SQL 开发人员指南

适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 开发人员指南

适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 开发人员指南: SQL 开发人员指南

Table of Contents

对于新项目，建议您使用新的适用于 Apache Flink Studio 的托管服务，而不是使用适用于 SQL 应用程序的 Kinesis Data Analytics。Managed Service for Apache Flink Studio 不仅操作简单，还具有高级分析功能，使您能够在几分钟内构建复杂的流处理应用程序。

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。

什么是适用于 SQL 应用程序的 Amazon Kinesis Data Analytics？

借助适用于 SQL 应用程序的 Amazon Kinesis Data Analytics，您可以使用标准 SQL 来处理和分析流数据。您可以使用该服务针对流式传输源快速编写和运行强大的 SQL 代码，以执行时间序列分析，为实时控制面板提供信息以及创建实时指标。

要开始使用 Kinesis Data Analytics，您可以创建一个 Kinesis Data Analytics 应用程序以持续读取和处理流数据。该服务支持从亚马逊 Kinesis Data Streams 和 Amazon Data Firehose 流媒体源提取数据。然后，您可以使用交互式编辑器编写 SQL 代码，并使用实时流数据测试它。您还可以配置 Kinesis Data Analytics 要将结果发送到的目标。

Kinesis Data Analytics 支持亚马逊数据 Firehose（亚马逊 S3、亚马逊 Redshift、亚马逊服务和 Splunk）和 OpenSearch 亚马逊 AWS LambdaKinesis Data Streams 作为目的地。

我应该什么时候使用 Amazon Kinesis Data Analytics？

通过使用 Amazon Kinesis Data Analytics，您可以快速编写 SQL 代码以使用近乎实时的方式持续读取、处理和存储数据。通过对流数据采用标准 SQL 查询，您可以构建转换数据并深入了解这些数据的应用程序。下面提供了一些使用 Kinesis Data Analytics 的示例方案：

- 生成时间序列分析 - 您可以基于时间范围计算指标，然后通过 Kinesis 数据传输流将值传输到 Amazon S3 或 Amazon Redshift。
- 为实时控制面板提供信息 - 您可以向下游发送处理的聚合流数据结果，以便为实时控制面板提供信息。
- 创建实时指标 - 您可以创建自定义指标和触发器，以用于实时监控、通知和警报。

有关 Kinesis Data Analytics 支持的 SQL 语言元素的信息，请参阅 Amazon Kinesis Data Analytics SQL 参考。

你是否首次使用 Amazon Kinesis Data Analytics？

如果您是首次接触 Amazon Kinesis Data Analytics 的用户，我们建议您按顺序阅读以下内容：

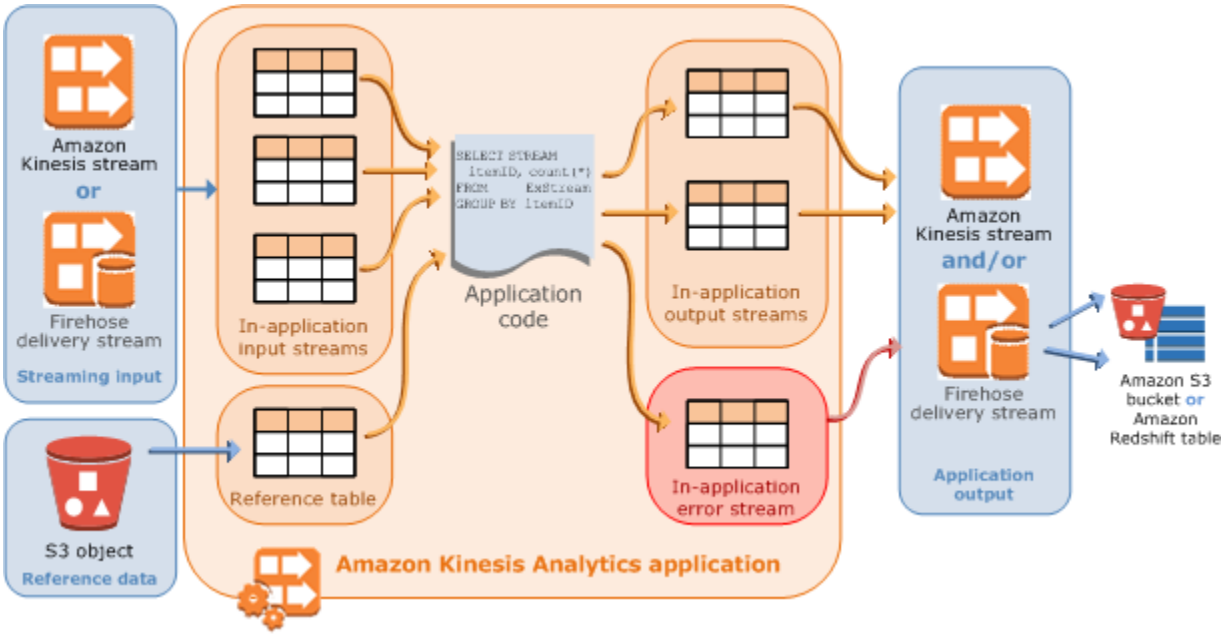
1. 阅读本指南的“工作原理”部分。本节介绍各种 Kinesis Data Analytics 组件，您可以使用这些组件来 end-to-end 创建体验。有关更多信息，请参阅适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理。
2. 尝试入门练习。有关更多信息，请参阅适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 入门。
3. 了解流式 SQL 概念。有关更多信息，请参阅流式 SