

数据库实例是在云中运行的独立数据库环境。它是 InfluxDB 版 Amazon Timestream 的基本组成部分。一个数据库实例可以包含多个用户创建的数据库（对于 InfluxDb 2.x数据库，则可以包含组织和存储桶），并且可以使用与访问独立的自管理的InfluxDB实例相同的客户端工具和应用程序进行访问。使用 AWS 命令行工具、Amazon Timestream InfluxDB API 操作或，可以轻松创建和修改数据库实例。AWS Management Console

> **Note**
>
> 适用于 InfluxDB 的 Amazon Timestream 支持使用 Influx 操作和 Influx API 用户界面访问数据库。适用于 InfluxDB 的 Amazon Timestream 不允许直接访问主机。

InfluxDB 实例最多可以有 40 个 Amazon Timestream。

每个数据库实例都有一个数据库实例名称。在与适用于 API Influ AWS CLI xDB 的 Amazon Timestream 和命令进行交互时，客户提供的这个名称可以唯一标识数据库实例。该客户在一个 AWS 区域中的数据库实例名称必须是唯一的。

数据库实例名称是 Timestream 为 InfluxDB 分配给您的实例DNS的主机名的一部分。例如，如果您将 influxdb1 指定为数据库实例名称，Timestream 将自动为您的实例分配一个DNS终端节点。示例终端节

点是influxdb1-3ksj4dla5nfjhi.us-east-1.timestream-influxdb.amazonaws.com，其中influxdb1是您的实例名称。

在示例端点中influxdb1-3ksj4dla5nfjhi.us-east-1.timestream-influxdb.amazonaws.com，字符串3ksj4dla5nfjhi是由生成的唯一账户标识符 AWS。示例3ksj4dla5nfjhi中特定区域中指定账户的标识符不会发生变化。因此，该账户创建的所有数据库实例共享相同的固定标识符。考虑固定标识符的以下特征：

- 目前 InfluxDB 的 Timestream 不支持数据库实例重命名。
- 如果您删除并重新创建具有相同数据库实例标识符的数据库实例，则端点相同。
- 如果您使用同一个账户在不同的区域创建数据库实例，则内部生成的标识符会有所不同，因为该区域不同，如在 influxdb2.4a3j5du5ks7md2.us-west-1.timestream-influxdb.amazonaws.com 中。

每个数据库实例仅支持一个 InfluxDB 数据库引擎的时间流。

创建数据库实例时，InfluxDB 要求指定组织名称。一个数据库实例可以托管多个组织和与每个组织关联的多个存储桶。

InfluxDB 版 Amazon Timestream 允许您在创建过程中为数据库实例创建主用户账户和密码。该主用户有权创建组织、存储桶，以及对您的数据执行读取、写入、删除和更新插入操作。您还可以访问 InfluxUI并在首次登录时检索操作员令牌。从那里，您还可以管理所有访问令牌。创建数据库实例时必须设置主用户密码，但您可以随时使用 Influx、Influx 或 Influx API U CLI I 对其进行更改。

数据库实例类

数据库实例类决定了 Amazon fi UbfkyxDB Timestream 数据库实例的计算和内存容量。您需要的数据库实例类取决于您的处理能力和内存要求。

数据库实例类由数据库实例类类型和大小共同组成。例如，db.influx是一种内存优化的数据库实例类类型，适用于与正在运行 InfluxDb 的工作负载相关的高性能内存要求。db.influx实例类类型中db.influx.2xlarge有一个数据库实例类。这个班级的大小是 2x large。

有关实例类定价的更多信息，请参阅亚马逊 Timestream for InfluxDB 定价。

数据库实例类类型

适用于 InfluxDB 的 Amazon Timestream 支持以下用例的数据库实例类，这些用例针对 InfluxDB 用例进行了优化。

- **db.influx**—这些实例类非常适合在开源 InfluxDB 数据库中运行内存密集型工作负载

数据库实例类的硬件规格

以下术语描述了数据库实例类的硬件规格：

- v CPU

  虚拟中央处理单元的数量 (CPUs)。虚拟CPU是一种容量单位，可用于比较数据库实例类别。

- 内存 (GiB)

  分配给数据库实例的（以 Gibibytes 为单位）。RAM内存和 v 之间的比例通常是一致的CPU。举个例子，以 db.influx 实例类为例，它的内存与 v 之CPU比类似于 EC2 r7g 实例类。

- 针对涌入进行了优化

  数据库实例使用经过优化的配置堆栈，并为输入/输出提供额外的专用容量。这种优化通过最小化输入/输出与来自您实例的其他流量之间的争用，为您提供最佳性能。

- 网络带宽

  与其他数据库实例类有关的网络速度。在下表中，您可以找到有关适用于 InfluxDB 的 Amazon Timestream 实例类的硬件详细信息。

实例类	v CPU	内存 (GiB)	存储类型	网络带宽 (Gbps)
db.infulx.medium	1	8	包括涌入 IOPS	10
db.influx.large	2	16	包括涌入 IOPS	10
db.influx.xlarge	4	32	包括涌入 IOPS	10
db.influx.2xlarge	8	64	包括涌入 IOPS	10
db.influx.4xlarge	16	128	包括涌入 IOPS	10
db.influx.8xlarge	32	256	包括涌入 IOPS	12
db.influx .12xlarge	48	384	包括涌入 IOPS	20

实例类	v CPU	内存 (GiB)	存储类型	网络带宽 (Gbps)
db.influx .16xlarge	64	512	包括涌入 IOPS	25

InfluxDB 实例存储

适用于 InfluxDB 的 Amazon Timestream 的数据库实例使用 Influx In IOPS cluded 卷存储数据库和日志存储。

在某些情况下，您的数据库工作负载可能无法达到您预配置IOPS的 100%。有关更多信息，请参阅影响存储性能的因素。有关 Timestream for InfluxDB 存储定价的更多信息，请参阅亚马逊 Timestream 定价。

适用于 InfluxDB 存储类型的亚马逊 Timestream

适用于 InfluxDB 的 Amazon Timestream 支持一种存储类型，即包含 Influx。IOPS你可以为存储空间高达 16 太字节 (TiB) 的 InfluxDB 实例创建 Timestream。

以下是可用存储类型的简要描述：

- Influx IO 包含存储：存储性能是每秒 I/O 操作数 (IOPS) 和存储卷执行读取和写入的速度（存储吞吐量）的组合。在 Influx Included 存储卷上，适用于 InfluxDB 的 Amazon Timestream 提供了 3 个存储层，预先配置了不同类型的工作负IOPS载所需IOPS的最佳存储层和吞吐量。

InfluxDB 实例大小调整

InfluxDB 实例的 Timestream 的最佳配置取决于许多因素，包括摄取率、批次大小、时间序列基数、并发查询和查询类型。为了提供规模建议，我们将重点放在具有以下特征的示例性工作负载上：

- 数据由一组 Telegraf 代理收集和写入，这些代理从数据中心收集系统CPU、内存、磁盘、IO 等。

  每个写入请求包含 5000 行。
- 在系统上执行的查询类型被归类为 “中等复杂性” 查询。此类查询具有以下特征：
  - 有多个函数和一两个正则表达式
  - 也可以按子句分组，或者采样时间范围为多周。
  - 通常需要几百毫秒到几千毫秒才能执行。

- CPU主要有利于查询性能。

实例类	存储类型	写入（每秒行数）	读取（每秒查询数）
db.influx.large	包括 Influx IO 3K	约 50,000	<10
db.influx.2xlarge	包括 Influx IO 3K	大约 15万	<25
db.influx.4xlarge	包括 Influx IO 3K	大约 20 万	~25
db.influx.4xlarge	包括 Influx IO 12K	大约 25 万	~35
db.influx.8xlarge	包括 Influx IO 12K	约 500,000	~50
db.influx.12xlarge	包括 Influx IO 12K	<750,000	<100

AWS 区域和可用区

Amazon 云计算资源在全球多个位置托管。这些位置由 AWS 区域和可用区组成。每个 AWS 区域都是一个独立的地理区域。每个 AWS 区域都有多个被称为可用区的隔离位置。

> **Note**
>
> 有关查找某个 AWS 区域的可用区域的信息，请参阅 Amazon EC2 用户指南中的区域和区域。

适用于 InfluxDB 的 Amazon Timestream 允许您将数据库实例和数据等资源放在多个位置。

Amazon 运营着 state-of-the-art高度可用的数据中心。数据中心有时会发生影响托管于同一位置的所有数据库实例的可用性的故障，虽然这种故障极少发生。如果您将所有数据库实例都托管在受此类故障影响的同一个位置，则您的所有数据库实例都将不可用。

请务必记住，每个 AWS 区域都是完全独立的。您启动的任何 Amazon Timestream for InfluxDB 活动（例如，创建数据库实例或列出可用的数据库实例）都只能在您当前的默认区域中运行。 AWS 可以在控制台中或通过设置 AWS_DEFAULT_REGION 环境变量更改原定设置 AWS 区域。或者可以通过使用带有 AWS Command Line Interface ()AWS CLI的--region参数来覆盖它。有关更多信息，请参阅配置 AWS Command Line Interface，特别是有关环境变量和命令行选项的部分。

要在特定 AWS 区域创建或使用适用于 InfluxDB 的 Amazon Timestream 数据库实例，请使用相应的区域服务终端节点。

AWS 地区可用性

有关目前提供适用于 InfluxDB 的 Amazon Timestream 的 AWS 区域以及每个区域的终端节点的更多信息，请参阅 Amazon Timestream 终端节点和配额。

AWS 区域设计

每个 AWS 区域都被设计为与其他 AWS 区域隔离。此设计可实现最大程度的容错能力和稳定性。

当您查看您的资源时，您只能看到与您指定的 AWS 区域相关的资源。这是因为 AWS 区域彼此隔离，而且我们不会自动跨 AWS 区域复制资源。

AWS 可用区

创建数据库实例时，适用于 InfluxDB 的 Amazon Timestream 会根据您的子网配置随机选择一个。可用区由 AWS 区域代码和字母标识符（例如`us-east-1a`）表示。

使用如下所示的 `describe-availability-zones` Amazon EC2 命令来描述指定区域内为您的账户启用的可用区。

```
aws ec2 describe-availability-zones --region region-name
```

例如，要描述为您的账户启用的美国东部（弗吉尼亚北部）区域 (us-east-1) 内的可用区，请运行以下命令：

```
aws ec2 describe-availability-zones --region us-east-1
```

在多可用区数据库部署中，您无法为主数据库实例和辅助数据库实例选择可用区。适用于 InfluxDB 的 Amazon Timesteram 会随机为你选择它们。有关多可用区部署的更多信息，请参阅... 配置和管理多可用区部署

适用于 InfluxDB 的 Amazon Timestream 的数据库实例账单

InfluxDB 实例的 Amazon Timestream 按以下组成部分计费：

- 数据库实例小时数（每小时）-基于数据库实例的数据库实例类别，例如 db.influx.large。定价以每小时为单位列出，但账单向下计算至秒，并以十进制形式显示时间。InfluxDB 使用量的 Amazon Timestream 以 1 秒为增量计费，最少为 10 分钟。有关更多信息，请参阅数据库实例类数据库实例类。
- 存储（每月每 GiB）— 您为数据库实例预配置的存储容量。有关更多信息，请参阅 InfluxDB 实例存储。
- 数据传输（每 GB）-从您的数据库实例传入和传出互联网和其他 AWS 地区的数据。

有关 InfluxDB 的亚马逊 Timestream 定价信息，请参阅亚马逊 Timestream for InfluxDB 定价页面。

为 InfluxDB 设置亚马逊 Timestream

在首次使用适用于 InfluxDB 的 Amazon Timestream 之前，请完成以下任务：

如果你已经有一个 AWS 账户，请了解你的 Amazon Timestream 对于 InfluxDB 的要求，并且更喜欢使用 InfluxDB 的 Amazon Timestream IAM 和 “VPCInfluxDB 的 Timestream 入门入门” 的默认设置。

注册一个 AWS 账号

如果您没有 AWS 帐户，请完成以下步骤来创建一个帐户。

要注册一个 AWS 账户

- 前往 “AWS 登录” 页面。
- 选择 “创建新帐户”，然后按照说明进行操作。

> **Note**
>
> 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。

当你注册一个 AWS 账户时，会创建一个 AWS 账户 root 用户。root 用户有权访问账户中的所有 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。您可以随时前往 https://aws.amazon.com/并选择 “我的账户”，查看您当前的账户活动并管理您的账户。

用户管理

创建管理用户

创建管理用户

注册 AWS 帐户后，请创建一个管理用户，这样您就不会使用 root 用户执行日常任务。

保护您的 AWS 账户 root 用户

选择 Root 用户并输入您的账户电子邮件地址，以账户所有者的身份 AWS 登录 AWS 管理控制台。在下一页上，输入您的密码。有关使用 root 用户登录的帮助，请参阅《登录用户指南》中的以 root 用户身份AWS登录

为您的 root 用户开启多重身份验证 (MFA)。有关说明，请参阅《用户指南》中的为您的 AWS 账户 root 用户（控制台）启用虚拟MFA设备。IAM

授予编程访问权限

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程式访问权限的方法取决于访问 AWS的用户类型。

要向用户授予编程访问权限，请选择以下选项之一：

哪个用户需要编程式访问权限？	目的	方式
员工身份（在IAM身份中心管理的用户）	使用临时证书签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	按照您要使用的接口的说明进行操作。* 有关，请参阅 AWS CLI 配置 AWS CLI为使用 AWS IAM身份中心 在 AWS 命令行界面用户指南 .* 有关 AWS SDKs工具和 AWS APIs，请参阅 IAM身份中心身份验证 在 AWSSDKs和《工具参考指南》。
IAM	使用临时证书签署对 AWS CLISDKs、和的编程请求APIs。	按照IAM用户指南中的将临时证书与 AWS 资源配合使用中的说明进行操作。
IAM	（不推荐）使用长期凭证签署对 AWS CLISDKs、和的编程请求APIs。	按照您要使用的接口的说明进行操作。有关的，请参阅 AWS CLI

哪个用户需要编程式访问权限？	目的	方式
		使用IAM用户凭证进行身份验证 在 AWS 命令行界面用户指南 .有关 AWS SDKs和工具，请参阅 使用长期凭证进行身份验证 在 AWS SDKs和工具参考指南 。对于 AWS APIs，请参阅 管理IAM用户的访问密钥 在 IAM用户指南。

确定要求

适用于 Influx 的 Amazon Timestream 的基本构建块是数据库实例。在数据库实例中，您可以创建自己的存储桶。数据库实例提供了称为终端节点 的网络地址。您的应用程序使用此终端节点连接到您的数据库实例。您还将使用浏览器中的相同端点访问您的InfluxUI。创建数据库实例时，需要指定存储、内存、数据库引擎和版本、网络配置和安全性等详细信息。您通过安全组控制对数据库实例的网络访问。

在创建数据库实例和安全组之前，您必须知道数据库实例和网络需求。下面是要考虑的一些重要事项：

- 资源需求-您的应用程序或服务的内存和处理器要求是什么？ 您使用这些设置来帮助您确定要使用哪个数据库实例类。有关数据库实例类的规范，请参阅数据库实例类。
- VPC和安全组 — 您的数据库实例很可能位于虚拟私有云中 (VPC)。要连接到数据库实例，您需要设置安全组规则。根据VPC您使用的类型和使用方式，这些规则的设置会有所不同。例如，你可以使用：默认值VPC或用户定义的VPC。

  以下列表描述了每个VPC选项的规则：

  - 默认 VPC-如果您的 AWS 账户VPC在当前 AWS 区域有默认值，VPC则该账户配置为支持数据库实例。如果您在创建数据库实例VPC时指定了默认值，请务必创建一个VPC安全组来授权从应用程序或服务连接到 Amazon Timestream for InfluxDB 数据库实例。使用VPC控制台上的安全组选项或创建VPC安全组。 AWS CLI有关更多信息，请参阅步骤 3：创建VPC安全组。
- 用户定义 VPC-如果您想在创建数据库实例VPC时指定用户定义的实例，请注意以下几点：
  - 请务必创建一个VPC安全组，以授权从应用程序或服务连接到 Amazon Timestream for InfluxDB 数据库实例。使用VPC控制台上的安全组选项或创建VPC安全组。 AWS CLI有关信息，请参阅步骤 3：创建VPC安全组。
  - VPC必须满足某些要求才能托管数据库实例，例如至少有两个子网，每个子网位于单独的可用区。有关信息，请参阅适用于 InfluxD VPCVPCsB 的亚马逊和亚马逊 Timestream。
- 高可用性-您需要故障转移支持吗？ 在适用于 InfluxDB 的 Amazon Timestream 上，多可用区部署在另一个可用区中创建一个主数据库实例和一个辅助备用数据库实例，以支持故障转移。我们建议将多可用区部署用于生产工作负载以保持高可用性。对于开发和测试用途，您可以使用不是多可用区的部署。有关更多信息，请参阅 多可用区数据库实例部署。
- IAM政策 — 您的 AWS 账户是否有授予执行 Amazon Timestream 对 InfluxDB 操作所需的权限的策略？ 如果您 AWS 使用IAM凭证进行连接，则您的IAM账户必须具有授予执行 Amazon Timestream for InfluxDB 控制平面操作所需的权限的IAM策略。有关更多信息，请参阅 适用于 InfluxDB 的亚马逊 Timestream 的身份和访问管理。
- 开放端口-您的数据库监听哪TCP个 /IP 端口？ 有些公司的防火墙可能会阻止与您的数据库引擎的原定设置端口进行连接。InfluxDB 的 Timestream 的默认值为 8086。
- AWS 区域-您希望您的数据库位于哪个 AWS 区域？ 通过让数据库紧邻应用程序或 Web 服务，可以减小网络延迟。有关更多信息，请参阅 AWS 区域和可用区。
- 数据库磁盘子系统-您的存储要求是什么？ 适用于 InfluxDB 的 Amazon Timestream 为其提供了三种配置 Influx IOPS 包含的存储类型：：
  - 包括 Influx Io 3k () IOPS SSD
  - 包括 Influx Io 12k () IOPS SSD
  - 包括 Influx Io 25k () IOPS SSD

  有关适用于 InfluxDB 存储的 Amazon Timestream 的更多信息，请参阅适用于 InfluxDB 数据库实例存储的亚马逊 Timestream。在了解创建安全组和数据库实例所需的信息之后，请继续执行下一步。

VPC通过创建安全组来提供对您中数据库实例的访问权限

VPC安全组提供对中数据库实例的访问权限VPC。它们充当关联数据库实例的防火墙，在数据库实例级别控制入站和出站流量。默认情况下，系统将创建带防火墙和默认安全组 (用于保护数据库实例) 的数据库实例。

在连接到数据库实例之前，必须先向允许连接的安全组添加规则。使用网络和配置信息来创建允许访问数据库实例的规则。

例如，假设您有一个应用程序可以访问中数据库实例上的数据库。VPC在这种情况下，必须添加自定义TCP规则，指定应用程序用于访问数据库的端口范围和 IP 地址。如果您在 Amazon EC2 实例上有应用程序，则可以使用为该亚马逊EC2实例设置的安全组。

创建用于VPC访问的安全组

要创建VPC安全组，请登录 AWS Management Console 并选择VPC。

> **Note**
>
> 确保你在VPC控制台中，而不是使用适用于 InfluxDB 的 Amazon Timesteam 控制台。

- 在的右上角 AWS Management Console，选择要在其中创建VPC安全组和数据库实例的AWS区域。在该 AWS 区域的 Amazon VPC 资源列表中，您应该至少看到一个VPC和多个子网。如果不这样做，则该 AWS 区域就没有默认值VPC。 。
- 在导航窗格中，选择安全组。
- 选择Create security group（创建安全组）。
- 在安全组页面的基本详细信息部分，输入安全组名称和描述。对于 VPC，选择VPC要在其中创建数据库实例的。
- 在 Inbound rules (入站规则) 中，请选择 Add rule (添加规则)。
  - 对于 “类型”，选择 “自定义” TCP。
  - 对于 S ourc e，选择安全组名称或输入访问数据库实例的 IP 地址范围（CIDR值）。如果您选择我的 IP，这会允许从浏览器中检测到的 IP 地址访问数据库实例。

  对于 Source，选择安全组名称或键入访问数据库实例的 IP 地址范围（CIDR值）。如果您选择我的 IP，这会允许从浏览器中检测到的 IP 地址访问数据库实例。
- （可选）在 Outbound rules (出站规则) 中，为出站流量添加规则。默认情况下，允许所有出站流量。

- 选择创建安全组。

在创建数据库实例时，您可以将此VPC安全组用作数据库实例的安全组。

> **Note**
>
> 如果您使用默认子网组VPC，则会为您创建一个跨越所有子网VPC的默认子网组。创建数据库实例时，可以选择默认 eifccntf，然后为数据库子网VPC组选择默认值。

完成所需的设置后，可以使用您的要求和安全组来创建数据库实例。为此，请遵循创建数据库实例中的说明。

InfluxDB 的 Timestream 入门

在以下示例中，您可以了解如何使用适用于 InfluxDB 服务的亚马逊 Timestream 创建和连接数据库实例。

> **Note**
>
> 务必先完成为 InfluxDB 设置亚马逊 Timestream部分中的任务，然后才能创建或连接到数据库实例。

主题

- 为 InfluxDB 实例创建并连接到 Timestream
- 为您的 InfluxDB 实例创建一个新的操作员令牌

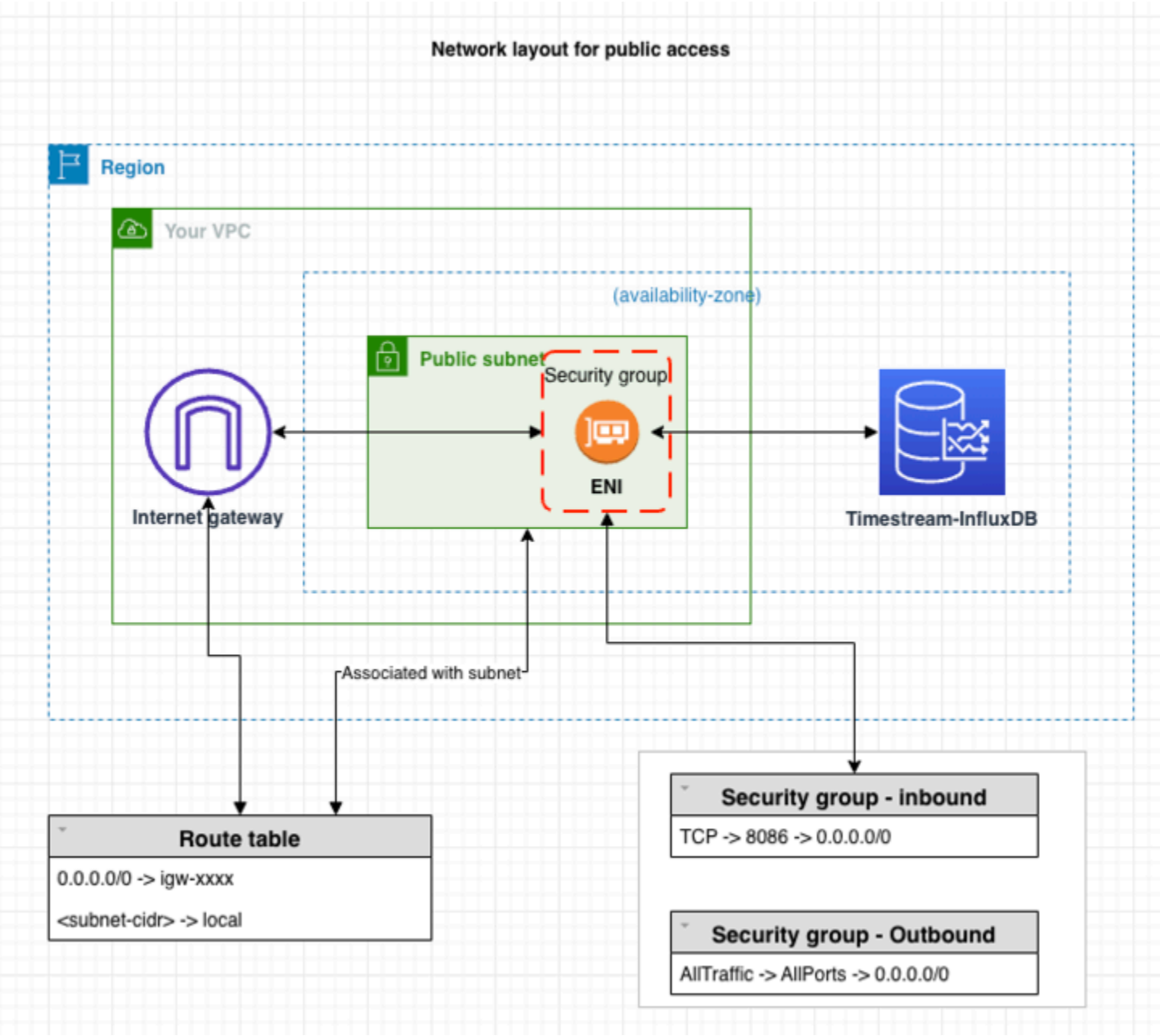
为 InfluxDB 实例创建并连接到 Timestream

本教程为 InfluxDB EC2 数据库实例创建了一个亚马逊实例和一个 Amazon Timestream。本教程向您展示如何使用 Telegraf 客户端将数据从EC2实例写入数据库实例。作为最佳实践，本教程在虚拟私有云中创建私有数据库实例 (VPC)。在大多数情况下VPC，相同的其他资源（例如EC2实例）可以访问数据库实例，但外部的资源VPC无法访问该实例。

完成本教程后，您的每个可用区中都有一个公有子网和私有子网VPC。在一个可用区中，EC2实例位于公有子网中，数据库实例位于私有子网中。

> **Note**
>
> 创建 AWS 账户不收取任何费用。但是，完成本教程后，您使用的 AWS 资源可能会产生费用。完成本教程后，如果不再需要这些资源，可以将其删除。

下图显示了可访问性公开时的配置。



> **Warning**
>
> 我们不建议使用 0.0.0.0/0 进行HTTP访问，因为您可以让所有 IP 地址通过访问您的公有 InfluxDB 实例。HTTP在测试环境中，这种方法在短时间内甚至不可接受。仅授权特定的 IP 地址或地址范围访问您的 InfluxDB 实例，使用 HTTP being for Webui 或访问权限。API

本教程使用创建了一个运行 InfluxDB 的数据库实例。 AWS Management Console我们将只关注数据库实例大小和数据库实例标识符。我们将使用其他配置选项的默认设置。此示例创建的数据库实例将是私有的。

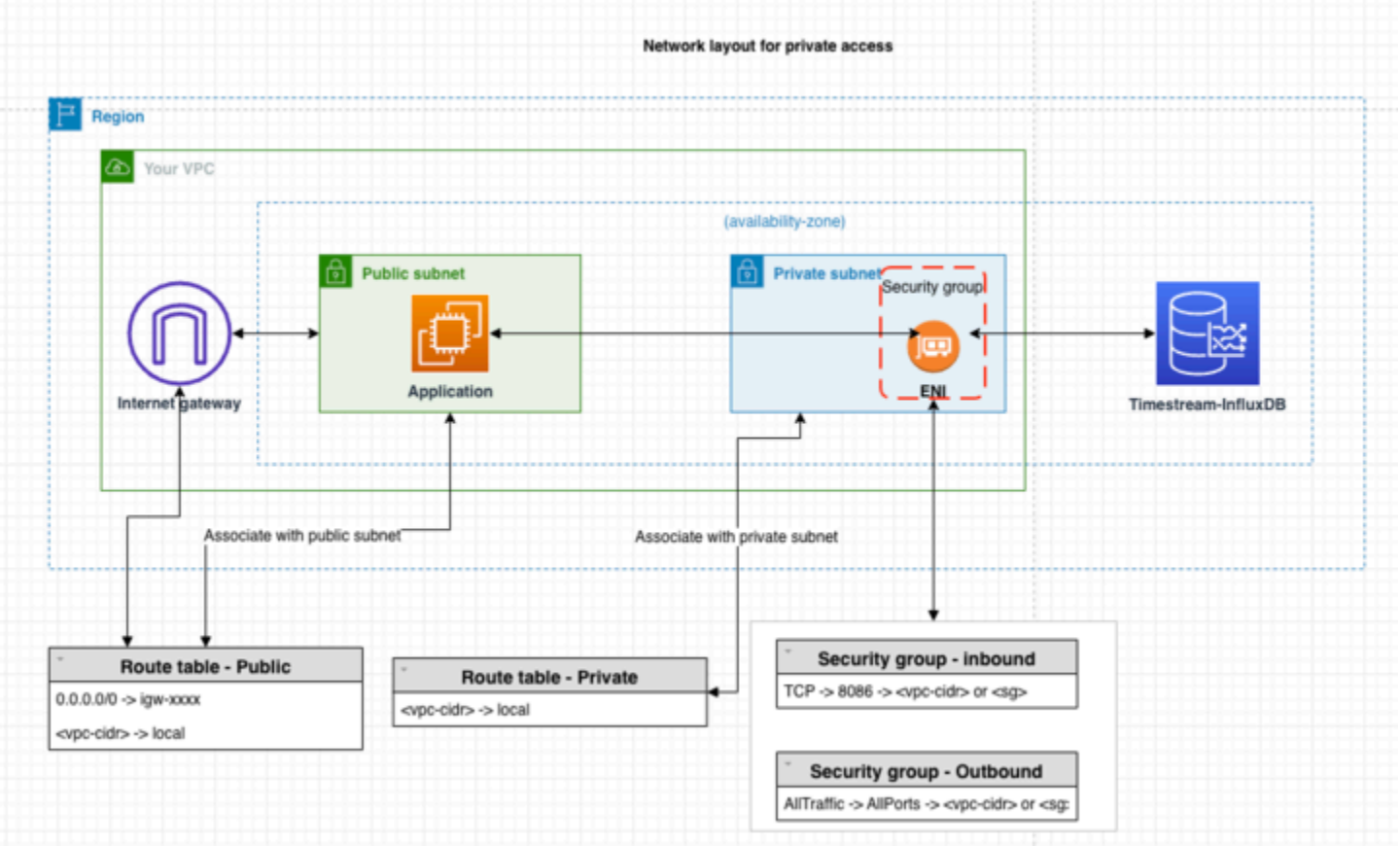
您可以配置的其他设置包括可用性、安全性和日志记录。要创建公有数据库实例，您必须在连接配置部分选择将您的实例设为 “可公开访问”。有关创建数据库实例的信息，请参阅创建数据库实例...

如果您的实例不可公开访问，请执行以下操作：

- 在实例上创建主机，通过该主机传输流量。VPC
- 设置到实例的 ssh 隧道。有关更多信息，请参阅使用 S AWS ystems Manager 转发亚马逊EC2实例端口
- 为了使证书生效，请将以下行添加到您的客户端计算机`/etc/hosts`的文件中：`127.0.0.1`。这是您的实例DNS的地址。
- 使用完全限定的域名连接到您的实例，例如 https: //< DNS >: 8086。

> **Note**
>
> Localhost 无法验证证书，因为 localhost 不是证书的一部分。SAN

下图显示了可访问性为私有时的配置：



先决条件

在开始之前，请完成以下各节中的步骤：

- 注册一个 AWS 账户。
- 创建管理用户。

步骤 1：创建亚马逊EC2实例

创建用于连接数据库的 Amazon EC2 实例。

1. 登录 AWS Management Console 并打开 Amazon EC2 控制台，网址为https://console.aws.amazon.com/ec2/。
2. 在的右上角 AWS Management Console，选择要在其中创建实例的 AWS EC2区域。
3. 选择 “EC2控制面板”，然后选择 “启动实例”。
4. 启动实例页面打开后，在启动实例页面上选择以下设置。

    a. 在名称和标签下，在名称中输入 ec2-database-connect。

b. 在 “应用程序和操作系统映像（亚马逊系统映像）” 下，选择亚马逊 Linux，然后选择亚马逊 Linux 2023 AMI。对于其他选项，保留默认选择。

c. 在 Instance type（实例类型）下，选择 t2.micro。

d. 在 Key pair (login) [密钥对（登录）] 下，选择 Key pair name（密钥对名称）以使用现有密钥对。要为 Amazon EC2 实例创建新的密钥对，请选择创建新密钥对，然后使用创建密钥对窗口来创建密钥对。有关创建新密钥对的更多信息，请参阅《Amazon Linux 实例EC2用户指南》中的创建密钥对。

e. 在 “网络设置” 中允许SSH流量，选择EC2实例的SSH连接来源。如果显示的 IP 地址对SSH连接来说是正确的，则可以选择 “我的 IP”。否则，您可以使用 Secure Shell (SSH) 确定用于连接到您VPC中的EC2实例的 IP 地址。要确定您的公有 IP 地址，可以在不同的浏览器窗口或选项卡中使用该服务 https://checkip.amazonaws.com。IP 地址的示例为 192.0.2.1/32。在许多情况下，您可以通过互联网服务提供商 (ISP) 或在没有静态 IP 地址的情况下从防火墙后面进行连接。如果是这样，请确保确定客户端计算机使用的 IP 地址范围。

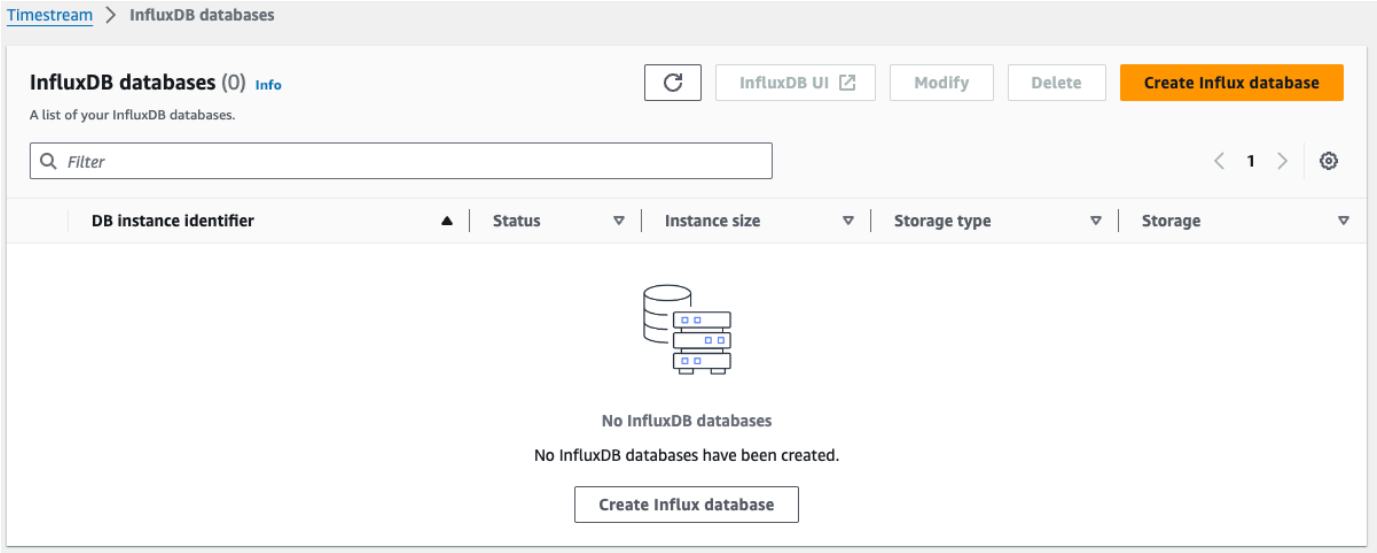
> **Warning**
>
> 我们不建议使用 0.0.0.0/0 进行SSH访问，因为您可以让所有 IP 地址都可以使用访问您的公有实例。EC2 SSH在测试环境中，这种方法在短时间内甚至不可接受，仅授权特定的 IP 地址或地址范围使用访问您的EC2实例SSH。

步骤 2：创建一个 InfluxDB 数据库实例

适用于 InfluxDB 的 Amazon Timestream 的基本构建块是数据库实例。你可以在这个环境中运行 InfluxDB 数据库。

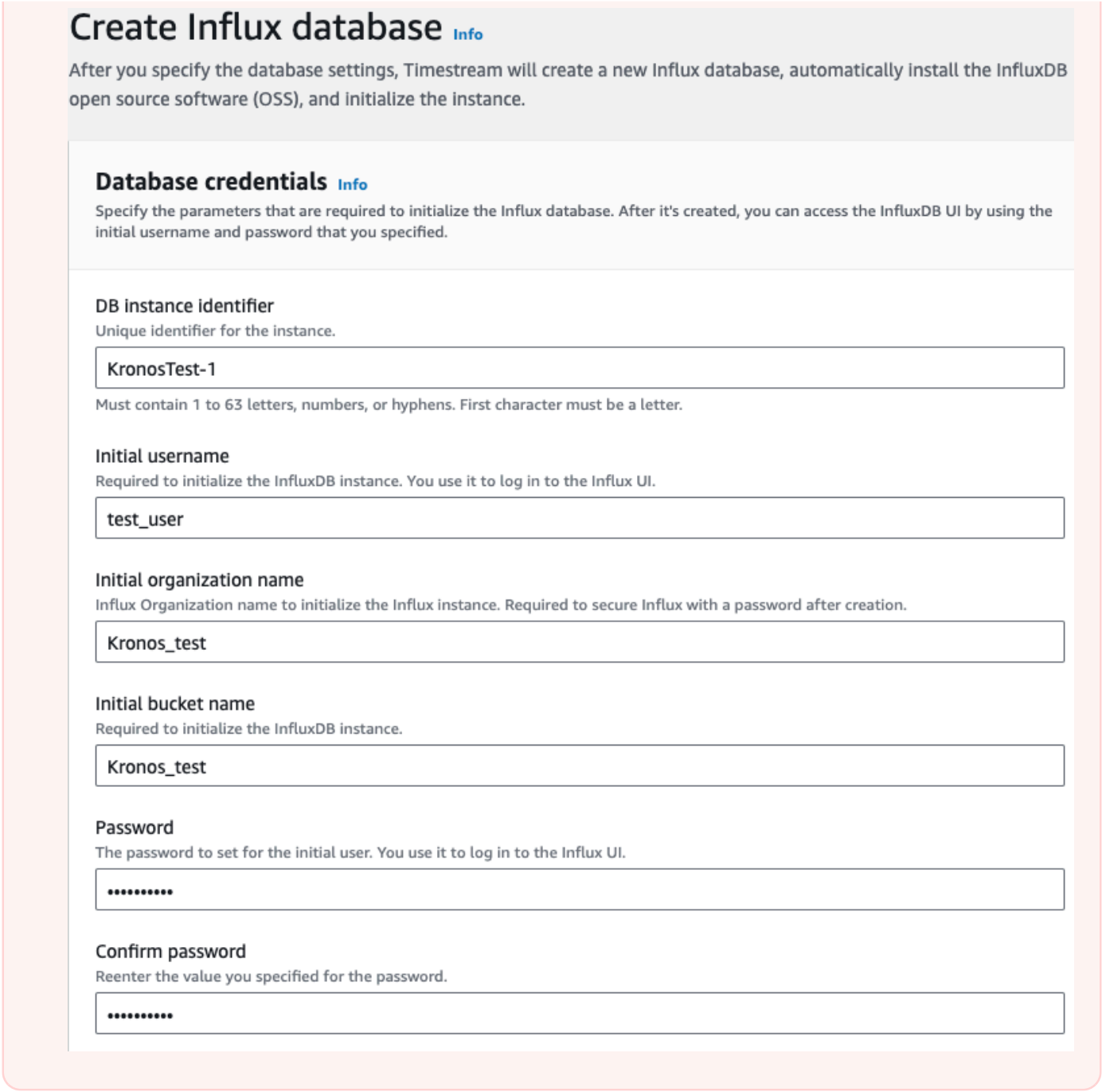
在此示例中，您将使用一个 db.influx.large 数据库实例类创建一个运行 InfluxDB 数据库引擎的数据库实例。

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台，网址为。https://console.aws.amazon.com/timestream/
2. 在适用于 InfluxDB 的 Amazon Timestream 控制台的右上角，选择 AWS 要在其中创建数据库实例的区域。
3. 在导航窗格中，选择 InfluxDB 数据库。
4. 选择创建 Influx 数据库。

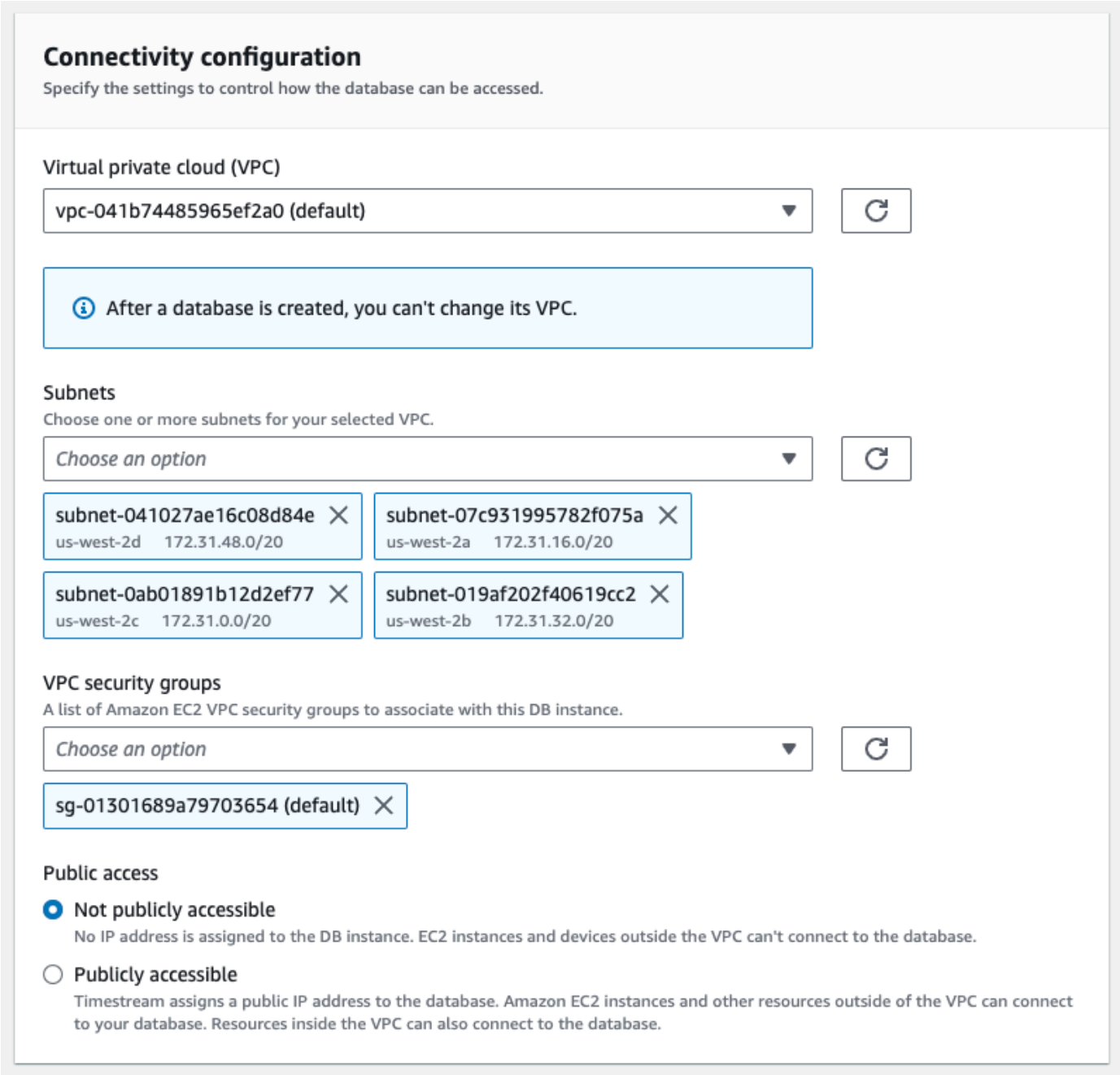


5. 在数据库实例标识符中，输入 KronosTest -1。
6. 提供 InfluxDB 基本配置参数：用户名、组织、存储桶名称和密码。

> **Important**
>
> 您将无法再次查看用户密码。如果没有密码，您将无法访问您的实例和获取操作员令牌。如果您不记录它，您可能需要更改它。请参阅 为您的 InfluxDB 实例创建一个新的操作员令牌。
>
> 如果您需要在数据库实例可用后更改用户密码，则可以修改数据库实例以执行此操作。有关修改数据库实例的更多信息，请参阅 更新数据库实例。



7. 对于数据库实例类，请选择 db.influx. large。
8. 对于数据库存储类别，请选择 infl ux Include IOPS d 3K。
9. 配置您的日志。有关更多信息，请参阅 设置在 Timestream Influxdb 实例上查看 InfluxDB 日志。
10. 在 “连接配置” 部分，确保您的InfluxDB实例与新创建EC2的实例位于同一个子网中。



11. 选择创建 Influx 数据库。
12. 在数据库列表中，选择您的新 InfluxDB 实例的名称以显示其详细信息。数据库实例在准备使用之前的状态为 “正在创建”。

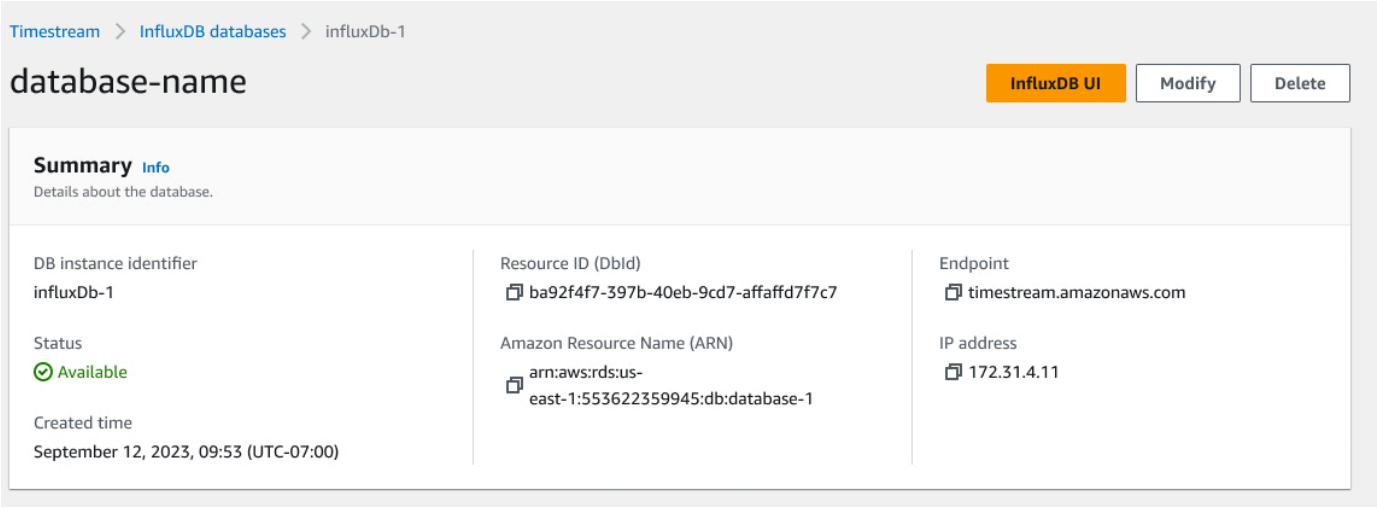
当状态变为 “可用” 时，您可以连接到数据库实例。根据数据库实例类和存储量，新实例可能需要等待 20 分钟时间才可用。

> **Important**
>
> 目前，您无法修改现有实例的计算（实例类型）和存储（存储类型）配置。

第 3 步：将 Telegraf 数据发送到你的 InfluxDB 实例

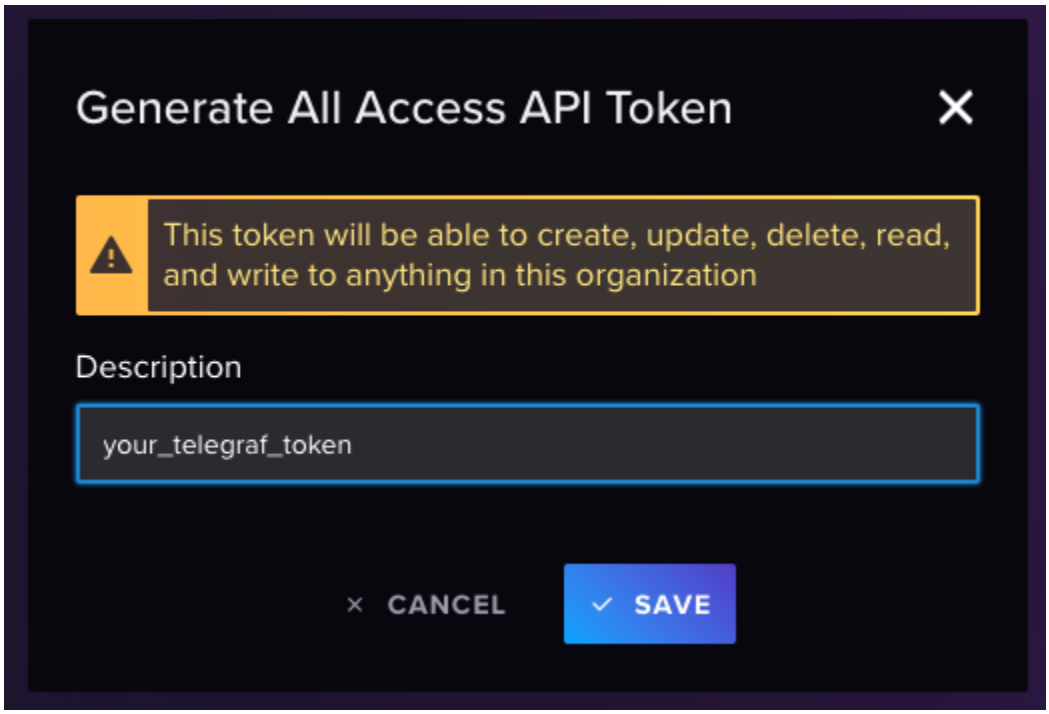
现在，您可以开始使用 Telegraf 代理将遥测数据发送到您的 InfluxDB 数据库实例。在此示例中，您将安装和配置 Telegraf 代理以向您的 InfluxDB 数据库实例发送性能指标。

1. 查找您的数据库实例的终端节点（DNS名称）和端口号。
   a. 登录 AWS 管理控制台并打开亚马逊 Timestream 控制台，网址为。https://console.aws.amazon.com/timestream/
   b. 在 Amazon Timestream 控制台的右上角，选择数据库 AWS 实例的区域。
   c. 在导航窗格中，选择 InfluxDB 数据库。
   d. 选择 InfluxDB 数据库实例名称以显示其详细信息。
   e. 在摘要部分，复制终端节点。另请注意端口号。您需要端点和端口号才能连接到数据库实例（InfluxDB 的默认端口号为 8086）。
2. 接下来，选择 InfluxDB 用户界面。



3. 这将打开一个新的浏览器窗口，您应该会在其中看到登录提示。输入您之前用于创建 InfluxDB 数据库实例的凭证。
4. 在导航窗格中，单击箭头并选择API代币。
5. 在此测试中，生成一个所有访问令牌。

> **Note**
>
> 对于生产场景，我们建议创建具有特定访问权限的令牌，这些存储桶专为Telegraf的特定需求而构建。



6. 您的代币将出现在屏幕上。

> **Important**
>
> 请务必复制并保存令牌，因为您将无法再次显示它。

7. 按照亚马逊 Linux EC2 实例EC2用户指南中连接你的 Linux 实例中的步骤连接到你的 Linux 实例，连接到你之前创建的实例。

   我们建议您使用连接到您的EC2实例SSH。如果SSH客户端实用程序安装在 Windows、Linux 或 Mac 上，则可以使用以下命令格式连接到实例：

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

例如，假设ec2-database-connect-key-pair.pem它存储在 Linux /dir1 上，而您的EC2实例IPv4DNS的公共存储在 Linux 上ec2-12-345-678-90.compute-1.amazonaws.com。你的SSH命令将如下所示：

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

8. 在您的实例上安装最新版本的 telegraf。为此，请使用以下命令：

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdata.repo
[influxdata]
name = InfluxData Repository - Stable
baseurl = https://repos.influxdata.com/stable/\$basearch/main
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdata-archive_compat.key
EOF

sudo yum install telegraf
```

9. 配置你的 Telegraf 实例。

> **Note**
>
> 如果 telegraf.conf 不存在或不包含timestream章节，则可以使用以下命令生成一个章节：
>
> ```
> telegraf –section-filter agent:inputs:outputs –input-filter cpu:mem –output-
> filter timestream config > telegraf.conf
> ```

a. 编辑配置文件，通常位于/etc/telegraf。

```
sudo nano /etc/telegraf/telegraf.conf
```

b. 为CPU、MEM和配置基本输入DISK。

```
[[inputs.cpu]]
  percpu = true
  totalcpu = true
```

```
  collect_cpu_time = false
  report_active = false

[[inputs.mem]]

[[inputs.disk]]
  ignore_fs = ["tmpfs", "devtmpfs", "devfs"]
```

c. 配置输出插件以将数据发送到您的 InfluxDB 数据库实例并保存您的更改。

```
[[outputs.influxdb_v2]]
   urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
   token = "<your_telegraf_token"
   organization = "your_org"
   bucket = "your_bucket"
   timeout = "5s"
```

d. 配置时间流目标。

```
# Configuration for sending metrics to Amazon Timestream.
[[outputs.timestream]]

  ## Amazon Region and credentials
  region = "us-east-1"
  access_key = "<AWS key here>"
  secret_key = "<AWS secret key here>"
  database_name = "<timestream database name>" # needs to exist

  ## Specifies if the plugin should describe t start.
  describe_database_on_start = false
  mapping_mode = "multi-table" # allows multible tables for each input metrics

  create_table_if_not_exists = true
  create_table_magnetic_store_retention_period_in_days = 365
  create_table_memory_store_retention_period_in_hours = 24

  use_multi_measure_records = true # Important to use multi-measure records
  measure_name_for_multi_measure_records = "telegraf_measure"
  max_write_go_routines = 25
```

10. 启用并启动 Telegraf 服务。

```
$ sudo systemctl enable telegraf
```

```
$ sudo systemctl start telegraf
```

第 4 步：删除亚马逊EC2实例和 InfluxDB 数据库实例

使用带有 InfluxUI 的 InfluxDB 数据库实例浏览 Telegraf 生成的数据后，删除您EC2和您的 InfluxDB 数据库实例，这样您就不再需要为它们付费。

要删除EC2实例，请执行以下操作：

1. 登录 AWS Management Console 并打开 Amazon EC2 控制台，网址为https://console.aws.amazon.com/ec2/。
2. 在导航窗格中，选择实例。
3. 选择EC2实例，选择实例状态，然后终止实例。
4. 当系统提示您确认时，选择终止。

有关删除EC2实例的更多信息，请参阅 Amazon EC2 用户指南中的终止您的实例。

要删除没有最终数据库快照的数据库实例，请执行以下操作：

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台，网址为。https://console.aws.amazon.com/timestream/
2. 在导航窗格中，选择 InfluxDB 数据库。
3. 选择要删除的数据库实例。
4. 对于 Actions，选择 Delete。
5. 完成确认并选择删除。

（可选）使用亚马逊托管 Grafana 连接到您的数据库实例

您可以使用亚马逊托管 Grafana 创建控制面板并使用适用于 InfluxDB 的 Amazon Timestream 监控EC2实例的性能。Amazon Managed Grafana 是 Grafana 的完全托管服务，Grafana 是一个流行的开源分析平台，可让您查询、可视化您的指标、日志和跟踪并发出警报。

为您的 InfluxDB 实例创建一个新的操作员令牌

如果您需要为新的InfluxDB实例获取操作员令牌，请执行以下步骤：

1. 要更改您的操作员令牌，我们建议使用Influx CLI。有关说明，请参阅：安装和使用 influx CLI。

2. 将您的配置CLI为使用--username-password以便能够创建运算符：

```
influx config create --config-name CONFIG_NAME1  --host-url "https://
yourinstanceid.eu-central-1.timestream-influxdb.amazonaws.com:8086" --org [YOURORG]
  --username-password [YOURUSERNAME] --active
```

3. 创建新的操作员令牌。系统将要求您输入密码以确认此步骤。

```
influx auth create --org [YOURORG] --operator
```

> **Important**
>
> 创建新的操作员令牌后，您将需要更新当前正在使用旧操作员令牌的所有客户端。

将数据从自我管理的 InfluxDB 迁移到 InfluxDB 的 Timestream

Influx 迁移脚本是一个 Python 脚本，用于在 InfluxDB OSS 实例之间迁移数据，无论这些实例是否由管理。 AWS

InfluxDB 是一个时间序列数据库。InfluxDB 包含点，其中包含许多键值对和一个时间戳。当点按键值对分组时，它们会形成一个序列。系列按称为测量值的字符串标识符进行分组。InfluxDB 通常用于操作监控、IOT数据和分析。存储桶是InfluxDB中一种用于存储数据的容器。 AWS-托管的 InfluxDB 是生态系统中的 InfluxDB。 AWS InfluxDB 提供了 InfluxDB v2API，用于访问数据和更改数据库。InfluxDB v2 API 是 Influx 迁移脚本用来迁移数据的工具。

- Influx 迁移脚本可以迁移存储桶及其元数据，迁移来自所有组织的所有存储桶，或者进行完全迁移，从而替换目标实例上的所有数据。
- 无论在哪个系统执行脚本，该脚本都会在本地备份源实例中的数据，然后将数据恢复到目标实例。数据保存在代码>influxdb-backup-目录中，每次迁移都有一个<timestamp></timestamp>目录。
- 该脚本提供了许多选项和配置，包括安装 S3 存储桶以限制迁移期间的本地存储使用量，以及选择在迁移期间使用哪些组织。

主题

- 准备
- 如何使用脚本

- 迁移概述

准备

InfluxDB 的数据迁移是通过利用 InfluxDB CLI 功能和 InfluxDB v2 的 Python 脚本完成的。API执行迁移脚本需要以下环境配置：

- 支持的版本：支持的最低版本为 2.3 的 InfluxDB 和 Influx CLI。
- 代币环境变量
  - 创建`INFLUX_SRC_TOKEN`包含源InfluxDB实例令牌的环境变量。
  - 创建`INFLUX_DEST_TOKEN`包含目标 InfluxDB 实例令牌的环境变量。
- Python 3
  - 检查安装:`python3 --version`.
  - 如果未安装，请从 Python 网站进行安装。需要最低版本 3.7。在 Windows 上，默认的 Python 3 别名只是 python。
  - Python 模块请求是必需的。使用以下命令进行安装：`shell python3 -m pip install requests`
  - TThe需要使用 Python 模块 influxdb_client。使用以下命令进行安装：`shell python3 -m pip install influxdb_client`
- InfluxDB CLI
  - 确认安装:`influx version`.
  - 如果未安装，请按照 InfluxDB 文档中的安装指南进行操作。

    将涌入添加到您的 $ PATH 中。
- S3 安装工具（可选）

  使用 S3 挂载时，所有备份文件都存储在用户定义的 S3 存储桶中。S3 挂载对于节省执行计算机上的空间或在需要共享备份文件时非常有用。如果未使用 S3 挂载，则省略该`--s3-bucket`选项，则将创建一个本地`influxdb-backup-<millisecond timestamp>`目录，将备份文件存储在运行脚本的同一目录中。

  对于 Linux：挂载点 s3。

  对于 Windows：rclone（需要事先配置 rclone）。
- 磁盘空间

- 迁移过程会自动创建唯一的目录来存储备份文件集，并将这些备份目录保留在 S3 或本地文件系统中，具体取决于提供的程序参数。
- 确保有足够的磁盘空间用于数据库备份，如果您选择省略该--s3-bucket选项并使用本地存储进行备份和恢复，则最好是现有InfluxDB数据库的大小的两倍。
- 在 Windows 上使用df -h (UNIX/Linux)或通过检查驱动器属性来检查空间。

- 直接连接

  确保运行迁移脚本的系统与源系统和目标系统之间存在直接的网络连接。 influx ping --host <host>是验证直接连接的一种方法。

如何使用脚本

运行脚本的一个简单示例是命令：

```
python3 influx_migration.py --src-host <source host> --src-bucket <source bucket> --
dest-host <destination host>
```

它会迁移单个存储桶。

可以通过运行以下命令查看所有选项：

```
python3 influx_migration.py -h
```

用法

```
shell   influx_migration.py [-h] [--src-bucket SRC_BUCKET] [--dest-bucket DEST_BUCKET]
 [--src-host SRC_HOST] --dest-host DEST_HOST [--full] [--confirm-full] [--src-org
 SRC_ORG] [--dest-org DEST_ORG] [--csv] [--retry-restore-dir RETRY_RESTORE_DIR] [--dir-
name DIR_NAME] [--log-level LOG_LEVEL] [--skip-verify] [--s3-bucket S3_BUCKET]
```

Options

- -confirm-full（可选）：使用--full不带--csv会将目标数据库中的所有令牌、用户、存储桶、仪表板和任何其他键值数据替换为源数据库中的令牌、用户、存储桶、仪表板和任何其他键值数据。--fullwit --csv h 仅迁移所有存储桶和存储桶元数据，包括存储桶组织。此选项 (--confirm-full) 将确认完全迁移，无需用户输入即可继续迁移。如果未提供此选项，且--full已提供--csv但未提供，则脚本将暂停执行并等待用户确认。这是一项关键操作，请谨慎行事。默认值为 false。

- -csv（可选）：是否使用 csv 文件进行备份和恢复。如果也通过，`--full`则所有组织中所有用户定义的存储桶都将被迁移，而不是系统存储桶、用户、令牌或仪表板。如果目标服务器中的所有存储桶都需要一个单一的组织，而不是其已经存在的源组织，请使用。`--dest-org`
- -dest-bucket DEST _ BUCKET（可选）：目标服务器中 InfluxDB 存储桶的名称不得是已经存在的存储桶。None如果`--src-bucket`未提供，则默认为`--src-bucket`或的值。
- -dest-host DEST _ HOST：目标服务器的主机。示例：http://localhost:8086。
- -dest-org DEST _ ORG（可选）：目标服务器中要将存储桶还原到的组织名称。如果省略此选项，则源服务器上所有迁移的存储分区都将保留其原始组织，并且如果不创建和切换组织，则迁移的存储分区可能无法在目标服务器中看到。此值将用于所有形式的恢复，无论是单个存储桶、完整迁移还是任何使用 csv 文件进行备份和还原的迁移。
- -dir-name DIR _ NAME（可选）：要创建的备份目录的名称。默认值为 `influxdb-backup-<timestamp>`。一定不存在。
- -full（可选）：是否执行完全恢复，将目标服务器上的所有数据替换为来自所有组织的源服务器的所有数据，包括所有键值数据，例如令牌、仪表板、用户等。覆盖`--src-bucket`和。`--dest-bucket`如果与一起使用`--csv`，则仅迁移存储桶的数据和元数据。默认值为 false。
- h，--help：显示帮助消息并退出。
- -loglevel LOG _ LEVEL（可选）：执行期间要使用的日志级别。选项包括调试、错误和信息。默认为信息。
- -retry-restore-dir RETRY _ RESTORE _ DIR（可选）：上一次还原失败时用于还原的目录，将跳过备份和目录创建，如果该目录不存在将失败，可以是 S3 存储桶中的一个目录。如果恢复失败，则可以用于恢复的备份目录路径将显示为相对于当前目录的路径。S3 存储桶将采用表单`influxdb-backups/<s3 bucket>/<backup directory>`。默认备份目录名称为`influxdb-backup-<timestamp>`。
- -s3-bucket S3_ BUCKET（可选）：用于存储备份文件的 S3 存储桶的名称。在 Linux 上，这只是 S3 存储桶的名称`my-bucket`，例如，给定`AWS_ACCESS_KEY_ID`且`AWS_SECRET_ACCESS_KEY`环境变量已设置或`${HOME}/.aws/credentials`存在。在 Windows 上，这是`rclone`配置的远程和存储桶名称，例如`my-remote:my-bucket`。迁移后，所有备份文件都将保留在 S3 存储桶中的已创建`influxdb-backups-<timestamp>`目录中。`influx-backups`将在运行此脚本的目录中创建一个名为的临时装载目录。如果未提供，则所有备份文件都将存储在本地运行此脚本的创建`influxdb-backups-<timestamp>`目录中。
- -skip-verify（可选）：跳过TLS证书验证。
- -src-bucket SRC _ BUCKET（可选）：源服务器中 InfluxDB 存储桶的名称。如果未提供，则`--full`必须提供。
- -src-host SRC _ HOST（可选）：源服务器的主机。默认为 http://localhost:8086。

如前所述，如果--s3-bucket要使用，rclone则需要使用mountpoint-s3和，但如果用户未提供值，则可以忽略该值--s3-bucket，在这种情况下，备份文件将存储在本地唯一的目录中。

迁移概述

满足先决条件后：

1. 运行迁移脚本：使用您选择的终端应用程序，运行 Python 脚本将数据从源 InfluxDB 实例传输到目标 InfluxDB 实例。
2. 提供凭证：提供主机地址和端口作为CLI选项。
3. 验证数据：通过以下方式确保数据正确传输：
   a. 使用 InfluxDB 用户界面并检查存储桶。
   b. 使用. 列出存储桶。`influx bucket list -t <destination token> --host <destination host address> --skip-verify`
   c. 正在使用`influx v1 shell -t <destination token> --host <destination host address> --skip-verify`并运行`SELECT * FROM <migrated bucket>.<retention period>.<measurement name> LIMIT 100 to view contents of a bucket or SELECT COUNT(*) FROM <migrated bucket>.<retention period>.<measurment name>`来验证是否已迁移了正确数量的记录。

Example 运行示例

1. 打开您选择的终端应用程序，并确保已正确安装所需的先决条件：

```
~ > python3 --version
Python 3.11.5
~ > influx version
Influx CLI 2.7.3 (git: 8b962c7e75) build_date: 2023-04-28T14:22:49Z
~ > s3fs --version
Amazon Simple Storage Service File System V1.92 (commit:unknown) with GnuTLS(gcrypt)
Copyright (C) 2010 Randy Rizun <rrizun@gmail.com>
License GPL2: GNU GPL version 2 <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
~ >
```

2. 导航到迁移脚本：

```
~ > cd sample-code/influxdb-sample/migration/influxdb
~/sample-code/influxdb-sample/migration/influxdb > ls *.py
influx_migration.py
~/sample-code/influxdb-sample/migration/influxdb >
```

3. 准备以下信息：

   a. 要迁移的源存储桶的名称。

   b. （可选）为目标服务器中迁移的存储桶选择一个新的存储桶名称。

   c. 源和目标涌入实例的根令牌。

   d. 源和目标涌入实例的主机地址。

   e. （可选）S3 存储桶名称和凭 AWS Command Line Interface 证；应在操作系统环境变量中设置凭证。

```
# AWS credentials (for timestream testing)
    export AWS_ACCESS_KEY_ID="xxx"
    export AWS_SECRET_ACCESS_KEY="xxx"
```

   f. 按如下方式构造命令：

```
python3 influx_migration.py --src-bucket [source-bucket-name]  --dest-bucket
 [dest-bucket-name] --src-host [source host] --dest-host [dest host] --s3-
bucket [s3 bucket name](optional) --log-level debug
```

   g. 执行脚本：

```
~/sample-code/influxdb-sample/migration/influxdb > python3 influx_migration.py --src-bucket primary-bucket --src-host $INFLUXDB_1_HOST --dest-host $KRO
NOS_HOST --dest-bucket new-bucket-name
```

   h. 等待脚本完成执行。

   i. 检查新迁移的存储桶的数据完整性，`performance.txt`。该文件位于运行脚本的同一目录下，包含有关每个步骤花费了多长时间的一些基本信息。

迁移场景

Example 示例 1：使用本地存储进行简单迁移

您想将单个存储桶（主存储桶）从源服务器迁移`(http://localhost:8086)`到目标服务器。`(http://dest-server-address:8086)`

在确保你能TCP访问（HTTP访问）两台托管在端口 8086 上的 InfluxDB 实例的计算机之后，你同时拥有源令牌和目标令牌，并将它们分别存储为环境变量 INFLUX_SRC_TOKENINFLUX_DEST_TOKEN，为了增加安全性：

```
python3 influx_migration.py --src-bucket primary-bucket --src-host http://
localhost:8086 --dest-host http://dest-server-address:8086
```

该输出值应该类似于以下内容：

```
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB CLI
2023/10/26 10:47:15 INFO: Downloading metadata snapshot
2023/10/26 10:47:15 INFO: Backing up TSM for shard 1
2023/10/26 10:47:15 INFO: Backing up TSM for shard 8245
2023/10/26 10:47:15 INFO: Backing up TSM for shard 8263
[More shard backups . . .]
2023/10/26 10:47:20 INFO: Backing up TSM for shard 8240
2023/10/26 10:47:20 INFO: Backing up TSM for shard 8268
2023/10/26 10:47:20 INFO: Backing up TSM for shard 2
INFO: influx_migration.py: Restoring bucket data and metadata using the InfluxDB CLI
2023/10/26 10:47:20 INFO: Restoring bucket "96c11c8876b3c016" as "primary-bucket"
2023/10/26 10:47:21 INFO: Restoring TSM snapshot for shard 12772
2023/10/26 10:47:22 INFO: Restoring TSM snapshot for shard 12773
[More shard restores . . .]
2023/10/26 10:47:28 INFO: Restoring TSM snapshot for shard 12825
2023/10/26 10:47:28 INFO: Restoring TSM snapshot for shard 12826
INFO: influx_migration.py: Migration complete
```

该目录influxdb-backup-<timestamp>将被创建并存储在运行脚本的目录中，其中包含备份文件。

Example 示例 2：使用本地存储和调试日志进行完全迁移

与上述相同，唯一的不同是您要迁移所有存储桶、令牌、用户和仪表板，删除目标服务器中的存储桶，以及使用选项在用户确认数据库迁移已完成的情况下继续操作。--confirm-full您还想查看性能衡量标准，以便启用调试日志记录。

```
python3 influx_migration.py --full --confirm-full --src-host http://localhost:8086 --
dest-host http://dest-server-address:8086 --log-level debug
```

该输出值应该类似于以下内容：

```
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB CLI
2023/10/26 10:55:27 INFO: Downloading metadata snapshot
2023/10/26 10:55:27 INFO: Backing up TSM for shard 6952
2023/10/26 10:55:27 INFO: Backing up TSM for shard 6953
[More shard backups . . .]
2023/10/26 10:55:36 INFO: Backing up TSM for shard 8268
2023/10/26 10:55:36 INFO: Backing up TSM for shard 2
DEBUG: influx_migration.py: backup started at 2023-10-26 10:55:27 and took 9.41 seconds
 to run.
INFO: influx_migration.py: Restoring bucket data and metadata using the InfluxDB CLI
2023/10/26 10:55:36 INFO: Restoring KV snapshot
2023/10/26 10:55:38 WARN: Restoring KV snapshot overwrote the operator token, ensure
 following commands use the correct token
2023/10/26 10:55:38 INFO: Restoring SQL snapshot
2023/10/26 10:55:39 INFO: Restoring TSM snapshot for shard 6952
2023/10/26 10:55:39 INFO: Restoring TSM snapshot for shard 6953
[More shard restores . . .]
2023/10/26 10:55:49 INFO: Restoring TSM snapshot for shard 8268
2023/10/26 10:55:49 INFO: Restoring TSM snapshot for shard 2
DEBUG: influx_migration.py: restore started at 2023-10-26 10:55:36 and took 13.51
 seconds to run.
INFO: influx_migration.py: Migration complete
```

Example 示例 3：使用CSV、目标组织和 S3 存储桶进行完全迁移

与前面的示例相同，但使用 Linux 或 Mac 并将文件存储在 S3 存储桶中`my-s3-bucket`。这样可以避免备份文件使本地存储容量过载。

```
python3 influx_migration.py --full --src-host http://localhost:8086 --dest-host http://
dest-server-address:8086 --csv --dest-org MyOrg --s3-bucket my-s3-bucket
```

该输出值应该类似于以下内容：

```
INFO: influx_migration.py: Creating directory influxdb-backups
INFO: influx_migration.py: Mounting influxdb-migration-bucket
INFO: influx_migration.py: Creating directory influxdb-backups/my-s3-bucket/influxdb-
backup-1698352128323
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB v2
 API
INFO: influx_migration.py: Restoring bucket data and metadata from csv
INFO: influx_migration.py: Restoring bucket some-bucket
```

```
INFO: influx_migration.py: Restoring bucket another-bucket
INFO: influx_migration.py: Restoring bucket primary-bucket
INFO: influx_migration.py: Migration complete
INFO: influx_migration.py: Unmounting influxdb-backups
INFO: influx_migration.py: Removing temporary mount directory
```

配置数据库实例

本节介绍如何为 InfluxDB 数据库实例设置亚马逊 Timestream。在创建数据库实例之前，请确定将运行该数据库实例的数据库实例类。此外，通过选择 AWS 区域来决定数据库实例的运行位置。接下来，创建数据库实例。

您可以使用数据库参数组配置数据库实例。数据库参数组充当应用于一个或多个数据库实例的引擎配置值的容器。

可用的参数取决于数据库引擎和数据库引擎版本。您可以在创建数据库实例时指定数据库参数组。也可以修改数据库实例来指定它们。

> **Important**
>
> 目前，您无法修改现有实例的计算（实例类型）和存储（存储类型）配置。

创建数据库实例

使用控制台

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream。
2. 在适用于 InfluxDB 的 Amazon Timestream 控制台的右上角，选择 AWS 要在其中创建数据库实例的区域。
3. 在导航窗格中，选择 InfluxDB 数据库。
4. 选择创建 Influx 数据库。
5. 在数据库实例标识符中，输入用于标识您的实例的名称。
6. 提供 InfluxDB 基本配置参数用户名、组织、存储桶名称和密码。

> **Important**
>
> 您的用户名、组织、存储桶名称和密码将作为密钥存储在将为您的账户创建的 S AWS ecrets Manager 中。

如果您需要在数据库实例可用后更改用户密码，则可以使用 Influx CLI 进行修改。

7.

8. 对于数据库实例类别，请选择更适合您的工作负载需求的实例大小。
9. 对于数据库存储类别，请选择适合您需求的存储类别。在所有情况下，您只需要配置分配的存储空间即可。
10. 在连接配置部分中，确保您的InfluxDB实例与需要连接到InfluxDB数据库实例Timestream的新客户端位于同一个子网中。您也可以选择公开您的数据库实例。
11. 选择创建 Influx 数据库。
12. 在数据库列表中，选择您的新 InfluxDB 实例的名称以显示其详细信息。在准备使用之前，数据库实例的状态为 “正在创建”。
13. 当状态变为 Available（可用）时，您便可以连接到该数据库实例。根据数据库实例类和存储量，新实例可能需要等待 20 分钟时间才可用。

使用 CLI

要使用创建数据库实例 AWS Command Line Interface，请使用以下参数调用`create-db-instance`命令：

```
--name
--vpc-subnet-ids
--vpc-security-group-ids
--db-instance-type
--db-storage-type
--username
--organization
--password
--allocated-storage
```

有关每项设置的信息，请参阅 数据库实例的设置。

Example 示例：使用默认引擎配置

对于 Linux、macOS 或 Unix：

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

对于 Windows：

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

使用 API

要使用创建数据库实例 AWS Command Line Interface，请使用以下参数调用`CreateDBInstance`命令：

有关每项设置的信息，请参阅 数据库实例的设置。

> **Important**
>
> 您收到的DBInstance响应对象的一部分 influxAuthParametersSecretArn。这将在您的帐户中 SecretsManager 保密。ARN只有在您的 InfluxDB 数据库实例可用后才会填充它。该密钥包含在此过程中提供的 influx 身份验证参数。`CreateDbInstance`这是一份READONLY副本，因为此密钥的任何updates/modifications/deletions副本都不会影响创建的数据库实例。如果您删除此密钥，我们的API回复仍将提及已删除的密钥ARN。

为InfluxDB数据库实例创建完Timestream后，我们建议您下载、安装和配置Influx。CLI

Influx CLI 提供了一种通过命令行与 InfluxDB 进行交互的简单方法。有关详细的安装和设置说明，请参阅使用 Influx CLI。

数据库实例的设置

您可以使用控制台、`create-db-instance`CLI命令或 Influx API DB 的 `CreateDBInstance` Timestream 操作创建数据库实例。

下表提供了有关您在创建数据库实例时选择的设置的详细信息。

控制台设置	描述	CLI选项和时间流参数 API
分配的存储空间	要为数据库实例分配的存储量（以 GiB 为单位）。有时，为数据库实例分配的存储空间高于数据库大小时可提高 I/O 性能。 有关更多信息，请参阅 InfluxDB 实例存储。	CLI: `allocated-storage` API: `allocated storage`
存储桶名称	用于初始化 InfluxDb 实例的存储桶的名称	CLI: `bucket` API: `bucket`
数据库实例类型	数据库实例的配置。例如，一个 db.influx.large 数据库实例类具有 16 GiB 的内存，2 GiB 内存经过了内存优化。vCPUs 如果可能，请选择一个足够大的数据库实例类型，以便在内存中保存典型的查询工作集。如果在内存中保留工作集，系统可以避免写入到磁盘，从而提高性能。有关更多信息，请参阅 数据库实例类类型。	CLI: `db-instance-type` API: `Dbinstancetype`
数据库实例标识符	数据库实例的名称。请使用与命名本地服务器相同的方式命名数据库实例。您的数据库实例标识符最多可包含 63 个字母数字字符，并且在您选择的 AWS 区域中对于您的账户必须是唯一的。	CLI: `db-instance-identifier` API: `Dbinstanceidentifier`

控制台设置	描述	CLI选项和时间流参数 API
数据库参数组	数据库实例的参数组。您可以选择原定设置参数组，也可以创建自定义参数组。 有关更多信息，请参阅使用数据库参数组。	CLI: `db-parameter-group-name` API: `DBParameterGroupName`
日志传送设置	用于存储 InfluxDB 日志的 S3 存储桶的名称。	CLI: `LogDeliveryConfiguration` API: `log-delivery-configuration`
多可用区部署	创建备用实例，以在另一个可用区中创建数据库实例的被动辅助副本，从而提供故障转移支持。建议将多可用区用于生产工作负载以保持高可用性。 对于开发和测试，您可以选择不创建备用实例。 有关更多信息，请参阅 配置和管理多可用区部署。	CLI: `MultiAz` API: `multi-az`
密码	这将是您用来初始化InfluxDB数据库实例的主用户密码。您将使用此密码登录 InfluxUI 以获取您的操作员令牌。	CLI: `password` API: `password`

控制台设置	描述	CLI选项和时间流参数 API
公共访问	是，为数据库实例提供公有 IP 地址，这意味着它可以在外部访问VPC。要公开访问，数据库实例还必须位于中的公有子网中VPC。 否，仅允许从内部访问数据库实例VPC。 要从数据库实例外部连接到数据库实例VPC，该数据库实例必须可公开访问。此外，必须使用数据库实例安全组的入站规则授予访问权限。此外，还必须满足其他要求。	CLI: `publicly-accessible` API: `PubliclyAccessible`
存储类型	您的数据库实例的存储类型 您可以根据您的工作负载要求在 3 种不同的类型之间进行选择 Provised influx IOPS 包含的存储： * IOPS 包括 Influx 3000 IOPS * IOPS 包括 Influx 12000 IOPS * INflux IOPS 包括 16000 IOPS 有关更多信息，请参阅 InfluxDB 实例存储。	CLI: `db-storage-type` API: `DbStorageType`
初始用户名	这将是用来初始化您的 InfluxDB 数据库实例的主用户。您将使用此用户名登录InfluxUI以获取您的操作员令牌。	CLI: `username` API: `Username`
子网	要与此数据库实例关联的 vpc 子网。	CLI: `vpc-subnet-ids` API: `VPCSubnetIds`

控制台设置	描述	CLI选项和时间流参数 API
VPC安全组（防火墙）	要与数据库实例关联的安全组。	CLI: `vpc-security-group-ids` API: `VPCSecurityGroupIds`

连接适用于 InfluxDB 数据库实例的 Amazon Timestream

在连接到数据库实例之前，您必须先创建数据库实例。有关信息，请参阅创建数据库实例。在 Amazon Timestream 配置您的数据库实例后，使用 InfluxDB API、Influx CLI 或任何与 InfluxDB 兼容的客户端或实用程序连接到数据库实例。

主题

- 查找适用于 InfluxDB 的 Amazon Timestream 数据库实例的连接信息
- 数据库身份验证选项
- 使用参数组

查找适用于 InfluxDB 的 Amazon Timestream 数据库实例的连接信息

数据库实例的连接信息包括其终端节点、端口、用户名、密码和有效的访问令牌，例如操作员或所有访问令牌。例如，对于 InfluxDB 数据库实例，假设终端节点值为。`influxdb1-123456789.us-east-1.timestream-influxdb.amazonaws.com`在本例中，端口值为 8086，数据库用户为管理员。鉴于此信息，您可以在连接字符串中指定以下值：

- 对于主机、主机名或DNS名称，请指定`influxdb1-123456789.us-east-1.timestream-influxdb.amazonaws.com`。
- 对于端口，请指定 8086。
- 对于用户，请指定管理员。
- 对于密码，请指定您在创建数据库实例时提供的密码。

> **Important**
>
> 当你为 InfluxDB 数据库实例创建 Timestream 时，你会收到DBInstance响应对象的一部分。`influxAuthParametersSecretArn`这将在您的帐户中保留一个 SecretsManager 秘密。只有在您的 InfluxDB 数据库实例可用后才会填充它。该密钥包含在此过程中提供的 influx 身份验证参数。`CreateDbInstance`这是一份READONLY副本，因为此密钥的任何updates/modifications/deletions副本都不会影响创建的数据库实例。如果您删除此密钥，我们的API回复仍将提及已删除的机密 arn。

端点对于每个数据库实例都是唯一的，端口和用户的值可能会有所不同。要连接到数据库实例，你可以使用 Influx、Influx CLI 或任何与 Influx API DB 兼容的客户端。

要查找数据库实例的连接信息，请使用 AWS 管理控制台。你也可以使用 AWS 命令行界面 (AWS CLI) `describe-db-instances` 命令或 timeStream-API `GetDBInstance` InfluxDB 操作。

使用 AWS Management Console

1. 登录 AWS Management Console 并打开亚马逊 Timestream 控制台。
2. 在导航窗格中，选择 InfluxDB 数据库以显示您的数据库实例列表。
3. 选择数据库实例的名称以显示其详细信息。
4. 在摘要部分，复制终端节点。另请注意端口号。您需要端点和端口号才能连接到数据库实例。

如果您需要查找用户名和密码信息，请选择配置详细信息选项卡，然后选择访问您的 Secr `influxAuthParametersSecretArn` ets Manager。

使用 CLI

- 要使用查找 InfluxDB 数据库实例的连接信息 AWS CLI，请调用命令。 get-db-instance在调用中，查询数据库实例 ID、终端节点、端口和 influxAuthParameters密钥。

  对于 Linux、macOS 或 Unix：

  ```
  aws timestream-influxdb get-db-instance --identifier id \
   --query "[name,endpoint,influxAuthParametersSecretArn]"
  ```

  对于 Windows：

```
aws timestream-influxdb get-db-instance --identifier id \
 --query "[name,endpoint,influxAuthParametersSecretArn]"
```

您的输出应类似于以下内容。要访问用户名信息，您需要查看InfluxAuthParameterSecret.

```
[
    [
        "mydb",
        "mydb-123456789012.us-east-1.timestream-influxdb.amazonaws.com",
        8086,
    ]
]
```

创建访问令牌

有了这些信息，您将能够连接您的实例，以检索或创建您的访问令牌。有几种方法可以实现这一点：

使用 CLI

1. 如果你还没有，请下载、安装和配置 in flux CLI。
2. 在配置 Influx CLI 配置时--username-password，使用进行身份验证。

```
influx config create --config-name YOUR_CONFIG_NAME  --host-url "https://
yourinstance.timestream-influxdb.amazonaws.com:8086"  --org yourorg --username-
password admin --active
```

3. 使用 infl ux auth create 命令来重新创建你的操作员令牌。请注意，此过程将使旧的操作员令牌失效。

```
influx auth create --org kronos --operator
```

4. 获得操作员令牌后，你可以使用 infl ux auth lis t 命令来查看你的所有代币。你可以使用 infl ux auth creat e 命令来创建所有访问令牌。

> **Important**
>
> 您需要先执行此步骤以获取操作员令牌，然后才能使用InfluxDB API 或创建新令牌。CLI

使用 Influx 用户界面

1. 使用创建的端点浏览到你的 InfluxDB 实例的时间流，登录并访问 InfluxDB 用户界面。您将需要使用用于创建InfluxDB数据库实例的用户名和密码。您可以从的响应对象`influxAuthParametersSecretArn`中指定的中检索此信息`CreateDbInstance`。

   或者，你可以从 InfluxDB 管理控制台的 Timestream 中打开 InfluxUI：

   a. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台，网址为。https://console.aws.amazon.com/timestream/

   b. 在适用于 InfluxDB 的 Amazon Timestream 控制台的右上角，选择您 AWS 创建数据库实例的区域。

   c. 在数据库列表中，选择InfluxDB实例的名称以显示其详细信息。在右上角，选择 “打开 Influx 用户界面”。

2. 登录到 InfluxUI 后，使用左侧导航栏导航到 “加载数据”，然后导航到 “API令牌”。
3. 选择 + GENERATE API TOKEN 并选择 “所有访问API令牌”。
4. 输入令API牌的描述并选择SAVE。
5. 复制生成的令牌并将其存储起来以便妥善保管。

> **Important**
>
> 从InfluxUI创建代币时，新创建的代币只会显示一次。请务必复制它们，因为如果没有，则需要重新创建它们。

使用 InfluxDB API

- 使用请求方法向 InfluxDB API `/api/v2/authorizations` 端点发送POST请求。

  在您的请求中包括以下内容：

  a. 标题：

     i. 授权：令牌 < INFLUX _ OPERATOR _ TOKEN >

     ii. 内容类型：应用程序/json

  b. 请求正JSON文：具有以下属性的正文：

i. 状态：“活跃”

ii. 描述：API代币描述

iii. OrgID：InfluxDB 组织 ID

iv. 权限：对象数组，其中每个对象代表InfluxDB资源类型或特定资源的权限。每个权限都包含以下属性：

A. 操作：“读取” 或 “写入”

B. 资源：表示要向其授予权限的 InfluxDB 资源的JSON对象。每个资源至少包含以下属性：orgID：InfluxDB 组织 ID

C. 类型：资源类型。有关存在哪些InfluxDB资源类型的信息，请使用the /api/v2/resources端点。

以下示例使用curl和 InfluxDB API 生成所有访问令牌：

```
export INFLUX_HOST=https://influxdb1-123456789.us-east-1.timestream-
influxdb.amazonaws.com
export INFLUX_ORG_ID=<YOUR_INFLUXDB_ORG_ID>
export INFLUX_TOKEN=<YOUR_INFLUXDB_OPERATOR_TOKEN>

curl --request POST \
"$INFLUX_HOST/api/v2/authorizations" \
  --header "Authorization: Token $INFLUX_TOKEN" \
  --header "Content-Type: text/plain; charset=utf-8" \
  --data '{
    "status": "active",
    "description": "All access token for get started tutorial",
    "orgID": "'"$INFLUX_ORG_ID"'",
    "permissions": [
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "authorizations"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "authorizations"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "buckets"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "buckets"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "dashboards"}},
```

```
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"dashboards"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "orgs"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "orgs"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"sources"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"sources"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "tasks"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"tasks"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"telegrafs"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"telegrafs"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "users"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"users"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"variables"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"variables"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"scrapers"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"scrapers"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"secrets"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"secrets"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"labels"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"labels"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "views"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"views"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"documents"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"documents"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationRules"}},
```

```
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "notificationRules"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "notificationEndpoints"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "notificationEndpoints"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "checks"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "checks"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "dbrp"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "dbrp"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "notebooks"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "notebooks"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "annotations"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "annotations"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "remotes"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "remotes"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "replications"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "replications"}}
    ]
  }
'
```

数据库身份验证选项

适用于 InfluxDB 的 Amazon Timestream 支持以下方式对数据库用户进行身份验证：

- 密码身份验证 – 数据库实例将执行用户账户的所有管理。你可以使用 InfluxUI、Influx 或 influx CLI 创建用户、指定密码和管理令牌。API
- 令牌身份验证-您的数据库实例执行用户账户的所有管理。您可以使用 Influx 和 Influx 使用操作员令牌创建用户CLI、指定密码和管理令牌。API

加密连接

您可以使用应用程序中的安全套接字层 (SSL) 或传输层安全 (TLS) 来加密与数据库实例的连接。InfluxDB与Kronos服务创建和管理的应用程序之间进行TLS握手所需的证书。续订证书后，实例将自动更新为最新版本，无需任何用户干预。

使用参数组

数据库参数指定数据库的配置方式。例如，数据库参数可以指定要分配给数据库的资源量（如内存）。

您可以通过将数据库实例与参数组关联来管理数据库配置。适用于 InfluxDB 的 Amazon Timestream 使用默认设置定义参数组。您还可以使用自定义设置定义您自己的参数组。

参数组概述

数据库参数组就像是引擎配置值的容器，这些值应用于一个或多个数据库实例。

主题

- 原定设置和自定义参数组
- 创建数据库参数组
- 静态和动态数据库实例参数
- 支持的参数和参数值

原定设置和自定义参数组

数据库实例使用数据库参数组。以下各节介绍配置和管理数据库实例参数组。

创建数据库参数组

您可以使用 AWS Management Console、或 Timestream API 创建新的数据库参数组。 AWS Command Line Interface

以下限制适用于数据库参数组名称：

- 名称必须为 1 到 255 个字母、数字或连字符。
- 原定设置参数组名称可以包含句点，例如 `default.InfluxDB.2.7`。但是，自定义参数组名称不能包含句点。
- 第一个字符必须是字母。

- 名称不能以 “dbpg-” 开头
- 名称不能以连字符结束，也不能包含两个连续的连字符。
- 如果您在未指定数据库参数组的情况下创建数据库实例，则该数据库实例将使用 InfluxDB 引擎默认值。

默认参数组的参数设置无法修改。您可以改而执行以下操作：

1. 创建新参数组。
2. 更改所需参数的设置。并非参数组中的所有数据库引擎参数都可供修改。
3. 更新您的数据库实例以使用自定义参数组。有关更新数据库实例的信息，请参阅更新数据库实例。

> **Note**
>
> 如果您已将数据库实例修改为使用自定义参数组，并且启动了数据库实例，则作为启动过程的一部分，适用于 InfluxDB 的 Amazon Timestream 会自动重启数据库实例。
> 目前，创建自定义参数组后，您将无法对其进行修改。如果您需要更改参数，则需要创建新的自定义参数组并将其分配给需要此配置更改的实例。如果您更新现有数据库实例以分配新的参数组，则该参数组将始终立即应用并重启您的实例。

静态和动态数据库实例参数

InfluxDB 数据库实例参数始终是静态的。它们的行为如下：

当您更改静态参数、保存数据库参数组并将其分配给实例时，参数更改将在实例重启后自动生效。

当您将新的数据库参数组与数据库实例关联时，Timestream 仅在数据库实例重启后才会应用修改后的静态参数。目前唯一的选择是立即申请。

有关更改数据库集群参数组的信息，请参阅 更新数据库实例。

支持的参数和参数值

要确定数据库实例支持的参数，请查看数据库实例使用的数据库参数组中的参数。有关更多信息，请参阅查看数据库参数组的参数值。

有关InfluxDB开源版本支持的所有参数的更多信息，请参阅 In fluxDB配置选项。目前，您只能修改以下 InfluxDB 参数：

参数	描述	默认值	值	有效范围	备注
flux-log-enabled	包括显示 Flux 查询详细日志的选项	FALSE	true，false	不适用	
日志级别	日志输出级别。InfluxDB 输出严重性级别大于或等于指定级别的日志条目。	info	调试、信息、错误	不适用	
没有任务	允许同时执行的查询数。设置为 0 允许无限数量的并发查询。	FALSE	true，false	不适用	
查询并发	禁用任务计划程序。如果有问题的任务阻止 InfluxDB 启动，请使用此选项在不调度或执行任务的情况下启动 InfluxDB。	1024		不适用	
query-queue-size	执行队列中允许的最大查询数。当达到队列限制时，新的查询将被拒绝。如果设置为 0，则队列	1024		不适用	

参数	描述	默认值	值	有效范围	备注
	中的查询数量不受限制。				
追踪类型	在 InfluxDB 中启用跟踪并指定跟踪类型。默认情况下，跟踪处于禁用状态。	""	log，jaeger	不适用	
指标已禁用	禁用暴露内部 Influ xDB HTTP 指标的 /metrics 端点。	FALSE		不适用	

参数	描述	默认值	值	有效范围	备注
http-idle-timeout	服务器在等待新请求时应保持已建立连接的最长持续时间。设置为，0表示没有超时。	3m0s	以单位为单位的持续时间 hoursminutes nds 。例如：duration ype=minut es,value= 10	时间： -最小值：0 -最大值：256205 分钟： -最小值：0 -最大值：15372286 秒： -最小值：0 -最大值：922337203 毫秒： -最小值：0 -最大值：9223 37203685	

参数	描述	默认值	值	有效范围	备注
http-read-header-timeout	服务器应尝试读取新请求HTTP标头的最长持续时间。设置0为不超时。	10s	以单位为单位的持续时间 hoursminutes nds 。例如：duration ype=minutes,value=10	时间： -最小值：0 -最大值：256205 分钟： -最小值：0 -最大值：15372286 秒： -最小值：0 -最大值：922337203 毫秒： -最小值：0 -最大值：9223 37203685	

参数	描述	默认值	值	有效范围	备注
http-read-timeout	服务器应尝试读取全部新请求的最长持续时间。设置0为不超时。	0	以单位为单位的持续时间 hoursminutes nds 。例如：duration type=minutes,value=10	时间： -最小值：0 -最大值：256205 分钟： -最小值：0 -最大值：15372286 秒： -最小值：0 -最大值：922337203 毫秒： -最小值：0 -最大值：9223 37203685	

参数	描述	默认值	值	有效范围	备注
http-write-timeout	服务器在处理和响应写入请求时应花费的最长时间。设置0为不超时。	0	以单位为单位的持续时间 hoursminutes nds 。例如：duration ype=minutes,value=10	时间： -最小值：0 -最大值：256205 分钟： -最小值：0 -最大值：15372286 秒： -最小值：0 -最大值：922337203 毫秒： -最小值：0 -最大值：9223 37203685	
influxql-max-select-buckets	一条SELECT语句可以创建的最大按时间分组的存储桶数。 0允许无限数量的存储桶。	0	长整型	最小值：0 最大值：9,223 ,372,036, 85,854,77 5,807	

参数	描述	默认值	值	有效范围	备注
influxql-max-select-point	一条SELECT语句可以处理的最大点数。0允许无限数量的积分。InfluxDB 每秒检查一次点数（因此超过最大值的查询不会立即中止）。	0	长整型	最小值：0 最大值：9,223,372,036,85,854,775,807	
influxql-max-select-series	一条SELECT语句可以返回的最大序列数。0允许无限数量的系列。	0	长整型	最小值：0 最大值：9,223,372,036,85,854,775,807	
pprof 已禁用	禁用/debug/pprof HTTP终端节点。此端点提供运行时分析数据，在调试时可能会有所帮助。	FALSE	布尔值	不适用	
query-initial-memory-bytes	为查询分配的初始内存字节数。	0	长整型	最小值：0 最大值：query-memory-bytes	

参数	描述	默认值	值	有效范围	备注
query-max-memory-bytes	允许查询的最大内存总字节数。	0	长整型	最小值：0 最大值：9,223,372,036,85,854,775,807	
query-memory-bytes	为新创建的用户会话指定生存时间 (TTL)（以分钟为单位）。	0	长整型	最小值：0 最大值：2,147,483,647	必须大于或等于 query-initial-memory-bytes。
会话时长	为新创建的用户会话指定生存时间 (TTL)（以分钟为单位）。	60	整数	最小值：0 最大值：2880	

参数	描述	默认值	值	有效范围	备注
session-renew-disabled	禁止在每次请求时自动延长用户的TTL会话。默认情况下，每个请求都会将会话的过期时间设置为从现在起的五分钟。禁用后，会话将在指定的会话时长后过期，即使用户最近处于活动状态，也会被重定向到登录页面。	FALSE	布尔值	不适用	
storage-cache-max-memory-尺寸	分片缓存开始拒绝写入之前可以达到的最大大小（以字节为单位）。	1073741824	长整型	最小值：0 最大值：54975 5813888	必须低于实例的总内存容量。 我们建议将其设置为低于总内存容量的15%。
storage-cache-snapshot-memory-尺寸	存储引擎将对缓存进行快照并将其写入TSM文件以提供更多可用内存的大小（以字节为单位）。	26214400	长整型	最小值：0 最大值：54975 5813888	必须小于storage-cache-max-memory-size。

参数	描述	默认值	值	有效范围	备注
storage-cache-snapshot-write-寒冷时长	如果分片未收到写入或删除操作，则存储引擎将对缓存进行快照并将其写入新TSM文件的持续时间。	10m0s	以单位为单位的持续时间 hoursminutes nds 。例如：duration ype=minut es,value= 10	时间： -最小值：0 -最大值：256205 分钟： -最小值：0 -最大值：15372286 秒： -最小值：0 -最大值：922337203 毫秒： -最小值：0 -最大值：9223 37203685	

参数	描述	默认值	值	有效范围	备注
storage-compact-full-write-寒冷时长	存储引擎在未收到写入或删除操作的情况下压缩分片中所有TSM文件的持续时间。	4h0m0s	以单位为单位的持续时间 hoursminutes nds 。例如：duration ype=minut es,value= 10	时间： -最小值：0 -最大值：256205 分钟： -最小值：0 -最大值：15372286 秒： -最小值：0 -最大值：922337203 毫秒： -最小值：0 -最大值：9223 37203685	
storage-compact-throughput-burst	TSM压缩可以写入磁盘的速率限制（以每秒字节为单位）。	50331648	长整型	最小值：0 最大值：9,223 ,372,036, 85,854,77 5,807	

参数	描述	默认值	值	有效范围	备注
storage-max-concurrent-compactions	可以同时运行的完全压实和关卡压实的最大数量。值为会导0致 50% 的runtime.GOMAXPROCS(0) 使用量在运行时使用。任何大于零的数字都将压缩限制为该值。此设置不适用于缓存快照。	0	整数	最小值：0 最大值：64	
storage-max-index-log-文件大小	索引预写日志 () 文件压缩成索引文件的大小（以字节为单位WAL）。较小的大小将导致日志文件的压缩速度更快，并以牺牲写入吞吐量为代价降低堆使用量。	1048576	长整型	最小值：0 最大值：9,223,372,036,854,775,807	
storage-no-validate-field-尺寸	跳过对传入写入请求的字段大小验证。	FALSE	布尔值	不适用	

参数	描述	默认值	值	有效范围	备注
storage-retention-check-interval	保留策略实施检查的时间间隔。	30m0s	以单位为单位的持续时间 hoursminutes nds 。例如：duration ype=minut es,value= 10	不适用	时间： -最小值：0 -最大值：256205 分钟： -最小值：0 -最大值：15372286 秒： -最小值：0 -最大值：922337203 毫秒： -最小值：0 -最大值：9223 37203685
storage-series-file-max-concurrent-snapshot-compactions	可以在数据库中所有系列分区上并行运行的最大快照压缩数。	0	整数	最小值：0 最大值：64	

参数	描述	默认值	值	有效范围	备注
storage-series-id-set-缓存大小	TSI索引中用于存储先前计算的系列结果的内部缓存的大小。当执行具有相同标签键/值谓词的后续查询时，可以快速返回缓存的结果，而不必重新计算。将此值设置为0将禁用缓存并可能降低查询性能。	100	长整型	最小值：0 最大值：9,223,372,036,85,854,775,807	
storage-wal-max-concurrent-写道	同时尝试写入WAL目录的最大次数。	0	整数	最小值：0 最大值：256	

参数	描述	默认值	值	有效范围	备注
storage-wal-max-write-延迟	当该WAL目录的并发活动写入次数达到最大值时，该WAL目录的写入请求将等待的最大时间。设置0为可禁用超时。	10m	以单位为单位的持续时间 hoursminutes nds 。例如：duration ype=minut es,value= 10	时间： -最小值：0 -最大值：256205 分钟： -最小值：0 -最大值：15372286 秒： -最小值：0 -最大值：922337203 毫秒： -最小值：0 -最大值：9223 37203685	
ui 已禁用	禁用 InfluxDB 用户界面 (UI)。默认情况下，用户界面处于启用状态。	FALSE	布尔值	不适用	

在参数组内设置参数不恰当可能会产生意外的不利影响，包括性能降低和系统不稳定。修改数据库参数时请务必谨慎。在将这些参数组更改应用于生产数据库实例之前，请尝试对测试数据库实例进行参数组设置更改。

使用数据库参数组

数据库实例使用数据库参数组。以下各节介绍配置和管理数据库实例参数组。

主题

创建数据库参数组

使用 AWS Management Console

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台。
2. 在导航窗格中，选择参数组。
3. 选择创建参数组。
4. 在组名框中，输入新数据库参数组的名称。
5. 在描述框中，输入新数据库参数组的描述。
6. 选择要修改的参数并应用所需的值。有关支持的参数的更多信息，请参阅支持的参数和参数值。
7. 选择保存。

使用 AWS Command Line Interface

- 要使用创建数据库参数组 AWS CLI，请使用以下参数调用`create-db-parameter-group`命令：

```
--db-parameter-group-name <value>
--description <value>
--endpoint_url <value>
--region <value>
--parameters (list) (string)
```

Example 示例

有关每项设置的信息，请参阅 数据库实例的设置。此示例使用默认引擎配置。

```
aws timestream-influxdb create-db-parameter-group
    --db-parameter-group-name YOUR_PARAM_GROUP_NAME\
    --endpoint-url YOUR_ENDPOINT
    --region YOUR_REGION \
    --parameters
 "InfluxDBv2={logLevel=debug,queryConcurrency=10,metricsDisabled=true}" \" \
    --debug
```

将数据库参数组与数据库实例关联

您可以使用自定义设置创建自己的数据库参数组。您可以使用 AWS Management Console、或 Timestream API-InfluxDB 将数据库参数组与数据库实例关联起来。 AWS Command Line Interface在创建或修改数据库实例时可以执行此操作。

有关创建数据库参数组的信息，请参阅创建数据库参数组。有关创建数据库实例的信息，请参阅创建数据库实例。有关修改数据库实例的信息，请参阅更新数据库实例...

> **Note**
>
> 当您将新的数据库参数组与数据库实例关联时，只有在数据库实例重启后才会应用修改后的静态参数。目前，仅支持立即申请。InfluxDB 的时间流仅支持静态参数。

使用 AWS Management Console

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台。
2. 在导航窗格中，选择 InfluxDB 数据库，然后选择要修改的数据库实例。
3. 选择更新。将出现 “更新数据库实例” 页面。
4. 更改数据库参数组设置。
5. 选择继续，查看修改摘要。
6. 目前仅支持立即申请。在某些情况下，此选项可能会导致中断，因为它会重启您的数据库实例。
7. 在确认页面上，检查您的更改。如果它们正确，请选择更新数据库实例以保存您的更改并应用更改。也可以选择 Back (返回) 编辑您的更改，或选择 Cancel (取消) 取消更改。

使用 AWS Command Line Interface

对于 Linux、macOS 或 Unix：

```
aws timestream-influxdb update-db-instance
--identifier YOUR_DB_INSTANCE_ID \
--region YOUR_REGION \
--db-parameter-group-identifier YOUR_PARAM_GROUP_ID \
--log-delivery-configuration "{\"s3Configuration\": {\"bucketName\":
 \"${LOGGING_BUCKET}\", \"enabled\": false }}"
```

对于 Windows：

```
aws timestream-influxdb update-db-instance
--identifier YOUR_DB_INSTANCE_ID ^
--region YOUR_REGION ^
--db-parameter-group-identifier YOUR_PARAM_GROUP_ID ^
--log-delivery-configuration "{\"s3Configuration\": {\"bucketName\":
 \"${LOGGING_BUCKET}\", \"enabled\": false }}"
```

列出数据库参数组

您可以列出您为 AWS 账户创建的数据库参数组。

使用 AWS Management Console

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台。
2. 在导航窗格中，选择参数组。
3. 数据库参数组将显示在列表中。

使用 AWS Command Line Interface

要列出 AWS 账户的所有数据库参数组，请使用 AWS Command Line Interface `list-db-parameter-groups`命令。

```
aws timestream-influxdb list-db-parameter-groups --region region
```

要返回 AWS 账户的特定数据库参数组，请使用 AWS Command Line Interface `get-db-parameter-group`命令。

```
aws timestream-influxdb get-db-parameter-group --region region --identifier identifier
```

查看数据库参数组的参数值

您可获得数据库参数组内所有参数的列表及它们的值。

使用 AWS Management Console

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台。
2. 在导航窗格中，选择参数组。
3. 数据库参数组将显示在列表中。
4. 选择参数组名称以查看其参数列表。

使用 AWS Command Line Interface

要查看数据库参数组的参数值，请使用带有以下必需参数的 AWS Command Line Interface `get-db-parameters`命令。

```
--db-parameter-group-name
```

使用 API

要查看数据库参数组的参数值，请使用带有以下必需参数的 Timestream API `GetDBParameters` 命令。

```
DBParameterGroupName
```

管理数据库实例

本节涵盖了管理 InfluxDB 实例的 Timestream 的各个方面，以确保最佳性能、可用性和监控功能。它提供了有关更新数据库实例配置、处理多可用区部署和故障转移过程的指导。它还说明了如何删除数据库实例并为InfluxDB实例设置日志查看。

主题

- 更新数据库实例
- 维护数据库实例
- 删除数据库实例

- 多可用区数据库实例部署
- 设置在 Timestream Influxdb 实例上查看 InfluxDB 日志

更新数据库实例

你可以更新 InfluxDB 实例的 Timestream 的以下配置参数：

- 实例类
- Deployment type（部署类型）
- 参数组
- 日志传送配置

> **Important**
>
> 我们建议您在修改生产实例之前，先在测试实例上测试所有更改，以了解其影响，尤其是在升级数据库版本时。在更新设置之前，请查看对数据库和应用程序的影响。某些修改需要重启数据库实例，从而导致停机。

使用 AWS Management Console

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台。
2. 在导航窗格中，选择 InfluxDB 数据库，然后选择要修改的数据库实例。
3. 选择 Modify（修改）。
4. 在修改数据库实例页面上，进行所需的更改。
5. 选择继续，查看修改摘要。
6. 选择下一步。
7. 查看您的更改。
8. 选择修改实例以应用您的更改。

> **Note**
>
> 这些修改需要重启 Influx 数据库实例，在某些情况下可能会导致中断。

使用 AWS Command Line Interface

要使用更新数据库实例 AWS Command Line Interface，请调用`update-db-instance`命令。指定数据库实例标识符以及要修改的选项值。有关各选项的信息，请参阅数据库实例的设置。

Example 示例

以下代码通过设置不同的 dbparametergroup 来修改 mydbinstance。更改将立即应用。

对于 Linux、macOS 或 Unix：

```
aws timestream-influxdb update-db-instance \
    --identifier mydbinstance \
    --db-instance-type desired-instance-type \
    --deployment-type desired-deployment-type \
    --db-parameter-group-name newparamgroup \
    --port 8086
```

对于 Windows：

```
aws timestream-influxdb update-db-instance ^
    --identifier mydbinstance ^
    --db-instance-type desired-instance-type ^
    --deployment-type desired-deployment-type ^
    --db-parameter-group-name newparamgroup
    --port 8086
```

维护数据库实例

适用于 InfluxDB 的亚马逊 Timestream 会定期在亚马逊 Timestream 上对 InfluxDB 资源进行维护。维护通常涉及更新数据库实例中的以下资源：

- 底层硬件
- 底层操作系统（OS）
- 数据库引擎版本

针对操作系统的更新最常见的原因是安全问题。

某些维护项目要求适用于 InfluxDB 的 Amazon Timestream 让您的数据库实例在短时间内离线。要求资源脱机的维护项目包括必需的操作系统或数据库修补。仅对与安全性和实例可靠性相关的修补程序自

动安排必需的修补。此类补丁很少发生，通常每隔几个月发生一次。它所需要的维护时间很少超过维护窗口的一小部分。

- 您的实例所在地区的维护时段配置为每天当地时间上午 12 点至凌晨 4 点之间进行。
- 在一周的七个维护窗口中的任何一个窗口中，可以每周对客户资源进行一次修补。

删除数据库实例

删除数据库实例会影响实例的可恢复性和快照可用性。请考虑以下问题：

- 如果您想删除 InfluxDB 资源的所有 Timestream，请注意，数据库实例资源会产生账单费用。
- 当数据库实例的状态为删除时，其 CA 证书值不会显示在 InfluxDB 的 Timestream 控制台中，也不会出现在 AWS Command Line Interface 命令或 Timestream 操作的输出中。API
- 删除数据库实例所需的时间因删除的数据量以及是否拍摄最终快照而异。

您可以使用 AWS Management Console、或 Timestream API 删除数据库实例。 AWS Command Line Interface您必须提供数据库实例的名称：

使用 AWS Management Console

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台。
2. 在导航窗格中，选择 InfluxDB 数据库，然后选择要删除的数据库实例。
3. 选择删除。
4. 在框中输入 “确认”。
5. 选择删除。

使用 AWS Command Line Interface

要在您的账户中查找数据库实例IDs的实例，请调用以下`list-db-instances`命令：

```
aws timestream-influxdb list-db-instances \
--endpoint-url YOUR_ENDPOINT \
--region YOUR_REGION
```

要使用删除数据库实例 AWSCLI，请使用以下选项调用`delete-db-instance`命令：

```
aws timestream-influxdb list-db-instances \
--identifier YOUR_DB_INSTANCE \
```

Example 示例

对于 Linux、macOS 或 Unix：

```
aws timestream-influxdb delete-db-instance \
    --identifier mydbinstance
```

对于 Windows：

```
aws timestream-influxdb delete-db-instance ^
    --identifier mydbinstance
```

多可用区数据库实例部署

适用于 InfluxDB 的 Amazon Timestream 为使用带有单个备用数据库实例的多可用区部署的数据库实例提供高可用性和故障转移支持。这种类型的部署称为多可用区数据库实例部署。适用于 InfluxDB 的亚马逊 Timestream 使用亚马逊故障转移技术。

在多可用区数据库实例部署中，Amazon Timestream 会自动在不同的可用区域预配置和维护同步备用副本。主数据库实例可以跨可用区同步复制到备用副本以提供数据冗余。运行具有高可用性的数据库实例可以提高数据库实例故障和可用区中断期间的可用性。有关可用区的更多信息，请参阅AWS 区域和可用区。

> **Note**
>
> 高可用性选项不是只读场景的扩缩解决方案。您不能使用备用副本来提供读取流量。

使用 Amazon Timestream 控制台，您只需在创建数据库实例时在可用性和持久性配置部分中指定创建备用实例选项，即可创建多可用区数据库实例部署。您还可以使用 AWS Command Line Interface 或 Ama API zon Timestream 指定多可用区数据库实例部署。使用`create-db-instance`或CLI命令或`CreateDBInstance`API操作。

与单可用区部署相比，使用多可用区数据库部署的数据库实例会增加写入和提交延迟。这可能是因为发生了同步数据复制。尽管在可用区之间设计了低延迟的网络连接，但如果您的部署故障转移到备用

副本，AWS 则延迟可能会发生变化。对于生产工作负载，我们建议您使用内IOPS含的 12K 或 16K 存储，IOPS以获得快速、稳定的性能。有关数据库实例类的更多信息，请参阅 数据库实例类。

配置和管理多可用区部署

InfluxDB 多可用区部署的时间流只能有一个备用空间。当部署有一个备用数据库实例时，称为多可用区数据库实例部署。多可用区数据库实例部署有一个备用数据库实例，可提供故障转移支持，但不提供读取流量。

> **Important**
>
> 您的实例必须至少有两个与之关联的子网，才能执行从单可用区到多可用区的更新。实例创建后，您无法将其部署模式从单可用区修改为多可用区。

您可以使用 AWS Management Console 来确定您的数据库实例是单可用区部署还是多可用区部署。

使用 AWS Management Console

1. 登录 AWS Management Console 并打开适用于 InfluxDB 的 Amazon Timestream 控制台。
2. 在导航窗格中，选择 InfluxDB 数据库，然后选择数据库标识符。

多可用区数据库实例部署具有以下特征：

- 数据库实例只有一行。
- Role（角色）的值为 Instance（实例）或 Primary（主要）。
- Multi-AZ（多可用区）的值为 Yes（是）。

亚马逊 Timestream 的故障转移流程

如果您的数据库实例因基础设施缺陷而导致计划内或计划外中断，如果您已开启多可用区，则适用于 InfluxDB 的 Amazon Timestream 会自动切换到其他可用区中的备用副本。完成故障转移所用的时间取决于在主数据库实例变为不可用时的数据库活动和其他条件。故障转移时间通常为 60–120 秒。不过，事务较多或时间较长的恢复过程可能延长故障转移时间。故障转移完成后，Timestream 控制台可能需要更多时间才能反映新的可用区。

> **Note**
>
> Amazon Timestream 会自动处理故障转移，因此您可以在没有管理干预的情况下尽快恢复数据库操作。如果出现下表中描述的任一情况，主数据库实例会自动切换到备用副本：

故障转移原因	描述
正在离线操作中修补 Timestream 数据库实例底层的操作系统。	在操作系统补丁或安全更新的维护时段内触发了故障切换。
Timestream 多可用区实例的主主机运行状况不佳。	多可用区数据库实例部署检测到受损的主数据库实例并进行故障转移。
由于网络连接中断，Timestream 多可用区实例的主主机无法访问。	Timestream 监控检测到主数据库实例存在网络可访问性故障，并触发了故障转移。
客户修改了 Timestream 实例。	InfluxDB 数据库实例修改的时间队触发了故障转移。有关更多信息，请参阅 更新数据库实例。
Timestream 多可用区主实例繁忙且没有响应。	主数据库实例没有响应。我们建议您执行以下操作：* 检查事件中是否使用了过多的CPU内存或交换空间。* 评估您的工作负载以确定您是否使用了适当的数据库实例类。有关更多信息，请参阅数据库实例类。
Timestream 多可用区实例的主主机底层的存储卷出现故障。	多可用区数据库实例部署在主数据库实例上检测到存储问题并进行故障转移。

设置DNS姓JVMTTL名查询

故障转移机制会自动更改数据库实例的域名系统 (DNS) 记录，使其指向备用数据库实例。因此，您需要重新建立与数据库实例之间的所有现有连接。在 Java 虚拟机 (JVM) 环境中，由于 Java DNS 缓存机制的工作原理，您可能需要重新配置JVM设置。

JVM缓存DNS名称查询。当将主机名JVM解析为 IP 地址时，它会将该 IP 地址缓存一段指定的时间，称为 time-to-live() TTL。

由于 AWS 资源使用的DNS名称条目偶尔会发生变化，因此我们建议您JVM使用不超过 60 秒的TTL值进行配置。这样做可以确保当资源的 IP 地址更改时，您的应用程序可以通过重新查询来接收和使用该资源的新 IP 地址。DNS

在某些 Java 配置中，会设置JVMTTL默认值，使其在重新启动之前永远不会刷新DNS条目。JVM因此，如果在应用程序仍在运行时 AWS 资源的 IP 地址发生了变化，则在您手动重启JVM并刷新缓存的 IP 信息之前，该资源将无法使用该资源。在这种情况下，必须设置，TTL以便定期刷新其缓存的 IP 信息。JVM

你可以TTL通过检索networkaddress.cache.ttl属性值来获得JVM默认值：

```
String ttl = java.security.Security.getProperty("networkaddress.cache.ttl");
```

> **Note**
>
> 根据您的版本JVM以及是否安装了安全管理器，默认值TTL可能会有所不同。许多JVMs提供的默认值TTL小于 60 秒。如果您使用的是这样的JVM，而不是使用安全管理器，则可以忽略本主题的其余部分。
> 要修改，请设置 JVM n TTL etworkaddress.cache.ttl 属性值。根据您的需求，使用下列方法之一：
>
> - 要为使用的所有应用程序全局设置属性值JVM，请在$JAVA_HOME/jre/lib/security/java.security文件networkaddress.cache.ttl中设置。
>
>   ```
>   networkaddress.cache.ttl=60
>   ```
>
> - 要仅在本地为应用程序设置属性，请在建立任何网络连接之前，在应用程序的初始化代码中设置 networkaddress.cache.ttl。
>
>   ```
>   java.security.Security.setProperty("networkaddress.cache.ttl" , "60");
>   ```

设置在 Timestream Influxdb 实例上查看 InfluxDB 日志

默认情况下，InfluxDB 会生成转到标准输出的日志。有关更多信息，请参阅管理 InfluxDB 日志

要查看从您通过 Timestream InfluxDB 创建的实例生成的InfluxDB日志，我们提供了提供每小时日志的机会。这些日志将转到您在创建实例之前必须创建的指定的 S3 存储桶。

- 在创建实例之前，提供的 Amazon S3 存储桶还必须通过提供带有 Timestream InfluxDB 服务主体的存储桶策略来授予 Timestream-InfluxDB 向该存储桶发送日志的权限，如下所示（替换 *{BUCKET_NAME}* 使用您的 Amazon S3 存储桶的真实名称：

```
{
    "Version": "2012-10-17",
    "Id": "PolicyForInfluxLogs",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream-influxdb.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::{BUCKET_NAME}/InfluxLogs/*"
        }
    ]
}
```

- 提供的存储桶必须与您创建的 Timestream InfluxDB 实例位于同一个账户和相同区域中

以下是你可以调用的命令来创建一个实例来接收涌入日志：

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

以下是此参数的格式。

```
-- log-delivery-configuration
{
    "S3Configuration": {
      "BucketName": "string",
      "Enabled": true|false
    }
}
```

- 此字段不是必填字段，并且默认情况下不启用日志记录。
- 不设置此字段等同于未启用日志。
- 日志将发送到前缀为的指定存储桶`InfluxLogs/`。
- 创建实例后，您可以使用`update-db-instance`API命令修改日志传输配置。

InfluxDB 提供不同类型的日志。可以通过设置 InfluxDB 参数来配置这些参数。使用 flux-log-enabled和日志级别参数来配置从实例发出的日志的类型。有关更多信息，请参阅 支持的参数和参数值。

向资源添加标签和标注

你可以使用标签为 InfluxDB 资源标记 Amazon Timestream。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。

亚马逊弹性计算云（亚马逊EC2）、亚马逊简单存储 AWS 服务（Amazon S3）、InfluxDB 的 Timestream 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。

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。

添加标签限制

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：

- InfluxDB 数据库实例的每个 Timestream 只能有一个具有相同密钥的标签。如果您尝试添加现有标签，则现有标签值将更新为新值。
- 值在标签类别中充当描述符。在 InfluxDB 的 Timestream 中，该值不能为空或为空。
- 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-assigned 的标签名称和值会自动分配aws:前缀，但您无法分配前缀。 AWS-分配的标签名称不计入 50 的标签限制。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。

InfluxDB Timestream 安全最佳实践

优化对 InfluxDB 的写入

与其他任何时间序列数据库一样，InfluxDB旨在能够实时摄取和处理数据。为了使系统保持最佳性能，我们建议在向InfluxDB写入数据时进行以下优化：

- 批量写入：向InfluxDB写入数据时，请分批写入数据，以最大限度地减少与每个写入请求相关的网络开销。最佳批量大小为每个写入请求使用 5000 行线路协议。要在一个请求中写入多行，每行协议必须用新行 (\ n) 分隔。
- 按键对标签进行排序：在将数据指向InfluxDB写入数据之前，请按字典顺序按键对标签进行排序。

```
measurement,tagC=therefore,tagE=am,tagA=i,tagD=i,tagB=think fieldKey=fieldValue
 1562020262

# Optimized line protocol example with tags sorted by key
measurement,tagA=i,tagB=think,tagC=therefore,tagD=i,tagE=am fieldKey=fieldValue
 1562020262
```

- 尽可能使用最粗略的时间精度：— InfluxDB 以纳秒精度写入数据，但是，如果您的数据不是以纳秒为单位收集的，则无需以该精度写入。为了获得更好的性能，请对时间戳使用尽可能粗略的精度。在以下情况下，您可以指定写入精度：
  - 使用时，SDK您可以指定 WritePrecision 何时设置点的时间属性。有关 InfluxDB 客户端库的更多信息，请参阅 In f luxDB 文档。
  - 使用 Telegraf 时，您可以在 Telegraf 代理配置中配置时间精度。精度指定为以整数 + 为单位的间隔（例如 0s,10ms,2us,4s）。有效的时间单位为 “ns”、“us”、“ms” 和 “s”。

    ```
    [agent]
     interval ="10s"
     metric_batch_size="5000"
     precision = "0s"
    ```

- 使用 gzip 压缩：— 使用 gzip 压缩来加快写入 InfluxDB 的速度并减少网络带宽。基准测试显示，压缩数据后，速度提高了5倍。

- 使用 Telegraf 时，在 telegraf.conf 的 Influxdb_v2 输出插件配置中，将 content_encoding 选项设置为 gzip：

```
[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  # ...
  content_encoding = "gzip"
```

- 使用客户端库时，每个 InfluxDB 客户端库都提供用于压缩写入请求或默认强制压缩的选项。每个库的启用压缩的方法各不相同。有关具体说明，请参阅 InfluxDB 文档
- 使用 InfluxDB API `/api/v2/write` 端点写入数据时，使用 gzip 压缩数据并将内容编码标头设置为 gzip。

为性能而设计

设计架构以实现更简单、性能更高的查询。以下准则将确保您的架构易于查询并最大限度地提高查询性能：

- 为查询而设计：选择易于查询的测量值、标签键和字段键。要实现此目标，请遵循以下原则：
  - 使用名称简单且能准确描述架构的度量。
  - 避免在同一个架构中对标签键和字段键使用相同的名称。
  - 避免在标签和字段键中使用保留的 Flux 关键字和特殊字符。
  - 标签存储描述字段的元数据，这些元数据在许多数据点中都很常见。
  - 字段存储唯一或高度可变的数据，通常是数字数据点。
  - 测量值和密钥不应包含数据，而应用于聚合或描述数据。数据将存储在标签和字段值中。
- 控制时间序列基数高序列基数是InfluxDB中写入和读取性能下降的主要原因之一。在InfluxDB的上下文中，高基数是指存在大量的唯一标签值。在 InfluxDB 中对标签值进行索引，这意味着大量的唯一值将生成更大的索引，从而降低数据摄取和查询性能。

  为了更好地理解和解决潜在的高基数相关问题，可以按照以下步骤操作：
  - 了解高基数的原因
  - 测量存储桶的基数
  - 采取行动解决高基数问题
- 高序列基数的原因 Influx DB 根据测量值和标签对数据进行索引，以加快数据读取速度。每组已编入索引的数据元素构成一个序列键。包含高度可变的信息（例如唯一值IDs、哈希值和随机字符串）的

@@ 标签会导致大量序列，也称为高序列基数。高系列基数是 InfluxDB 中高内存使用率的主要驱动因素。

- 测量序列基数如果您遇到性能下降或在 Timestream for InfluxDB 实例中看到内存使用量不断增加，我们建议您测量存储桶的系列基数。

  InfluxDB 提供的函数允许你在 FluxQL 和 InfluxQL 中测量序列基数。

  - 在 Flux 中使用该函数 `influxdb.cardinality()`
  - 在 FluxQL 中使用以下命令 `SHOW SERIES CARDINALITY`

  在这两种情况下，引擎都将返回数据中唯一序列密钥的数量。请记住，不建议在任何 Timestream 上使用超过 1000 万个 InfluxDB 实例的系列密钥。

- 高序列基数的原因如果你遇到任何一个存储桶的基数都很高，你可以采取一些更正步骤来修复这个问题：

  - 查看您的标签：确保您的工作负载不会生成大多数条目标签具有唯一值的情况。如果唯一标签值的数量总是会随着时间的推移而增加，或者日志类型的消息被写入数据库，其中每条消息都有时间戳、标签等的唯一组合，则可能会发生这种情况。你可以使用以下 Flux 代码来帮助你找出哪些标签对你的高基数问题影响最大：

```
// Count unique values for each tag in a bucketimport "influxdata/influxdb/schema"

cardinalityByTag = (bucket) => schema.tagKeys(bucket: bucket)
    |> map(
        fn: (r) => ({
            tag: r._value,
            _value: if contains(set: ["_stop", "_start"], value: r._value) then
                0
            else
                (schema.tagValues(bucket: bucket, tag: r._value)
                    |> count()
                    |> findRecord(fn: (key) => true, idx: 0))._value,
        }),
    )
    |> group(columns: ["tag"])
    |> sum()

cardinalityByTag(bucket: "example-bucket")
```

  如果您遇到非常高的基数，则上面的查询可能会超时。如果您遇到超时，请逐一运行以下查询。

生成标签列表：

```
// Generate a list of tagsimport "influxdata/influxdb/schema"

schema.tagKeys(bucket: "example-bucket")
```

计算每个标签的唯一标签值：

```
// Run the following for each tag to count the number of unique tag valuesimport
 "influxdata/influxdb/schema"

tag = "example-tag-key"

schema.tagValues(bucket: "my-bucket", tag: tag)
    |> count()
```

我们建议您在不同的时间点运行这些代码，以确定哪个标签的增长速度更快。

- 改进您的架构：遵循我们中讨论的建模建议InfluxDB Timestream 安全最佳实践。
- 移除或聚合较旧的数据以减少基数：考虑您的用例是否需要所有导致高基数问题的数据。如果不再需要这些数据或不经常访问这些数据，则可以对其进行聚合、删除或将其导出到其他引擎，例如用于实时分析的 Timestream 进行长期存储和分析。

故障排除

无法识别 “开发” 版本的警告

迁移期间可能会显示警告 “WARN：假设支持最新的备份/恢复APIs，则无法解析服务器报告的版本 “dev”。可以忽略此警告。

恢复阶段迁移失败

如果在恢复阶段迁移失败，用户可以使用该`--retry-restore-dir`标志重新尝试恢复。使用带有先前备份目录路径的`--retry-restore-dir`标志跳过备份阶段并重试恢复阶段。如果在还原期间迁移失败，则会显示创建的用于迁移的备份目录。

恢复失败的可能原因包括：

- InfluxDB 目标令牌无效 — 目标实例中存在的与源实例同名的存储桶。对于单个存储桶迁移，请使用`--dest-bucket`选项为迁移的存储桶设置唯一的名称
- 连接失败，无论是源主机还是目标主机，还是与可选的 S3 存储桶。

适用于 InfluxDB 的亚马逊 Timestream 基本操作指南

以下是每个人在使用适用于 InfluxDB 的 Amazon Timestream 时都应遵循的基本操作指南。请注意，适用于 InfluxDB 的 Amazon Timestream 服务等级协议要求您遵循以下指南：

- 使用指标来监控您的内存和存储使用情况。CPU您可以将 Amazon 设置 CloudWatch 为在使用模式发生变化或接近部署容量时通知您。这样，您就可以保持系统的性能和可用性。
- 当接近存储容量限制时，可以纵向扩展数据库实例。存储和内存中应含有一些缓冲区，以适应应用程序的意外增大需求。请记住，此时，您需要创建一个新实例并迁移数据才能实现这一目标。
- 如果您的数据库工作负载需要的 I/O 超过您的配置，那么出现故障转移或数据库故障后，恢复的速度将会变缓。要增加数据库实例的 I/O 容量，请执行以下任一或所有操作：
  - 迁移到具有更高 I/O 容量的其他数据库实例。
  - 如果您已经在使用 Influx IOPS 包含的存储空间，请配置IOPS包含更高的存储类型。
- 如果您的客户端应用程序正在缓存数据库实例的域名服务 (DNS) 数据，请将 time-to-live (TTL) 值设置为小于 30 秒。数据库实例的底层 IP 地址在故障转移后可能会发生变化。因此，长时间缓存DNS数据可能会导致连接失败。您的应用程序可能会尝试连接到不再使用的 IP 地址。

数据库实例RAM建议

适用于 InfluxDB 的 Amazon Timestream 性能最佳实践是分配足够的资源，RAM这样你的工作集几乎完全驻留在内存中。工作集是经常在实例上使用的数据和索引。使用数据库实例的次数越多，工作集的增长量就越大。

InfluxDB 的时间流中的安全性

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。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 Timestream for InfluxDB 的合规性计划，请参阅按合规计划划分的范围内的 AWS 服务。

- 云端安全-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。

本文档将帮助您了解在使用 Timestream for InfluxDB 时如何应用分担责任模型。以下主题向您展示了如何为 InfluxDB 配置 Timestream 以满足您的安全性和合规性目标。您还将学习如何使用其他AWS服务来帮助您监控和保护InfluxDB资源的时间流。

主题

概述

本文档可帮助您了解在使用 Amazon Timestream for InfluxDB 时如何应用分担责任模型。以下主题向您展示了如何为 InfluxDB 配置 Amazon Timestream 以实现您的安全与合规目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 Amazon Timestream for InfluxDB 资源。

您可以管理对 Amazon Timestream 的 InfluxDB 资源和数据库实例上的数据库的访问权限。你用来管理访问权限的方法取决于用户需要使用适用于 InfluxDB 的 Amazon Timestream 执行的任务类型：

- 在基于 Amazon VPC 服务的虚拟私有云 (VPC) 中运行您的数据库实例，以实现网络访问控制。

- 使用 AWS Identity and Access Management (IAM) 策略分配权限，以确定允许谁管理 Amazon Timestream 的 InfluxDB 资源。例如，您可以使用IAM来确定允许谁创建、描述、修改和删除数据库实例、标记资源或修改安全组。
- 使用安全组控制哪些 IP 地址或 Amazon EC2 实例可以连接到数据库实例上的数据库。首次创建数据库实例时，只能通过关联安全组指定的规则访问该实例。
- 使用安全套接字层 (SSL) 或传输层安全 (TLS) 连接您的数据库实例。
- 使用 InfluxDB 引擎的安全功能来控制谁可以登录数据库实例上的数据库。这些功能就像本地网络上的数据库一样工作。有关更多信息，请参阅 InfluxDB 的时间流中的安全性。

> **Note**
>
> 您必须仅为您的使用案例配置安全性。你不必为适用于 InfluxDB 的 Amazon Timestream 管理的进程配置安全访问权限。这些过程包括创建备份、在主数据库实例和只读副本之间复制数据以及其他过程。

主题

一般安全

主题

权限

应向InfluxDB用户授予最低权限权限。迁移期间只能使用授予特定用户的令牌，而不是操作员令牌。

InfluxDB 的时间流使用IAM权限来控制用户权限。我们建议向用户授予他们所需的特定操作和资源的访问权限。有关更多信息，请参阅授予最低权限访问权限。

网络访问

Influx 迁移脚本可以在本地运行，在同一系统上的两个 InfluxDB 实例之间迁移数据，但假设迁移的主要用例是通过网络（本地或公共网络）迁移数据。随之而来的是安全方面的考虑。默认情况下，Influx 迁移脚本将验证TLS启用状态的实例的TLS证书：我们建议用户TLS在其 InfluxDB 实例中启用证书，不要在脚本中使用该`--skip-verify`选项。

我们建议您使用允许列表来限制来自预期来源的网络流量。为此，您可以将网络流量限制为仅来自已知的 InfluxDB 实例。IPs

依赖项

应使用所有依赖项的最新主要版本，包括 Influx、InfluxDB CLI、Python、Requests 模块以及可选的依赖项，例如和。`mountpoint-s3` `rclone`

S3 存储桶

如果将 S3 存储桶用作迁移的临时存储，我们建议启用TLS、版本控制和禁用公共访问。

使用 S3 存储桶进行迁移

1. 打开 AWS Management Console，导航至 Amazon 简单存储服务，然后选择存储桶。
2. 选择您要使用的存储桶。
3. 选择权限选项卡。
4. 在屏蔽公共访问权限（存储桶设置）下，请选择编辑。
5. 选中 “阻止所有公共访问”。
6. 选择 Save changes（保存更改）。
7. 在 Bucket policy（存储桶策略）下，请选择 Edit（编辑）。
8. 输入以下内容，<example-bucket>替换为您的存储桶名称，以强制使用 1.2 或更高TLS版本进行连接：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceTLSv12orHigher",
            "Principal": {
                "AWS": "*"
```

```
            },
            "Action": [
                "s3:*"
            ],
            "Effect": "Deny",
            "Resource": [
                "arn:aws:s3:::<example bucket>/*",
                "arn:aws:s3:::<example bucket>"
            ],
            "Condition": {
                "NumericLessThan": {
                    "s3:TlsVersion": 1.2
                }
            }
        }
    ]
}
```

9. 选择 Save changes（保存更改）。
10. 选择属性选项卡。
11. 在存储桶版本控制下，请选择编辑。
12. 选中 “启用”。
13. 选择 Save changes（保存更改）。

有关 Amazon S3 存储桶最佳安全实践的信息，请参阅亚马逊简单存储服务的安全最佳实践。

使用适用于 InfluxDB 的 Amazon Timestream 进行数据库身份验证

适用于 InfluxDB 的 Amazon Timestream 支持两种对数据库用户进行身份验证的方法。

密码和访问令牌数据库身份验证使用不同的数据库身份验证方法。因此，特定用户只能使用一种身份验证方法登录到数据库。在这两种情况下，InfluxDB都会执行用户帐户和API令牌的所有管理。

密码验证

在 InfluxDB 数据库实例创建过程中，您创建了组织、用户和密码。用户有权管理您的 InfluxDB 数据库实例的 Timestream 中的所有内容。使用此用户名和密码组合，您将能够使用InfluxUI LogIn 进入您的实例，还可以使用Influx生成操作CLI员令牌。

创建用户、删除存储桶、组织等需要操作员令牌。有关更多信息，请参阅 数据库身份验证选项。

API代币

InfluxDB API 令牌可确保InfluxDB与外部工具（例如客户端或应用程序）之间的安全交互。API令牌属于特定用户，用于标识用户组织内的InfluxDB权限。

InfluxDB 中有三种类型的API代币：

- 操作员令牌：授予对InfluxDB 2.x OSS 中所有组织和所有组织资源的完全读写权限。某些操作（例如检索服务器配置）需要操作员权限。要在设置过程完成CLI后使用 InfluxDB UI 或 Influx 手动创建操作员令牌，必须使用现有的操作员令牌或用户名和密码。`api/v2` API要在不使用现有操作员令牌的情况下创建新的操作员令牌，请参阅 infl uxd 恢复身份验证CLI。

  > **Important**
  >
  > 由于操作员令牌对数据库中的所有组织具有完全的读写权限，因此我们建议为每个组织创建一个All-Access令牌并使用这些令牌来管理InfluxDB。这有助于防止组织间的意外互动。

- All-Access API Token：授予对组织中所有资源的完全读写权限。
- 读/写令牌：授予组织中特定存储桶的读取权限、写入权限或两者兼而有之。

所有 InfluxDb 令牌都是长期存在的代币，没有设定的到期日期，因此不建议使用您的操作员或所有访问令牌从您的客户或Telegraf代理发送的监控数据，也不建议将其嵌入仪表板应用程序中。对于这些应用程序，只需拥有完成工作所需的权限即可创建读/写令牌。有关如何创建InfluxDB令牌的更多信息，请参阅创建令牌。

密文

InfluxDB 操作员令牌是在实例设置时生成的；其他类型的令牌，例如所有访问令牌和读/写令牌，可以使用 Influx、Influx v2 API 或 Timestream for Inf CLI luxDB 多用户轮换功能创建。有关如何生成、查看、分配和删除令牌，请参阅管理API令牌。

我们建议您轮换 Timestream 以获取经常使用的 InfluxDB 令牌， AWS Secrets Manager 并通过环境变量存储令牌。有关环境变量中令牌的使用情况以及如何轮换 InfluxDB 用户和轮换秘密令牌的时间流，请参阅使用令牌。

另请参见：

- 适用于 InfluxDB 的 Amazon Timestream 中的基础设施安全
- InfluxDB Timestream 安全最佳实践

适用于 InfluxDB 的亚马逊 Timestream 如何使用机密

InfluxDB 的 Timestream 支持通过用户界面进行用户名和密码身份验证，以及用于最低权限客户端和应用程序连接的令牌凭据。InfluxDB 用户的 Timestream 在其组织内拥有`allAccess`权限，而令牌可以拥有任何一组权限。遵循安全API令牌管理的最佳实践，应创建用户来管理令牌，以便在组织内进行精细访问。有关InfluxDB的Timestream管理员最佳实践的更多信息，可以在Influxdata文档中找到。

AWS Secrets Manager 是一项密钥存储服务，可用于保护数据库凭据、API密钥和其他机密信息。然后，在你的代码中，你可以用调API用 Secrets Manager 来替换硬编码的凭据。这有助于确保密钥不会被检查你的代码的人泄露，因为密码不存在。有关 Secrets Manager 的概述，请参阅什么是 S AWS ecrets Manager。

当您创建数据库实例时，InfluxDB 的 Timestream 会自动创建一个管理员密钥供您使用多用户轮换功能。 AWS Lambda 为了轮换 InfluxDB 用户和令牌的 Timestream，您必须手动为要轮换的每个用户或令牌创建一个新的密钥。使用 Lambda 函数，可以将每个密钥配置为按计划轮换。设置新的轮换密钥的过程包括上传 Lambda 函数代码、配置 Lambda 角色、定义新密钥和配置密钥轮换计划。

密钥的内容

当你将 InfluxDB 用户凭证的 Timestream 存储在密钥中时，请使用以下格式。

单用户：

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbIdentifier": "<required: DB identifier>"
}
```

当您为 InfluxDB 实例创建 Timestream 时，管理员密钥会自动存储在 Secrets Manager 中，其中包含用于多用户 Lambda 函数的证书。将设置`adminSecretArn`为在数据库实例摘要页面上找到的`Authentication Properties Secret Manager ARN`值或管理员密钥ARN的值。要创建新的管理员密钥，您必须已经拥有关联的凭据，并且这些凭据必须具有管理员权限。

当你将 InfluxDB 令牌凭证的 Timestream 存储在密钥中时，请使用以下格式。

多用户：

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
```

```
  "org": "<required: organization to associate token with>",
  "adminSecretArn": "<required: ARN of the admin secret>",
  "type": "<required: allAccess or operator or custom>",
  "dbIdentifier": "<required: DB identifier>",
  "token": "<required unless generating a new token: token being rotated>",
  "writeBuckets": "<optional: list of bucketIDs for custom type token, must be input
 within plaintext panel, for example ['id1','id2']>",
  "readBuckets": "<optional: list of bucketIDs for custom type token, must be input
 within plaintext panel, for example ['id1','id2']>",
  "permissions": "<optional: list of permissions for custom type token, must be input
 within plaintext panel, for example ['write-tasks','read-tasks']>"
}
```

当你将 InfluxDB 管理员凭据的 Timestream 存储在密钥中时，请使用以下格式：

管理员密钥：

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbIdentifier": "<required: DB identifier>",
  "organization": "<optional: initial organization>",
  "bucket": "<optional: initial bucket>"
}
```

要启用密钥的自动轮换，密钥的JSON结构必须正确。轮换秘密有关如何轮换 InfluxDB 机密的时间流，请参阅。

修改密钥

在 Timestream for InfluxDB 实例创建过程中生成的凭据存储在你账户的 Secrets Manager 密钥中。GetDbInstance响应对象包含一个，`influxAuthParametersSecretArn`其中包含此类机密的 Amazon 资源名称 (ARN)。只有在您的 InfluxDB 实例的时间流可用后，才会填充该密钥。这是一份READONLY副本，因为此密钥的任何updates/modifications/deletions副本都不会影响创建的数据库实例。如果您删除此密钥，则API响应仍将引用已删除的密钥ARN。

要在 InfluxDB 实例的 Timestream 中创建新令牌而不是存储现有令牌凭证，您可以通过将密钥中的`token`值留空，然后使用多用户轮换函数（将 Lambda `AUTHENTICATION_CREATION_ENABLED`环境变量设置为`true`）来创建非操作员令牌。如果您创建新令牌，则密钥中定义的权限将分配给该令牌，并且在首次成功轮换后无法更改。有关轮换密钥的更多信息，请参阅轮换 Secrets Manager AWS 密钥。

如果删除了密钥，则不会删除 InfluxDB 实例的 Timestream 中的关联用户或令牌。

轮换秘密

你可以使用 InfluxDB 单用户和多用户轮换的时间流 Lambda 函数来轮换 InfluxDB 用户和令牌凭证的时间流。使用单用户 Lambda 函数轮换 InfluxDB 实例的 Timestream 的用户证书，并使用多用户 Lambda 函数轮换 InfluxDB 实例的时间流的令牌凭证。

使用单用户和多用户 Lambda 函数轮换用户和令牌是可选的。InfluxDB 凭证的时间流永不过期，任何暴露的凭据都可能对您的数据库实例进行恶意操作。使用 Secrets Manager 轮换 InfluxDB 凭证的 Timestream 的好处是增加了安全层，可以将暴露凭证的攻击向量限制在下一个轮换周期之前的时间范围内。如果您的数据库实例没有轮换机制，则在手动删除之前，任何公开的凭证都将一直有效。

此外，您还可以配置 Secrets Manager 以根据指定的计划自动轮换密钥。这使您能够将长期密钥替换为短期密钥，这有助于显著减少泄露风险。有关使用 Secrets Manager 轮换密钥的更多信息，请参阅轮换 S AWS ecrets Manager 密钥。

轮换用户

当您使用单用户 Lambda 函数轮换用户时，每次轮换后都会为该用户分配一个新的随机密码。有关如何启用自动轮换的更多信息，请参阅为非数据库 Secrets Manager AWS 密钥设置自动轮换。

轮换管理员密钥

要轮换管理员密钥，您可以使用单用户轮换功能。您需要将`engine`和`dbIdentifier`值添加到密钥中，因为这些值不会在数据库初始化时自动填充。密钥的内容有关完整的密钥模板，请参阅。

要查找 InfluxDB Timestream 实例的管理员密钥，你可以使用 InfluxDB Timestream 实例摘要页面中的管理员密钥ARN。建议您轮换所有 Timestream 以获取 InfluxDB 管理员机密，因为管理员用户对 InfluxDB 实例的 Timestream 拥有更高的权限。

Lambda 轮换函数

您可以使用单用户轮换功能轮换 InfluxDB 用户的时间流，方法是使用带有新密钥的，然后为 InfluxDB 用户的 Timestream 添加必填字段。密钥的内容有关秘密轮换 Lambda 函数的更多信息，请参阅按 Lam bda 函数轮换。

单用户轮换函数使用密钥中定义的凭据通过 Timestream for InfluxDB 数据库实例进行身份验证，然后生成新的随机密码并为用户设置新密码。有关秘密轮换 Lambda 函数的更多信息，请参阅按 Lam bda 函数轮换。

Lambda 函数执行角色权限

使用以下IAM策略作为单用户 Lambda 函数的角色。该策略为 Lambda 函数提供了为InfluxDB用户执行Timestream秘密轮换所需的权限。

用您 AWS 账户中的值替换IAM政策中下面列出的所有项目：

- {rotating_secret_arn} — 正在轮换的ARN密钥可以在 Secrets Manager 的机密细节中找到。
- {db_instance_arn} — InfluxDB 实例的时间流ARN可以在 InfluxDB 实例的时间流摘要页面上找到。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
     "Resource": "{rotating_secret_arn}"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "timestream-influxdb:GetDbInstance"
            ],
     "Resource": "{db_instance_arn}",
            "Effect": "Allow"
        }
    ]
}
```

轮换代币

您可以使用多用户轮换功能轮换 InfluxDB 令牌的时间流，方法是使用带有新密钥的，然后为 InfluxDB 令牌的 Timestream 添加必填字段。密钥的内容有关秘密轮换 Lambda 函数的更多信息，请参阅按 Lam bda 函数轮换。

你可以使用 InfluxDB 的 Timestream for InfluxDB 多用户 Lambda 函数轮换 InfluxDB 令牌的时间流。在 Lambda 配置`true`中将`AUTHENTICATION_CREATION_ENABLED`环境变量设置为以启用令牌创建。要创建新令牌，请使用密钥的内容作为您的密钥值。省略新密`token`钥中的键值对并将设置`type`为`allAccess`，或者定义特定权限并将类型设置为。`custom`轮换功能将在第一个轮换周期中创建一个新代币。轮换后，您无法通过编辑密钥来更改令牌权限，任何后续轮换都将使用在数据库实例中设置的权限。

Lambda 轮换函数

多用户轮换功能通过使用管理员密钥中的管理员凭据创建新的权限相同令牌来轮换令牌凭证。Lambda 函数会在创建替换令牌、将新令牌值存储在密钥中以及删除旧令牌之前验证密钥中的令牌值。如果 Lambda 函数正在创建新标记，它将首先验证`AUTHENTICATION_CREATION_ENABLED`环境变量设置为`true`、密钥中没有令牌值以及令牌类型不是类型运算符。

Lambda 函数执行角色权限

使用以下IAM策略作为多用户 Lambda 函数的角色。该策略为 Lambda 函数提供了所需的权限，以便对 InfluxDB 令牌的 Timestream 执行秘密轮换。

用您 AWS 账户中的值替换IAM政策中下面列出的所有项目：

- {rotating_secret_arn} — 正在轮换的ARN密钥可以在 Secrets Manager 的机密细节中找到。
- {authentication_properties_admin_secret_arn} — InfluxDB 管理员密钥的时间流可以在 InfluxDB 实例的时间流摘要页面上找到。ARN
- {db_instance_arn} — InfluxDB 实例的时间流ARN可以在 InfluxDB 实例的时间流摘要页面上找到。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
```

```
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
     "Resource": "{rotating_secret_arn}"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
     "Resource": "{authentication_properties_admin_secret_arn}"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "timestream-influxdb:GetDbInstance"
            ],
     "Resource": "{db_instance_arn}",
            "Effect": "Allow"
        }
    ]
}
```

InfluxDB 的 Timestream 中的数据保护

分担责任模型 AWS 分担责任模型适用于InfluxDB的Amazon Timestream中的数据保护。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础架构上的内容的控制。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FAQ。 有关欧洲数据保护的信息，请参阅责任AWS 共担模型和AWS安全GDPR博客上的博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭据并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。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 资源通信。我们需要 TLS 1.2，建议使用 TLS 1.3。
- 使用API进行设置和用户活动记录 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 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 AWS 通过命令行界面或访问时需要 FIPS 140-3 经过验证的加密模块API，请使用端点。FIPS有关可用FIPS端点的更多信息，请参阅联邦信息处理标准 (FIPS) 140-3。

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）。这包括你使用控制台、、API或 AWS 服务 使用适用于 InfluxDB 或其他版本的 Timestream 时。 AWS CLI AWS SDKs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URL向外部服务器提供，我们强烈建议您不要在中包含凭据信息，URL以验证您对该服务器的请求。

有关InfluxDB数据保护主题的 Timestream 的更多详细信息，例如静态加密和密钥管理，请选择以下任何可用主题。

主题

静态加密

InfluxDB 静态加密的 Timestream 使用存储在 () 中的加密密钥对所有静态数据进行加密，从而增强安全性。AWS Key Management ServiceAWS KMS此功能减少保护敏感数据时涉及的操作负担和复杂性。利用静态加密，可以构建符合严格加密合规性和法规要求的安全敏感型应用程序。

- InfluxDB 数据库实例的 Timestream 上的加密功能默认处于启用状态，并且无法关闭。行业标准的 AES -256 加密算法是使用的默认加密算法。
- AWS KMS 用于在 InfluxDB 的 Timestream 中进行静态加密。
- 您无需修改数据库实例客户端应用程序即可使用加密。

传输中加密

InfluxDB 的所有时间流数据在传输过程中都经过了加密。默认情况下，InfluxDB 的 Timestream 和来自 Timestream 的所有通信都使用传输层安全 () TLS 加密进行保护。

使用支持的 1.2 或 1.3 TLS 版本保护进出亚马逊 Timestream for InfluxDB 的流量。

适用于 InfluxDB 的亚马逊 Timestream 的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可以帮助管理员安全地控制对 AWS 资源的访问权限。IAM管理员控制谁可以进行身份验证（登录）和授权（有权限）使用Timestream获取InfluxDB资源。IAM无需支付额外费用即可使用。 AWS 服务

主题

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 AWS 账户根用户、IAM用户身份或通过担任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身份中心）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，您的管理员之前使用IAM角色设置了联合身份。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API请求AWS 签名版本 4。

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，请参阅用户指南中的多因素身份验证和AWS IAM Identity Center 用户指南IAM中的AWS 多因素身份验证。IAM

IAM 用户和组

IAM用户是您内部 AWS 账户 对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时证书，而不是创建拥有密码和访问密钥等长期凭证的IAM用户。但是，如果您有需要IAM用户长期凭证的特定用例，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》中的定期轮换需要长期凭证的用例的访问密钥。

IAM群组是指定IAM用户集合的身份。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，您可以拥有一个名为的群组，IAMAdmins并授予该群组管理IAM资源的权限。

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，请参阅用户指南中的IAMIAM用户用例。

IAM角色

IAM角色是您内部具有特定权限 AWS 账户 的身份。它与IAM用户类似，但与特定人员无关。要在中临时扮IAM演角色 AWS Management Console，可以从用户切换到IAM角色（控制台）。您可以通过调用 AWS CLI 或 AWS API操作或使用自定义操作来代入角色URL。有关使用角色的方法的更多信息，请参阅《IAM用户指南》中的代入角色的方法。

IAM具有临时证书的角色在以下情况下很有用：

- 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。有关用于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。为了控制您的身份在进行身份验证后可以访问的内容，Ident IAM ity Center 会将权限集关联到中的IAM角色。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 用户或角色可以代入一个IAM角色，为特定任务临时获得不同的权限。

- 跨账户访问-您可以使用IAM角色允许其他账户中的某人（受信任的委托人）访问您账户中的资源。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）。要了解角色和基于资源的跨账户访问策略之间的区别，请参阅IAM用户指南IAM中的跨账户资源访问权限。
- 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，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序EC2或在 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，然后此操作在其他服务中启动另一个操作。FAS使用调用委托人的权限 AWS 服务以及 AWS 服务 向下游服务发出请求的请求。FAS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。有关提出FAS请求时的政策详情，请参阅转发访问会话。
  - 服务角色-服务IAM角色是服务代替您执行操作的角色。IAM管理员可以在内部创建、修改和删除服务角色IAM。有关更多信息，请参阅《IAM用户指南》 AWS 服务中的创建角色以向委派权限。
  - 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 Amazon 上运行的应用程序 EC2 — 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。这比在EC2实例中存储访问密钥更可取。要为EC2实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含该角色，并允许在EC2实例上运行的程序获得临时证书。有关更多信息，请参阅IAM用户指南中的使用IAM角色向在 Amazon EC2 实例上运行的应用程序授予权限。

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策略文档结构和内容的更多信息，请参阅《IAM用户指南》中的JSON策略概述。

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。要授予用户对其所需资源执行操作的权限，IAM管理员可以创建IAM策略。然后，管理员可以将IAM策略添加到角色中，用户可以代入这些角色。

IAM无论您使用何种方法执行操作，策略都会定义该操作的权限。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、或获取角色信息 AWS API。

基于身份的策略

基于身份的策略是可以附加到身份（例如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用户指南中的使用客户托管策略定义自定义IAM权限。

基于身份的策略可以进一步归类为内联策略或托管策略。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。要了解如何在托管策略或内联策略之间进行选择，请参阅《IAM用户指南》中的在托管策略和内联策略之间进行选择。

基于资源的策略

基于资源的JSON策略是您附加到资源的策略文档。基于资源的策略的示例包括IAM角色信任策略和 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IAM中使用 AWS 托管策略。

访问控制列表 (ACLs)

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。ACLs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 AWS WAF、和亚马逊VPC就是支持的服务示例ACLs。要了解更多信息ACLs，请参阅《亚马逊简单存储服务开发者指南》中的访问控制列表 (ACL) 概述。

其他策略类型

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-权限边界是一项高级功能，您可以在其中设置基于身份的策略可以向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 (SCPs)-SCPs 是为中的组织或组织单位 (OU) 指定最大权限的JSON策略 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。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。对成员账户中的实体（包括每个实体）的权限进行了SCP限制 AWS 账户根用户。有关 Organization SCPs s 和的更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 会话策略 – 会话策略是当您以编程方式为角色或联合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。

适用于 InfluxDB 的亚马逊 Timestream 是如何使用的 IAM

IAM你可以在 InfluxDB 的 Amazon Timestream 上使用的功能

IAM功能	InfluxDB 支持的时间流
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件密钥	否
ACLs	否
ABAC（策略中的标签）	是
临时凭证	是

IAM功能	InfluxDB 支持的时间流
主体权限	是
服务角色	否
服务相关角色	是

要全面了解 InfluxDB 的 Timestream 和其他 AWS 服务如何与大多数IAM功能配合使用，请参阅《用户指南》IAMIAM中与之配合使用的AWS 服务。

InfluxDB 的 Timestream 基于身份的策略

支持基于身份的策略：是

基于身份的策略是可以附加到身份（例如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用户指南中的使用客户托管策略定义自定义IAM权限。

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。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可以在JSON策略中使用的所有元素，请参阅IAM用户指南中的IAMJSON策略元素参考。

InfluxDB Timestream 基于身份的策略示例

要查看 InfluxDB 基于身份的策略的时间流示例，请参阅... 适用于 InfluxDB 的 Amazon Timestream 的基于身份的策略示例

InfluxDB 的 Timestream 中基于资源的策略

支持基于资源的策略：否

基于资源的JSON策略是您附加到资源的策略文档。基于资源的策略的示例包括IAM角色信任策略和 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。

要启用跨账户访问权限，您可以将整个账户或另一个账户中的IAM实体指定为基于资源的策略中的委托人。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不

同位置时 AWS 账户，可信账户中的IAM管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM用户指南》IAM中的跨账户资源访问权限。

InfluxDB Timestream 的策略操作

支持策略操作：是

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`Action`元素描述了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。也有一些例外，例如没有匹配API操作的仅限权限的操作。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。

要查看 InfluxDB 操作的时间流列表，请参阅《服务授权参考》中的 Amazon Timestream 为 InfluxDB 定义的操作。

InfluxDB 的 Timestream 中的策略操作在操作前使用以下前缀：

```
timestream-influxdb
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "timestream-influxdb:action1",
      "timestream-influxdb:action2"
         ]
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 `Describe` 开头的所有操作，包括以下操作：

```
"Action": "timestream-influxdb:Describe*"
```

InfluxDB Timestream 的策略资源

支持策略资源：是

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。

ResourceJSON策略元素指定要应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。最佳做法是，使用资源的 Amazon 资源名称 (ARN) 来指定资源。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": "*"
```

要查看 InfluxDB 资源类型及其ARNs类型的时间流列表，请参阅《服务授权参考》中的 Amazon Timestream 为 InfluxDB 定义的资源。要了解您可以为每种资源指定哪些操作，请参阅 Amazon Timestream 为 InfluxDB 定义的操作。ARN

InfluxDB Timestream 的策略条件密钥

支持特定于服务的策略条件密钥：否

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，只有在资源上标有IAM用户的用户名时，您才能向IAM用户授予访问该资源的权限。有关更多信息，请参阅《IAM用户指南》中的IAM策略元素：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。

InfluxDB 的时间流中的访问控制列表 (ACLs)

支持ACLs：否

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。ACLs与基于资源的策略类似，尽管它们不使用JSON策略文档格式。

InfluxDB 带时间流的基于属性的访问控制 (ABAC)

支持ABAC（策略中的标签）：是

基于属性的访问控制 (ABAC) 是一种基于属性定义权限的授权策略。在中 AWS，这些属性称为标签。您可以将标签附加到IAM实体（用户或角色）和许多 AWS 资源。为实体和资源添加标签是的第一步。ABAC然后，您可以设计ABAC策略，允许在委托人的标签与他们尝试访问的资源上的标签匹配时进行操作。

ABAC在快速增长的环境中很有用，也有助于解决策略管理变得繁琐的情况。

要基于标签控制访问，您需要使用 `aws: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。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关更多信息ABAC，请参阅《IAM用户指南》中的使用ABAC授权定义权限。要查看包含设置步骤的教程ABAC，请参阅IAM用户指南中的使用基于属性的访问控制 (ABAC)。

在 InfluxDB 的 Timestream 中使用临时证书

支持临时凭证：是

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关其他信息，包括哪些 AWS 服务 适用于临时证书 AWS 服务 ，请参阅《IAM用户指南》IAM中的 “适用于临时证书”。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《用户指南》中的从IAM用户切换到IAM角色（控制台）。

您可以使用 AWS CLI 或手动创建临时证书 AWS API。然后，您可以使用这些临时证书进行访问 AWS。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅中的临时安全证书IAM。

InfluxDB 的 Timestream 的跨服务主体权限

支持转发访问会话 (FAS)：是

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，然后此操作在其他服务中启动另一个操作。FAS使用调用委托人的权限 AWS 服务以及 AWS 服务 向下游服务发出请求的请求。FAS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。有关提出FAS请求时的政策详情，请参阅转发访问会话。

InfluxDB 的 Timestream 的服务角色

支持服务角色：否

服务IAM角色是服务代替您执行操作的角色。IAM管理员可以在内部创建、修改和删除服务角色IAM。有关更多信息，请参阅《IAM用户指南》 AWS 服务中的创建角色以向委派权限。

> **Warning**
>
> 更改服务角色的权限可能会中断 InfluxDB 功能的时间流。只有在 InfluxDB 的 Timestream 提供相关指导时才编辑服务角色。

InfluxDB 的 Timestream 的服务相关角色

支持服务相关角色：是

服务相关角色是一种链接到的服务角色。 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。

有关创建或管理服务相关角色的详细信息，请参阅与之配合IAM使用的AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

适用于 InfluxDB 的 Amazon Timestream 的基于身份的策略示例

默认情况下，用户和角色无权为InfluxDB资源创建或修改Timestream。他们也无法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或来执行任务 AWS API。要授予用户对其所需资源执行操作的权限，IAM管理员可以创建IAM策略。然后，管理员可以将IAM策略添加到角色中，用户可以代入这些角色。

要了解如何使用这些示例策略文档创建IAM基于身份的JSON策略，请参阅IAM用户指南中的创建IAM策略（控制台）。

有关 Timestream 为 InfluxDB 定义的操作和资源类型（包括每种资源类型的格式）的详细信息，请参阅《ARNs服务授权参考》中的 Amazon Timestream for InfluxDB 的操作、资源和条件密钥。

主题

- 策略最佳实践
- 使用 InfluxDB 控制台的 Timestream
- 允许用户查看他们自己的权限
- 访问一个 Amazon S3 存储桶
- 允许所有操作
- 创建、描述、删除和更新数据库实例

策略最佳实践

基于身份的策略决定是否有人可以在你的账户中创建、访问或删除InfluxDB资源的 Timestream。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，请遵循以下指南和建议：

- 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AWS 管策略或工作职能托管策略。
- 应用最低权限权限-使用IAM策略设置权限时，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。有关使用应用权限IAM的更多信息，请参阅《IAM用户指南》IAM中的策略和权限。
- 使用IAM策略中的条件进一步限制访问权限-您可以在策略中添加条件以限制对操作和资源的访问权限。例如，您可以编写一个策略条件来指定所有请求都必须使用发送SSL。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IAMJSON策略元素：条件。
- 使用 A IAM ccess Analyzer 验证您的IAM策略以确保权限的安全性和功能性 — A IAM ccess Analyzer 会验证新的和现有的策略，以便策略符合IAM策略语言 (JSON) 和IAM最佳实践。IAMAccess Analyzer 提供了 100 多项策略检查和可行的建议，可帮助您制定安全和实用的策略。有关更多信息，请参阅《IAM用户指南》中的使用 A IAM ccess Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果您的场景需要IAM用户或 root 用户 AWS 账户，请打开MFA以提高安全性。要要求MFA何时调用API操作，请在策略中添加MFA条件。有关更多信息，请参阅《IAM用户指南》MFA中的使用进行安全API访问。

有关中最佳做法的更多信息IAM，请参阅《IAM用户指南》IAM中的安全最佳实践。

使用 InfluxDB 控制台的 Timestream

要访问适用于 InfluxDB 的 Amazon Timestream 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 InfluxDB 资源的时间流的详细信息。 AWS 账户如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅拨打 AWS CLI 或电话的用户，您无需为其设置最低控制台权限 AWS API。相反，只允许访问与他们尝试执行的API操作相匹配的操作。

为确保用户和角色仍然可以使用 InfluxDB 的 Timestream 控制台，还要将 InfluxDB 的时间流`ConsoleAccess`或`ReadOnly` AWS 托管策略附加到实体。有关更多信息，请参阅《用户指南》中的向IAM用户添加权限。

允许用户查看他们自己的权限

此示例说明如何创建允许IAM用户查看附加到其用户身份的内联和托管策略的策略。此策略包括在控制台上或使用或以编程方式完成此操作的 AWS CLI 权限。 AWS API

```
{
    "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": "*"
        }
    ]
}
```

访问一个 Amazon S3 存储桶

在本示例中，您想向 AWS 账户中的IAM用户授予访问您的 Amazon S3 存储桶的权限。examplebucket您还想要允许该用户添加、更新和删除对象。

除了授予该用户 s3:PutObject、s3:GetObject 和 s3:DeleteObject 权限外，此策略还授予 s3:ListAllMyBuckets、s3:GetBucketLocation 和 s3:ListBucket 权限。这些是控制台所需的其他权限。此外，s3:PutObjectAcl 和 s3:GetObjectAcl 操作需要能够在控制台中复制、剪切和粘贴对象。有关为用户授予权限并使用控制台测试用户的示例演练，请参阅示例演练：使用用户策略控制对桶的访问。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListBucketsInConsole",
         "Effect":"Allow",
         "Action":[
            "s3:ListAllMyBuckets"
         ],
         "Resource":"arn:aws:s3:::*"
      },
      {
         "Sid":"ViewSpecificBucketInfo",
         "Effect":"Allow",
         "Action":[
            "s3:ListBucket",
            "s3:GetBucketLocation"
         ],
         "Resource":"arn:aws:s3:::examplebucket"
      },
      {
```

```
         "Sid":"ManageBucketContents",
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:PutObjectAcl",
            "s3:GetObject",
            "s3:GetObjectAcl",
            "s3:DeleteObject"
         ],
         "Resource":"arn:aws:s3:::examplebucket/*"
      }
   ]
}
```

允许所有操作

以下是允许在 Timestream 中对 InfluxDB 进行所有操作的示例策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream-influxdb:*"
            ],
            "Resource": "*"
        }
    ]
}
```

创建、描述、删除和更新数据库实例

以下示例策略允许用户创建、描述、删除和更新数据库实例sampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream-influxdb:CreateDbInstance",
                "timestream-influxdb:GetDbInstance",
```

```
                "timestream-influxdb:DeleteDbInstance",
                "timestream-influxdb:UpdateDbInstance"
            ],
            "Resource": "arn:aws:timestream-influxdb:us-east-1:<account_ID>:dbinstance/
sampleDB"
        }
    ]
}
```

对 Amazon Timestream 的 InfluxDB 身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用适用于InfluxDB的Timestream时可能遇到的常见问题，以及. IAM

主题

- 我无权在 Timestream 中为 InfluxDB 执行操作
- 我想允许 AWS 账户之外的人访问我的 Timestream 以获取 InfluxDB 资源

我无权在 Timestream 中为 InfluxDB 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` 用户尝试使用控制台查看有关虚构 *`my-example-widget`* 资源的详细信息，但不拥有虚构 `timestream-influxdb:`*`GetWidget`* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 timestream-influxdb:GetWidget on resource: my-example-widget
```

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 `timestream-influxdb:`*`GetWidget`* 操作访问 *`my-example-widget`* 资源。

我想允许 AWS 账户之外的人访问我的 Timestream 以获取 InfluxDB 资源

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 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 控制对中数据库实例的访问权限 VPC
- 要了解 InfluxDB 的 Timestream 是否支持这些功能，请参阅适用于 InfluxDB 的 Amazon Timestream 是如何使用的。IAM
- 要了解如何通过您拥有的 AWS 账户提供对资源的访问权限，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。
- 要了解如何向第三方 AWS 账户提供对您的资源的访问权限，请参阅《IAM用户指南》中的向第三方 AWS 账户提供访问权限。
- 要了解如何通过联合身份验证提供访问权限，请参阅《用户指南》中的向经过外部身份验证的用户提供访问权限（联合身份验证）。IAM
- 要了解使用角色和基于资源的策略进行跨账户访问的区别，请参阅用户指南中的IAM角色与基于资源的策略的IAM区别。

控制对中数据库实例的访问权限 VPC

使用亚马逊虚拟私有云（亚马逊VPC），您可以将 AWS 资源（例如适用于InfluxDB数据库实例的 Amazon Timestream）启动到虚拟私有云（ ）中。VPC当您使用 Amazon 时VPC，您可以控制自己的虚拟网络环境。您可以选择自己的 IP 地址范围、创建子网以及配置路由和访问控制列表。

VPC安全组控制对内部数据库实例的访问权限VPC。每条VPC安全组规则都允许特定来源访问与VPC该VPC安全组关联的数据库实例。源可以是一系列地址（例如 203.0.113.0/24），也可以是其他安全组。VPC通过将VPC安全组指定为源，可以允许来自使用该源VPC安全组的所有实例（通常是应用程序服务器）的传入流量。在尝试连接到您的数据库实例之前，请根据您的VPC用例进行配置。以下是访问中数据库实例的常见场景VPC：

中的一个数据库实例，由同一个 Amazon EC2 实例VPC访问 VPC

中数据库实例的常见用途VPC是与在同一EC2实例中运行的应用程序服务器共享数据VPC。该EC2实例可能运行带有与数据库实例交互的应用程序的 Web 服务器。

中的数据库实例由不同EC2实例中的实例VPC访问 VPC

在某些情况下，您的数据库实例与您用于访问它的EC2实例VPC不同。如果是，则可以使用VPC对等互连来访问数据库实例。

中的数据库实例，由客户端应用程序通过 Internet VPC 访问

要通过 Internet VPC 从客户端应用程序访问中的数据库实例，您需要使用单个公VPC有子网配置并使用公有子网创建数据库实例。您还可以在中配置互联网网关，VPC以启用通过互联网进行通信。

要从数据库实例外部连接到数据库实例VPC，该数据库实例必须可公开访问。此外，必须使用数据库实例安全组的入站规则授予访问权限，并且必须满足其他要求。

有关VPC安全组的更多信息，请参阅 Amazon Virtual Private Cloud 用户指南中的安全组。

有关如何连接到 InfluxDB 数据库实例的 Timestream 的详细信息，请参阅。连接适用于 InfluxDB 数据库实例的 Amazon Timestream

安全组情况

中数据库实例的常见用途VPC是与在同一 Amazon EC2 实例中运行的应用程序服务器共享数据VPC，外部的客户端应用程序可以访问这些数据VPC。在这种情况下，您可以使用InfluxDB的时间流和InfluxDB的时间流上的VPC页面以及EC2API操作来创建必要的实例和安全组： AWS Management Console

1. 创建VPC安全组（例如sg-0123ec2example）并定义使用客户端应用程序的 IP 地址作为源的入站规则。此安全组允许您的客户端应用程序连接到使用此安全组VPC的EC2实例。
2. 为应用程序创建EC2实例，并将该EC2实例添加到您在上一步中创建VPC的安全组(sg-0123ec2example)。
3. 创建第二个VPC安全组（例如sg-6789rdsexample），并通过将您在步骤 1 (sg-0123ec2example) 中创建VPC的安全组指定为源来创建新规则。
4. 创建新的数据库实例，并将该数据库实例添加到您在上一步中创建VPC的安全组(sg-6789rdsexample)。创建数据库时，请使用与您在步骤 3 中创建VPC的安全组(sg-6789rdsexample) 规则中指定的端口号相同的端口号。

创建VPC安全组

您可以使用VPC控制台为数据库实例创建VPC安全组。有关创建安全组的信息，请参阅 Amazon Virtual Private Cloud 用户指南中的安全组。

将安全组与数据库实例关联

您可以使用适用于 InfluxDB 的 Timestream 控制台上的 “更新”、InfluxDB 的 Timestre UpdateDBInstance am 或命令，将安全组与数据库实例关联。API update-db-instance AWS CLI

以下CLI示例关联特定的VPC安全组并从数据库实例中移除数据库安全组

```
aws timestream-influxdb update-db-instance --identifier dbName --vpc-security-group-
ids sg-ID
```

有关修改数据库实例的信息，请参阅更新数据库实例。

使用适用于 InfluxDB 的 Amazon Timestream 的服务相关角色

适用于 InfluxDB 的 Amazon Timestream 使用 AWS Identity and Access Management () 服务相关角色IAM。服务相关角色是一种直接链接到 AWS 服务的独特IAM角色，例如适用于 InfluxDB 的 Amazon Timestream。InfluxDB 服务相关角色的亚马逊 Timestream 是由亚马逊 Timestream 为 InfluxDB 预定义的。它们包括服务代表您的数据库实例调用 AWS 服务所需的所有权限。

服务相关角色可以更轻松地为 InfluxDB 设置 Amazon Timestream，因为您不必手动添加必要的权限。这些角色已存在于您的 AWS 账户中，但已关联到 Amazon Timestream，适用于 InfluxDB 用例，并且具有预定义的权限。只有适用于 InfluxDB 的 Amazon Timestream 可以担任这些角色，并且只有这些角色才能使用预定义的权限策略。只有先删除角色的相关资源，才能删除角色。这可以保护你的 Amazon Timestream 上的 InfluxDB 资源，因为你不能无意中删除访问这些资源的必要权限。

有关支持服务相关角色的其他服务的信息，请参阅与之配合使用的AWS 服务，IAM并在 “服务相关角色” 列中查找标有 “是” 的服务。选择是和链接，查看该服务的服务相关角色文档。

目录

适用于 InfluxDB 的 Amazon Timestream 的服务相关角色权限

适用于 InfluxDB 的 Amazon Timestream 使用名为的服务相关角色 AmazonTimestreamInfluxDBServiceRolePolicy— 该策略允许 InfluxDB 的 Timestream 在必要时代表您管理管理 AWS 集群的资源。

AmazonTimestreamInfluxDBServiceRolePolicy服务相关角色权限策略允许适用于 InfluxDB 的 Amazon Timestream 在指定资源上完成以下操作：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "DescribeNetworkStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:DescribeSubnets",
    "ec2:DescribeVpcs",
    "ec2:DescribeNetworkInterfaces"
   ],
   "Resource": "*"
  },
  {
   "Sid": "CreateEniInSubnetStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterface"
   ],
   "Resource": [
    "arn:aws:ec2:*:*:subnet/*",
    "arn:aws:ec2:*:*:security-group/*"
   ]
  },
  {
   "Sid": "CreateEniStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterface"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "Null": {
     "aws:RequestTag/AmazonTimestreamInfluxDBManaged": "false"
```

```
    }
   }
  },
  {
   "Sid": "CreateTagWithEniStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateTags"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "Null": {
     "aws:RequestTag/AmazonTimestreamInfluxDBManaged": "false"
    },
    "StringEquals": {
     "ec2:CreateAction": [
      "CreateNetworkInterface"
     ]
    }
   }
  },
  {
   "Sid": "ManageEniStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterfacePermission",
    "ec2:DeleteNetworkInterface"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "Null": {
     "aws:ResourceTag/AmazonTimestreamInfluxDBManaged": "false"
    }
   }
  },
  {
   "Sid": "PutCloudWatchMetricsStatement",
   "Effect": "Allow",
   "Action": [
    "cloudwatch:PutMetricData"
   ],
   "Condition": {
    "StringEquals": {
     "cloudwatch:namespace": [
```

```
      "AWS/Timestream/InfluxDB",
      "AWS/Usage"
     ]
    }
   },
   "Resource": [
    "*"
   ]
  },
  {
   "Sid": "ManageSecretStatement",
   "Effect": "Allow",
   "Action": [
    "secretsmanager:CreateSecret",
    "secretsmanager:DeleteSecret"
   ],
   "Resource": [
    "arn:aws:secretsmanager:*:*:secret:READONLY-InfluxDB-auth-parameters-*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  }
 ]
}
```

允许IAM实体创建 AmazonTimestreamInfluxDBServiceRolePolicy服务相关角色

将以下策略声明添加到该IAM实体的权限中：

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole",
        "iam:PutRolePolicy"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/
timestreamforinfluxdb.amazonaws.com/AmazonTimestreamInfluxDBServiceRolePolicy*",
    "Condition": {"StringLike": {"iam:AWS ServiceName":
 "timestreamforinfluxdb.amazonaws.com"}}
}
```

允许IAM实体删除 AmazonTimestreamInfluxDBServiceRolePolicy服务相关角色

将以下策略声明添加到该IAM实体的权限中：

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/
timestreamforinfluxdb.amazonaws.com/AmazonTimestreamInfluxDBServiceRolePolicy*",
    "Condition": {"StringLike": {"iam:AWS ServiceName":
 "timestreamforinfluxdb.amazonaws.com"}}
}
```

或者，您可以使用 AWS 托管策略为InfluxDB提供对Amazon Timestream的完全访问权限。

创建服务相关角色 () IAM

您无需手动创建服务相关角色。当你创建数据库实例时，适用于 InfluxDB 的 Amazon Timestream 会为你创建服务相关角色。

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。当您创建数据库实例时，适用于 InfluxDB 的 Amazon Timestream 会再次为您创建服务相关角色。

编辑适用于 InfluxDB 的 Amazon Timestream 的服务相关角色的描述

适用于 InfluxDB 的 Amazon Timestream 不允许你编辑服务相关角色。 AmazonTimestreamInflux DBServiceRolePolicy创建服务相关角色后，将无法更改角色名称，因为可能有多个实体引用该角色。但是，您可以使用编辑角色的描述IAM。

编辑服务相关角色描述（IAM控制台）

您可以使用IAM控制台编辑与服务相关的角色描述。

编辑服务相关角色的描述（控制台）

1. 在IAM控制台的左侧导航窗格中，选择角色。
2. 以下代码示例显示如何将 IAM 策略附加到用户。
3. 在 Role description 的最右侧，选择 Edit。
4. 在框中输入新描述，然后选择 Save（保存）。

编辑服务相关角色描述 () IAM CLI

您可以使用中的IAM操作 AWS Command Line Interface 来编辑与服务相关的角色描述。

更改服务相关角色的描述 () CLI

1. （可选）要查看角色的当前描述，请使用 f AWS CLI or IAM 操作get-role。

   Example

   ```
   $ aws iam get-role --role-name AmazonTimestreamInfluxDBServiceRolePolicy
   ```

   使用角色名称（而不是ARN）来指代CLI操作中的角色。例如，如果角色具有以下内容ARN:`arn:aws:iam::123456789012:role/myrole`，则将该角色称为**myrole**。

2. 要更新服务相关角色的描述，请使用 fo AWS CLI r IAM 操作update-role-description。

   Linux 和 macOS

   ```
   $ aws iam update-role-description \
       --role-name AmazonTimestreamInfluxDBServiceRolePolicy \
       --description "new description"
   ```

   Windows

   ```
   $ aws iam update-role-description ^
       --role-name AmazonTimestreamInfluxDBServiceRolePolicy ^
       --description "new description"
   ```

编辑服务相关角色描述 () IAM API

您可以使用IAMAPI来编辑与服务相关的角色描述。

更改服务相关角色的描述 () API

1. （可选）要查看角色的当前描述，请使用IAMAPI操作 GetRole.

   Example

   ```
   https://iam.amazonaws.com/
      ?Action=GetRole
   ```

```
   &RoleName=AmazonTimestreamInfluxDBServiceRolePolicy
   &Version=2010-05-08
   &AUTHPARAMS
```

2. 要更新角色的描述，请使用IAMAPI操作 UpdateRoleDescription.

   Example

```
https://iam.amazonaws.com/
   ?Action=UpdateRoleDescription
   &RoleName=AmazonTimestreamInfluxDBServiceRolePolicy
   &Version=2010-05-08
   &Description="New description"
```

删除适用于 InfluxDB 的 Amazon Timestream 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功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除您的服务相关角色，然后才能将其删除。

适用于 InfluxDB 的 Amazon Timestream 不会为您删除服务相关角色。

清除服务相关角色

在使用IAM删除服务相关角色之前，请先确认该角色没有与之关联的资源（集群）。

在控制台中检查服务相关角色是否有活动会IAM话

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控制台的左侧导航窗格中，选择角色。然后选择AmazonTimestreamInfluxDBServiceRolePolicy角色的名称（不是复选框）。
3. 在所选角色的 Summary 页面上，选择 Access Advisor 选项卡。
4. 在访问顾问选项卡查看服务相关角色的近期活动。

删除服务相关角色（IAM控制台）

您可以使用IAM控制台删除服务相关角色。

删除服务相关角色 (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IAM控制台的左侧导航窗格中，选择角色。然后，选中要删除的角色名称旁边的复选框，而不是名称或行本身。
3. 对于页面顶部的角色操作，请选择删除角色。
4. 在确认页面中，查看上次访问服务的数据，该数据显示了每个选定角色上次访问 AWS 服务的时间。这样可帮助您确认角色当前是否处于活动状态。如果要继续，请选择 Yes, Delete 以提交服务相关角色进行删除。
5. 观看IAM控制台通知，监控服务相关角色删除的进度。由于IAM服务相关角色的删除是异步的，因此在您提交要删除的角色之后，删除任务可能会成功或失败。如果任务失败，您可以从通知中选择 View details 或 View Resources 以了解删除失败的原因。

删除服务相关角色 () IAM CLI

您可以使用中的IAM操作 AWS Command Line Interface 来删除服务相关角色。

删除服务相关角色 () CLI

1. 如果您不知道要删除的服务相关角色的名称，请输入以下命令。此命令列出了您账户中的角色及其 Amazon 资源名称 (ARNs)。

   ```
   $ aws iam get-role --role-name role-name
   ```

   使用角色名称（而不是ARN）来指代CLI操作中的角色。例如，如果某个角色具有ARN`arn:aws:iam::123456789012:role/myrole`，则将该角色称为**`myrole`**。
2. 如果服务相关角色正被使用或具有关联的资源，则无法删除它，因此您必须提交删除请求。如果不满足这些条件，该请求可能会被拒绝。您必须从响应中捕获 `deletion-task-id` 以检查删除任务的状态。输入以下命令以提交服务相关角色的删除请求。

   ```
   $ aws iam delete-service-linked-role --role-name role-name
   ```
3. 输入以下命令以检查删除任务的状态。

   ```
   $ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
   ```

删除任务的状态可能是 NOT_STARTED、IN_PROGRESS、SUCCEEDED 或 FAILED。如果删除失败，则调用会返回失败的原因，以便您进行问题排查。

删除服务相关角色 () IAM API

您可以使用IAMAPI来删除服务相关角色。

删除服务相关角色 () API

1. 要提交与服务相关的名册的删除请求，请致电 DeleteServiceLinkedRole。 在请求中，指定角色名称。

   如果服务相关角色正被使用或具有关联的资源，则无法删除它，因此您必须提交删除请求。如果不满足这些条件，该请求可能会被拒绝。您必须从响应中捕获 DeletionTaskId 以检查删除任务的状态。
2. 要查看删除状态，请致电 GetServiceLinkedRoleDeletionStatus。 在请求中，指定DeletionTaskId。

   删除任务的状态可能是 NOT_STARTED、IN_PROGRESS、SUCCEEDED 或 FAILED。如果删除失败，则调用会返回失败的原因，以便您进行问题排查。

InfluxDB 服务相关角色的 Amazon Timestream 支持的区域

适用于 InfluxDB 的 Amazon Timestream 支持在提供服务的所有地区使用服务相关角色。有关更多信息，请参阅AWS 服务端点。

AWS 适用于 InfluxDB 的 Amazon Timestream 的托管策略

要向用户、群组和角色添加权限，使用 AWS 托管策略比自己编写策略要容易得多。创建IAM客户托管策略以仅向您的团队提供他们所需的权限需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些政策涵盖常见用例，可在您的 AWS 账户中使用。有关 AWS 托管策略的更多信息，请参阅《IAM用户指南》中的AWS 托管策略。

AWS 服务维护和更新 AWS 托管策略。您无法更改 AWS 托管策略中的权限。服务偶尔会向 AWS 管理型策略添加额外权限以支持新特征。此类更新会影响附加策略的所有身份（用户、组和角色）。当启动新特征或新操作可用时，服务最有可能会更新 AWS 管理型策略。服务不会从 AWS 托管策略中移除权限，因此策略更新不会破坏您的现有权限。

此外，还 AWS 支持跨多个服务的工作职能的托管策略。例如，ReadOnlyAccess AWS 托管策略提供对所有 AWS 服务和资源的只读访问权限。当服务启动一项新功能时， AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅《IAM用户指南》中的工作职能AWS 托管策略。

AWS 托管策略： AmazonTimestreamInfluxDBServiceRolePolicy

您不能将 AmazonTimestreamInfluxDBServiceRolePolicy AWS 托管策略附加到账户中的身份。此策略是 AWS TimestreamforInflux数据库服务相关角色的一部分。此角色允许服务管理您账户中的网络接口和安全组。

InfluxDB 的 Timestream 使用此策略中的权限来管理EC2安全组和网络接口。这是管理 InfluxDB 数据库实例的时间流所必需的。

要以JSON格式查看此政策，请参阅AmazonTimestreamInfluxDBServiceRolePolicy。

AWS-适用于 InfluxDB 的 Amazon Timestream 的托管策略

AWS 通过提供由创建和管理的独立IAM策略来解决许多常见用例 AWS。托管式策略可授予常用案例的必要权限，因此，您可以免去调查都需要哪些权限的工作。有关更多信息，请参阅《IAM用户指南》中的AWS 托管策略。

以下 AWS 托管策略仅适用于 InfluxDB 的 Timestream，您可以将其附加到账户中的用户：

AmazonTimestreamInfluxDBFullAccess

您可以将该`AmazonTimestreamInfluxDBFullAccess`策略附加到您的IAM身份。此策略授予管理权限，允许对所有 Timestream 的 InfluxDB 资源进行完全访问权限。

您也可以创建自己的自定义IAM策略，以授予亚马逊 Timestream 对 Influ API xDB 操作的权限。您可以将这些自定义策略附加到需要这些权限的IAM用户或群组。

要以JSON格式查看此政策，请参阅AmazonTimestreamInfluxDBFullAccess。

InfluxDB 更新托管策略的时间流 AWS

查看自该服务开始跟踪这些更改以来，InfluxDB Timestream AWS 托管策略更新的详细信息。要获得有关此页面更改的自动提醒，请在 InfluxDB 文档历史记录页面上订阅 Tim RSS estream 上的提要。

更改	描述	日期
AmazonTimestreamInfluxDBFullAccess – 对现有策略的更新	已将`ec2:DescribeRouteTables` 操作添加到现有`AmazonTimestreamInfluxDBFullAccess` 托管策略中。此操作用于描述您的路由表	10/08/2024
AWS 托管策略： AmazonTimestreamInfluxDBServiceRolePolicy：新策略	适用于 InfluxDB 的 Amazon Timestream 添加了一项新策略，允许该服务管理您账户中的网络接口和安全组。	03/14/2024
AmazonTimestreamInfluxDBFullAccess：新策略	适用于 InfluxDB 的 Amazon Timestream 添加了一项新策略，为创建、更新、删除和列出亚马逊 Timestream InfluxDB 实例以及创建和列出参数组提供完全的管理权限。	03/14/2024

通过端点连接到 InfluxDB 的时间流 VPC

你可以通过虚拟私有云中的私有接口端点直接连接到 InfluxDB 的 Timestream ()。VPC当你使用接口VPC端点时，你VPC和 InfluxDB 的 Timestream 之间的通信完全在网络内进行。 AWS

InfluxDB 的 Timestream 支持由提供支持的亚马逊虚拟私有云（亚马逊VPC）终端节点。AWS PrivateLink每个VPC端点都由一个或多个弹性网络接口 (ENIs) 表示，您的VPC子网中有私有 IP 地址。

接口VPC端点将您VPC直接连接到 InfluxDB 的 Timestream，无需互联网网关、NAT设备、VPN连接或连接。 AWS Direct Connect 你中的实例VPC不需要公有 IP 地址即可与 InfluxDB 的 Timestream 通信。

区域

InfluxDB 的 Timestream 支持VPC所有支持 InfluxDB 时间流 AWS 区域 的VPC端点和端点策略。

主题

InfluxDB 端点的时间流VPC注意事项

在为 InfluxDB 的 Timestream 设置接口VPC端点之前，请查看指南中的接口端点属性和限制主题。AWS PrivateLink

InfluxDB 对VPC端点的支持的时间流包括以下内容。

- 你可以使用你的VPC终端节点从你的端点调用所有 Timestream 进行 InfluxDB API 操作。VPC
- 您可以使用 AWS CloudTrail 日志通过端点审核您对 InfluxDB 资源的 Timestream 使用情况。VPC有关详细信息，请参阅记录您的VPC终端节点。

为 InfluxD VPC B 的 Timestream 创建端点

你可以使用亚马逊VPC控制台或亚马逊为 InfluxDB 的 Timestream 创建VPC终端节点。VPC API有关更多信息，请参阅《AWS PrivateLink 指南》中的创建接口端点。

- 要为 InfluxDB 的 Timestream 创建VPC端点，请使用以下服务名称：

  ```
  com.amazonaws.region.timestream-influxdb
  ```

  例如，在美国西部（俄勒冈）区域 (us-west-2)，服务名称为：

  ```
  com.amazonaws.us-west-2.timestream-influxdb
  ```

为了便于使用VPC终端节点，您可以为终VPC端节点启用私有DNS名称。如果您选择启用DNS名称选项，InfluxDB DNS 主机名的标准时间流将解析到您的端点。VPC例如，`https://`

`timestream-influxdb.us-west-2.amazonaws.com`将解析为连接到服务名称的VPC端点`com.amazonaws.us-west-2.timestream-influxdb`。

使用此选项可以更轻松地使用VPC端点。默认情况下， AWS SDKs和 AWS CLI 使用标准的 Timestream 作为 InfluxDB DNS 主机名，因此您无需URL在应用程序和命令中指定VPC端点。

有关更多信息，请参阅 AWS PrivateLink 指南中的通过接口端点访问服务。

连接到 Inf VPC luxDB 端点的时间流

你可以使用或通过VPC端点连接到 InfluxDB 的 Timestream。 AWS SDK AWS CLI AWS Tools for PowerShell要指定VPC端点，请使用其DNS名称。

如果您在创建VPC终端节点时启用了私有主机名，则无需在CLI命令或应用程序配置URL中指定VPC终端节点。InfluxDB DNS 主机名的标准时间流解析到您的端点。VPC AWS CLI 和默认SDKs使用此主机名，因此您可以开始使用该端点连接到 InfluxDB 区域VPC端点的 Timestream，而无需更改脚本和应用程序中的任何内容。

要使用私有主机名，您的`enableDnsHostnames`和`enableDnsSupport`属性VPC必须设置为。`true`要设置这些属性，请使用ModifyVpcAttribute操作。有关详情，请参阅《Amazon VPC 用户指南》VPC中的查看和更新您的DNS属性。

控制对VPC终端节点的访问

要控制 InfluxDB Timestream 对VPC终端节点的访问权限，请将VPC终端节点策略附加到您的终端节点。VPC端点策略决定委托人是否可以使用VPC端点在 Timestream 上调用 Timestream 对 InfluxDB 资源进行 InfluxDB 操作。

您可以在创建VPC终端节点时创建终端节点策略，也可以随时更改VPC终端节点策略。使用VPC管理控制台或CreateVpcEndpoint或ModifyVpcEndpoint操作。您也可以使用 AWS CloudFormation 模板创建和更改VPC终端节点策略。有关使用VPC管理控制台的帮助，请参阅AWS PrivateLink 指南中的创建接口终端节点和修改接口终端节点。

> **Note**
>
> InfluxDB 的 Timestream 支持从 2020 年 7 月开始的VPC端点策略。VPC在该日期之前创建的 InfluxDB Timestream 端点具有默认的VPC端点策略，但您可以随时对其进行更改。

主题

- 关于VPC终端节点策略
- 默认VPC终端节点策略
- 创建VPC终端节点策略
- 查看VPC终端节点策略

关于VPC终端节点策略

要使使用VPC端点的 Timestream for InfluxDB 请求成功，委托人需要来自两个来源的权限：

- IAM策略必须授予委托人对资源调用操作的权限。
- VPC终端节点策略必须授予委托人使用终端节点发出请求的权限。

默认VPC终端节点策略

每个VPC端点都有VPC终端节点策略，但您无需指定策略。如果未指定策略，则默认的终端节点策略允许所有委托人对终端节点上的所有资源执行所有操作。

但是，对于 InfluxDB 资源的 Timestream，委托人还必须拥有从IAM策略调用操作的权限。因此，实际上，默认策略规定，如果委托人有权对资源调用操作，他们也可以使用端点来调用该操作。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Principal": "*",
      "Resource": "*"
    }
  ]
}
```

要允许委托人仅将VPC终端节点用于其允许的部分操作，请创建或更新VPC终端节点策略。

创建VPC终端节点策略

VPC终端节点策略决定委托人是否有权使用VPC终端节点对资源执行操作。对于 InfluxDB 资源的 Timestream，委托人还必须有权根据策略执行操作，IAM

每个VPC端点策略声明都需要以下元素：

- 可执行操作的主体
- 可执行的操作
- 可对其执行操作的资源

策略声明未指定VPC终端节点。相反，它适用于该策略所关联的任何VPC终端节点。有关更多信息，请参阅 Amazon VPC 用户指南中的使用VPC终端节点控制对服务的访问。

AWS CloudTrail 记录使用该VPC终端节点的所有操作。

查看VPC终端节点策略

要查看VPC终端节点的终端节点策略，请使用VPC管理控制台或DescribeVpcEndpoints操作。

以下 AWS CLI 命令获取具有指定终端节点 ID 的终VPC端节点的策略。

在使用此命令之前，请将示例终端节点 ID 替换为您账户中的有效终端节点 ID。

```
$ aws ec2 describe-vpc-endpoints \
--query 'VpcEndpoints[?VpcEndpointId==`vpc-endpoint-id`].[PolicyDocument]'
--output text
```

在策略声明中使用VPC终端节点

当请求来自VPC或使用端点时，您可以控制对 InfluxDB 资源和操作的 Timestream 访问权限。VPC为此，请在IAM策略中使用以下全局条件密钥之一。

- 使用`aws:sourceVpce`条件密钥根据VPC端点授予或限制访问权限。
- 使用`aws:sourceVpc`条件密钥根据托管私有终端节点的VPC权限来授予或限制访问权限。

> **Note**
>
> 根据您的VPC终端节点创建密钥策略和IAM策略时要谨慎行事。如果策略声明要求请求来自特定VPC或VPC端点，则代表您使用 Timestream for InfluxDB 资源的集成 AWS 服务发出的请求可能会失败。
>
> 此外，当请求来自 Amazon VPC 终端节点时，`aws:sourceIP`条件密钥无效。要限制对VPC终端节点的请求，请使用`aws:sourceVpce`或`aws:sourceVpc`条件键。有关更多信息，请参阅AWS PrivateLink 指南中的VPC终端节点和终端VPC节点服务的身份和访问管理。

您可以使用这些全局条件键来控制对此类操作的访问权限 CreateDbInstance，这些操作不依赖于任何特定资源。

记录您的VPC终端节点

AWS CloudTrail 记录使用该VPC终端节点的所有操作。当向 Timestream for InfluxDB 发出的请求使用VPC端点时，VPC端点 ID 会出现在记录该请求的AWS CloudTrail 日志条目中。您可以使用端点 ID 来审计 Influx VPC DB 端点的 Timestream 使用情况。

但是，您的 CloudTrail 日志不包括其他账户中的委托人请求的操作，也不包括Timestream对InfluxDB资源和其他账户中的别名进行InfluxDB操作的请求。此外，为了保护您的安全VPC，被VPC终端节点策略拒绝但本来会被允许的请求不会记录在中AWS CloudTrail。

在 InfluxDB 的 Timestream 中记录和监控

监控是维护 InfluxDB 的 Timestream 及其解决方案的可靠性、可用性和性能的重要组成部分。 AWS 您应该从 AWS 解决方案的所有部分收集监控数据，以便在出现多点故障时可以更轻松地对其进行调试。但是，在开始监控 InfluxDB 的 Timestream 之前，您应该创建一个包含以下问题的答案的监控计划：

- 监控目的是什么？
- 您将监控哪些资源？
- 监控这些资源的频率如何？
- 您将使用哪些监控工具？
- 谁负责执行监控任务？
- 出现错误时应通知谁？

下一步是通过测量不同时间和不同负载条件下的性能，为环境中InfluxDB性能的正常时间流建立基准。在监控 InfluxDB 的 Timestream 时，请存储历史监控数据，以便可以将其与当前性能数据进行比较，识别正常的性能模式和性能异常，并设计解决问题的方法。

要建立基准，您至少应监控以下各项：

- 系统错误，以便您可以确定是否有任何请求导致了错误。

主题

- 监控工具

- 使用记录 InfluxDB 调API用的时间流 AWS CloudTrail

监控工具

AWS 提供了各种工具，您可以使用这些工具来监控 InfluxDB 的时间流。您可以配置其中的一些工具来为您执行监控任务，但有些工具需要手动干预。建议您尽可能实现监控任务自动化。

主题

- 自动监控工具
- 手动监控工具

自动监控工具

你可以使用以下自动监控工具来监视 InfluxDB 的 Timestream，并在出现问题时进行报告：

- A CloudWatch mazon Alarms — 在您指定的时间段内观察单个指标，并根据该指标在多个时间段内相对于给定阈值的值执行一项或多项操作。该操作是向亚马逊简单通知服务 (亚马逊SNS) 主题或 Amazon A EC2 uto Scaling 策略发送的通知。 CloudWatch 警报不会仅仅因为它们处于特定状态就调用操作；该状态必须已更改并保持了指定的时间段。有关更多信息，请参阅 使用 Amazon 进行监控 CloudWatch。

手动监控工具

监控 InfluxDB Timestream 的另一个重要部分是手动监视 CloudWatch 警报未涵盖的项目。InfluxDB、CloudWatch Trusted Advisor、和其他 AWS Management Console 仪表板的 Timestream 提供了 at-a-glance环境状态的视图。 AWS

- CloudWatch 主页显示以下内容：
  - 当前告警和状态
  - 告警和资源图表
  - 服务运行状况

  此外，您还可以使用 CloudWatch 执行以下操作：
  - 创建自定义控制面板以监控您关心的服务
  - 绘制指标数据图，以排除问题并弄清楚趋势
  - 搜索和浏览您的所有 AWS 资源指标

- 创建和编辑告警以接收问题通知

使用记录 InfluxDB 调API用的时间流 AWS CloudTrail

InfluxDB 的 Timestream 与一项服务集成，该服务提供用户 AWS CloudTrail、角色或 AWS 服务在 InfluxDB 的 Timestream 中采取的操作的记录。 CloudTrail 将 InfluxDB 的时间流的数据定义语言 (DDL) API 调用捕获为事件。捕获的调用包括来自 InfluxDB 控制台的 Timestream 调用和对 InfluxDB 操作的 Timestream 的代码调用。API如果您创建跟踪，则可以允许将 CloudTrail事件持续传输到亚马逊简单存储服务 (Amazon S3) Service 存储桶，包括适用于 InfluxDB 的 Timestream 的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 主机上的事件历史记录中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向 Timestream for InfluxDB 发出的请求、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅AWS CloudTrail 用户指南。

InfluxDB 信息的时间流 CloudTrail

CloudTrail 在您创建 AWS 账户时已在您的账户上启用。当 InfluxDB 的 Timestream 中发生活动时，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。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，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录 AWS 账户中的事件，包括 InfluxDB Timestream 的事件，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，当您在控制台中创建跟踪时，该跟踪将应用于所有AWS区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。

有关更多信息，请参阅《AWS CloudTrail 用户指南》中的以下主题：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 为以下各项配置亚马逊SNS通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件
- 从多个账户接收 CloudTrail 日志文件
- 记录数据事件

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用 root 还是 AWS Identity and Access Management (IAM) 用户凭证发出
- 请求是使用角色还是联合用户的临时安全凭证发出的
- 请求是否由其他 AWS 服务发出

有关更多信息，请参阅CloudTrail userIdentity元素。

适用于 InfluxDB 的 Amazon Timestream 的合规性验证

作为多个合规计划的一部分，第三方审计师评估适用于InfluxDB的Amazon Timestream的安全性和合规性。 AWS 这些功能包括：

- GDPR
- HIPAA
- PCI
- SOC

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全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，并提供了部署以安全性和合规性为重点 AWS 的基准环境的步骤。
- 在 Amazon Web Services 上进行HIPAA安全与合规架构 — 本白皮书描述了各公司如何使用 AWS 来创建HIPAA符合条件的应用程序。

> **Note**
>
> 并非所有 AWS 服务 人都有HIPAA资格。有关更多信息，请参阅《HIPAA合格服务参考》。

- AWS 合AWS 规资源 — 此工作簿和指南集合可能适用于您的行业和所在地区。

- 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的最佳实践， AWS 服务并将指南映射到跨多个框架（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)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- AWS Security Hub— 这 AWS 服务 可以全面了解您的安全状态 AWS。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践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务 检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 可以帮助您满足各种合规性要求 PCIDSS，例如满足某些合规性框架规定的入侵检测要求。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

适用于 InfluxDB 的 Amazon Timestream 中的弹性

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。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失效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。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

适用于 InfluxDB 的 Amazon Timestream 会定期进行内部备份并将其保留 24 小时，以支持可用性和耐久性。快照是在删除期间拍摄的，并保留 30 天以支持恢复。要访问或使用这些内容，请向AWS 支持部门提交工单。

您可以创建具有多可用区恢复功能的实例。有关更多信息，请参阅多可用区数据库实例部署。

适用于 InfluxDB 的 Amazon Timestream 中的基础设施安全

作为一项托管服务，适用于InfluxDB的Amazon Timestream受亚马逊网络服务：安全流程概述白皮书中描述的 AWS 全球网络安全程序的保护。

您可以使用 AWS 已发布的控制平面API调用通过网络访问 InfluxDB 的 Timestream。有关更多信息，请参阅控制平面和数据平面。客户端必须支持传输层安全 (TLS) 1.2 或更高版本。我们推荐使用 TLS 1.2 或 1.3。客户端还必须支持具有完全向前保密性的密码套件 ()，例如 Ephemeral Diffie-Hellman

(PFS) 或 Elliptic Curve Ephemeral Diffie-Hellman ()。DHE ECDHE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委托人关联的私有访问密钥对请求进行签名。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。

InfluxDB 的时间流经过精心设计，因此您的流量可以隔离到您的 InfluxDB 实例的时间流所在的特定 AWS 区域。

安全组

安全组控制着流量在数据库实例内外拥有的访问权限。默认情况下，数据库实例的网络访问处于关闭状态。您可以在安全组中指定规则，允许从 IP 地址范围、端口或安全组进行访问。配置传入规则后，会向与该安全组关联的所有数据库实例应用相同的规则。

有关更多信息，请参阅 控制对中数据库实例的访问权限 VPC。

InfluxDB 的 Timestream 中的配置和漏洞分析

配置和 IT 控制由您（我们的客户）共同 AWS 负责。有关更多信息，请参阅责任 AWS 共担模型。除了分担责任模型外，InfluxDB用户Timestream还应注意以下几点：

- 客户有责任使用相关的客户端依赖项修补其客户端应用程序。
- 如果合适，客户应考虑进行渗透测试（参见https://aws.amazon.com/security/渗透测试/。）

InfluxDB 的 Timestream 中的事件响应

InfluxDB 服务事件的 Amazon Timestream 在 Personal Health Dashboard 中报告。您可以在此处了解有关仪表板的 AWS Health 更多信息。

InfluxDB 的时间流支持使用进行报告。 AWS CloudTrail有关更多信息，请参阅 使用记录 InfluxDB 调API用的时间流 AWS CloudTrail。

适用于 InfluxDB API 和接口终端节点的 Amazon Timestream () VPC AWS PrivateLink

您可以通过创建接口终端节点在您VPC和亚马逊 Amazon Timestream 之间为 InfluxDB 控制平面API终端节点建立私有连接。VPC接口端点由提供支持AWS PrivateLink。 AWS PrivateLink 允许您在没有互联网网关、NAT设备、VPN连接或 Direct Connect 连接的情况下私下访问 Amazon Timestre AWS am 以进行 InfluxDB API 操作。

您中的实例VPC不需要公有 IP 地址即可与 Influ API xDB 终端节点的 Amazon Timestream 通信。您的实例也不需要公有 IP 地址即可使用任何可用的时间流进行 InfluxD API B 操作。你VPC和亚马逊 Timestream for InfluxDB 之间的流量不会离开亚马逊网络。每个接口端点均由子网中的一个或多个弹性网络接口表示。有关弹性网络接口的更多信息，请参阅 Amazon EC2 用户指南中的弹性网络接口。

- 有关VPC终端节点的更多信息，请参阅 Amazon VPC 用户指南中的接口VPC终端节点 (AWS PrivateLink)。
- 有关 InfluxDB 操作的时间流的更多信息，请参阅 InfluxDB API 操作的时间流。API

创建接口VPC端点后，如果您为端点启用私有DNS主机名，则为 InfluxDB 端点的默认 Timestream (https://timestream-influxb.*Region*.amazonaws.com) 解析到您的终端节点。VPC如果您不启用私有DNS主机名，Amazon 会VPC提供一个DNS终端节点名称，您可以按以下格式使用该名称：

```
VPC_Endpoint_ID.timestream-influxb.Region.vpce.amazonaws.com
```

有关更多信息，请参阅 Amazon VPC 用户指南中的接口VPC终端节点 (AWS PrivateLink)。InfluxDB 的 Timestream 支持调用你内部的所有API操作。VPC

> **Note**
>
> 只能为中的一个VPC端点启用私有DNS主机名。VPC如果要创建其他VPC端点，则应为其禁用私有DNS主机名。

VPC端点注意事项

在为 InfluxDB VPC 终端节点为 Amazon Timestream 设置接口API终端节点之前，请务必查看亚马逊用户指南中的接口终端节点属性和限制。VPC所有与管理 InfluxDB 资源的 Amazon Timestream 相关的 InfluxDB API 操作的 Timestream 均可从你使用中获得。VPC AWS PrivateLink VPCInflux API DB 端点的 Timestream 支持端点策略。默认情况下，允许通过端点对 Timestream 进行完全访问以进行 InfluxDB API 操作。有关更多信息，请参阅 Amazon VPC 用户指南中的使用VPC终端节点控制对服务的访问。

为 InfluxDB 的时间流创建接口VPC端点 API

你可以使用亚马逊控制台API或 InfluxDB 为 Amazon Timestream 创建VPC终端节点。VPC AWS CLI 有关更多信息，请参阅 Amazon VPC 用户指南中的创建接口终端节点。

创建接口VPC终端节点后，您可以为该终端节点启用私有DNS主机名。当你这样做时，InfluxDB 的默认 Amazon Timestream 端点 (https://timestream-influxb。*Region*.amazonaws.com) 解析到您的终端节点。VPC有关更多信息，请参阅 Amazon VPC 用户指南中的通过接口终端节点访问服务。

为 Influ VPC xDB 的亚马逊 Timestream 创建终端节点策略 API

您可以将端点策略附加到您的VPC终端节点，以控制对 Influ API xDB Timestream 的访问权限。此策略指定以下内容：

- 可执行操作的主体。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅 Amazon VPC 用户指南中的使用VPC终端节点控制对服务的访问。

Example VPCInflu API xDB 操作的 Timestream 端点策略

以下是 Inf API luxDB Timestream 的端点策略示例。当连接到端点时，此策略允许所有资源的所有委托人访问列出的 Timestream 以进行 InfluxDB API 操作。

```
{
 "Statement": [{
  "Principal": "*",
  "Effect": "Allow",
  "Action": [
   "timestream-influxb:CreateDbInstance",
   "timestream-influxb:UpdateDbInstance"
  ],
  "Resource": "*"
 }]
}
```

Example VPC终端节点策略拒绝来自指定 AWS 账户的所有访问权限

以下VPC终端节点策略拒绝 AWS 账号 *123456789012* 使用终端节点访问资源的所有权限。此策略允许来自其他账户的所有操作。

```
{
 "Statement": [{
   "Action": "*",
   "Effect": "Allow",
```

```
    "Resource": "*",
    "Principal": "*"
   },
   {
    "Action": "*",
    "Effect": "Deny",
    "Resource": "*",
    "Principal": {
     "AWS": [
      "123456789012"
     ]
    }
   }
  ]
}
```

InfluxDB Timestream 安全最佳实践

适用于 InfluxDB 的 Amazon Timestream 提供了许多安全功能，供您在制定和实施自己的安全策略时考虑。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。由于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，因此将其视为有用的考虑因素而不是惯例。

实施最低权限访问

在授予权限时，你可以决定谁将获得哪个 Timestream for InfluxDB 资源的权限。您可以对这些资源启用希望允许的特定操作。因此，您应仅授予执行任务所需的权限。实施最低权限访问对于减小安全风险以及可能由错误或恶意意图造成的影响至关重要。

使用IAM角色

生产者和客户端应用程序必须具有有效的凭据才能访问 InfluxDB 数据库实例的 Timestream。您不应将 AWS 证书直接存储在客户端应用程序或 Amazon S3 存储桶中。这些是不会自动轮换的长期凭证，如果它们受到损害，可能会对业务产生重大影响。

相反，您应该使用IAM角色来管理您的创建器和客户端应用程序的临时证书，以便访问 InfluxDB 数据库实例的 Timestream。在使用角色时，您不必使用长期凭证（如用户名和密码或访问密钥）来访问其他资源。

有关更多信息，请参阅《IAM用户指南》中的以下主题：

- IAM角色
- 针对角色的常见情形：用户、应用程序和服务

使用 AWS Identity and Access Management (IAM) 账户控制 InfluxDB API 操作对亚马逊 Timestream 的访问权限，尤其是创建、修改或删除 InfluxDB 资源的亚马逊 Timestream 的操作。此类资源包括数据库实例、安全组和参数组。

- 为每个管理 Amazon Timestream for InfluxDB 资源的人员创建一个单独的用户，包括你自己。不要使用 AWS 根凭证来管理 InfluxDB 资源的 Amazon Timestream。
- 授予每位用户执行其职责所需的最低权限集。
- 使用IAM群组有效管理多个用户的权限。
- 定期轮换 IAM 凭证。
- 将 S AWS ecrets Manager 配置为自动轮换 InfluxDB 的 Amazon Timestream 的密钥。有关更多信息，请参阅 S AWS ecrets Manager 用户指南中的轮换 S AWS ecrets Manager 密钥。您也可以通过编程方式从 S AWS ecrets Manager 中检索凭证。有关更多信息，请参阅 S ecrets Manager 用户指南中的检索AWS 密钥值。
- 使用保护您的InfluxDB流入API令牌的时间流。API代币

实施从属资源中的服务器端加密

可以在 InfluxDB 的 Timestream 中对静态数据和传输中的数据进行加密。有关更多信息，请参阅 传输中加密。

CloudTrail 用于监控API呼叫

InfluxDB 的 Timestream 与一项服务集成，该服务提供用户 AWS CloudTrail、角色或 AWS 服务在 InfluxDB 的 Timestream 中采取的操作的记录。

使用收集的信息 CloudTrail，您可以确定向 Timestream for InfluxDB 发出的请求、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

有关更多信息，请参阅 the section called “使用记录 LiveAnalytics API通话的时间流 AWS CloudTrail”。

适用于 InfluxDB 的 Amazon Timestream 支持控制平面 CloudTrail 事件，但不支持数据平面。有关更多信息，请参阅控制平面和数据平面。

公开可用性

当您在基于 Amazon VPC 服务的虚拟私有云 (VPC) 中启动数据库实例时，您可以打开或关闭该数据库实例的公共可访问性。要指定您创建的数据库实例的DNS名称是否可以解析为公有 IP 地址，请使用公共可访问性参数。通过使用此参数，您可以指定数据库实例是否有公共访问权限

如果您的数据库实例位于VPC但无法公开访问，则您也可以使用 AWS Site-to-Site VPN连接或 Di AWS rect Connect 连接从私有网络访问该实例。

如果您的数据库实例可公开访问，请务必采取措施防止或帮助缓解与拒绝服务相关的威胁。有关更多信息，请参阅拒绝服务攻击简介和保护网络。

API参考

有关适用于 InfluxDB 的亚马逊 Timestream 的完整列表和详细信息，请参阅适用于 InfluxDB 的APIs亚马逊 Timestream。APIs

有关所有 AWS 服务的通用错误代码，请参阅 Supp AWS ort 部分。

文档历史记录

变更	说明	日期
仅限文档的更新	更新了配额主题以区分默认配额和系统限制。	2024年10月22日
亚马逊 Timestream 现在支持查询见解	Timestream 现在支持查询见解功能，该功能可帮助您优化查询、提高查询性能并降低成本。	2024年10月22日
适用于 InfluxDB 的亚马逊 Timestream 更新了现有政策。	适用于 InfluxDB 的 Amazon Timestream 已将该`ec2:DescribeRouteTables` 操作添加到现有`AmazonTimestreamInfluxDBFullAccess` 托管策略中，用于描述您的路由表	2024年10月8日
`AmazonTimestreamReadOnlyAccess` — 更新现有政策	的 Timestream LiveAnalytics 已向`AmazonTimestreamReadOnlyAccess` 托管策略添加了描述 AWS 账户 设置	2024 年 6 月 3 日

	的DescribeAccountSettings 权限。	
亚马逊 Timestream LiveAnalytics 现在支持 Timestream 计算单位 () TCUs	LiveAnalytics 目前，Amazon Timestream 支持时间流计算单位 (TCUs)，用于衡量为满足您的查询需求分配的计算容量。	2024 年 4 月 29 日
添加了新政策	适用于 InfluxDB 的 Amazon Timestream 添加了两项新策略：一项允许该服务管理您账户中的网络接口和安全组。有关更多信息，请参阅AmazonTimestreamInfluxDBServiceRolePolicy。另一个提供创建、更新、删除和列出 Amazon Timestream InfluxDB 实例以及创建和列出参数组的完全管理权限。有关更多信息，请参阅AmazonTimestreamInfluxDBFullAccess。	2024 年 3 月 14 日
适用于 InfluxDB 的亚马逊 Timestream 现已正式上市。	本文档涵盖了适用于 InfluxDB 的 Amazon Timestream 的初始版本。	2024 年 3 月 14 日
Amazon Timestream for LiveAnalytics Query 事件可用于 AWS CloudTrail	Amazon Timestream LiveAnalytics 现在向发布查询API数据事件。 AWS CloudTrail客户可以审核在其 AWS 账户中提出的所有查询API请求，并查看诸如哪个IAM用户/角色发出了请求、何时发出请求、查询了哪些数据库和表以及请求的查询 ID 等信息。	2023 年 9 月 12 日

适用于 Amazon Timestream LiveAnalytics UNLOAD	Amazon Timestream LiveAnalytics 现在支持UNLOAD将查询结果导出到 S3。	2023 年 5 月 12 日
Amazon Timestream 用于 LiveAnalytics 更新现有政策。	已向托管策略添加批量加载权限。	2023 年 2 月 24 日
用于 LiveAnalytics 批量加载的 Amazon Timestream。	亚马逊 Timestream LiveAnalytics 目前支持批量加载功能。	2023 年 2 月 24 日
亚马逊 Timestream LiveAnalytics 目前支持。 AWS Backup	亚马逊 Timestream LiveAnalytics 目前支持。 AWS Backup	2022 年 12 月 14 日
Amazon Timestream 获取托 LiveAnalytics 管政策的 AWS 更新	有关 AWS 托管策略和 Amazon Timestream 的新信息 LiveAnalytics，包括对现有托管策略的更新。	2021 年 11 月 29 日
适用于 Amazon Timestream 的 LiveAnalytics 支持预定查询	Amazon Timestream LiveAnalytics 目前支持根据计划代表您运行查询。	2021 年 11 月 29 日
适用于的亚马逊 Timestream LiveAnalytics 支持磁性存储。	Amazon Timestream LiveAnalytics 现在支持使用磁性存储进行表写入。	2021 年 11 月 29 日
用于 LiveAnalytics 多指标记录的 Amazon Timestream。	LiveAnalytics 目前，Amazon Timestream 支持更紧凑的格式来存储您的时间序列数据。	2021 年 11 月 29 日
Amazon Timestream 获取托 LiveAnalytics 管政策的AWS更新	有关 AWS 托管策略和 Amazon Timestream 的新信息 LiveAnalytics，包括对现有托管策略的更新。	2021 年 5 月 24 日
Amazon Timestream LiveAnalytics 版现已在欧洲（法兰克福）地区上市。	Amazon Timestream 版现已在欧洲（法兰克福）地区正式上市（`eu-central-1` ）。LiveAnalytics	2021 年 4 月 23 日

适用于 Amazon Timestream 的 LiveAnalytics 版本现在支持 VPC终端节点 ()AWS PrivateLink。	Amazon Timestream LiveAnalytics 目前支持使用 VPC终端节点 ()AWS PrivateLink。	2021 年 3 月 23 日
亚马逊 Timestream 现在支持跨表查询。	您可以使用 Amazon Timestream LiveAnalytics 来运行跨表查询。	2021 年 2 月 10 日
Amazon Timestream LiveAnalytics 目前支持增强的查询执行统计数据。	Amazon Timestream LiveAnalytics 现在支持增强的查询执行统计数据，例如扫描的数据量。	2021 年 2 月 10 日
Amazon Timestream LiveAnalytics 目前支持高级时间序列函数。	您可以使用 Amazon Timestream LiveAnalytics 来运行带有高级时间序列函数的SQL查询，例如导数、积分和相关性。	2021 年 2 月 10 日
Amazon Timestream LiveAnalytics 现在已合规 HIPAAISO，且PCI合规。	现在，您可以将 Amazon Timestream LiveAnalytics 用于需要和PCI合HIPAA规ISO基础设施的工作负载。	2021 年 1 月 27 日
亚马逊 Timestream LiveAnalytics 目前支持开源 Telegraf 和 grafana。	现在，您可以使用开源、插件驱动的服务器代理 Telegraf 来收集和报告指标，也可以使用 Grafana（数据库的开源分析和监控平台）和 Amazon Timestream 的开源分析和监控平台。 LiveAnalytics	2020 年 11 月 25 日
适用于 Amazon Timestream 的 LiveAnalytics 版现已正式上市。	本文档涵盖了 Amazon Timestream 的初始版本。 LiveAnalytics	2020 年 9 月 30 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。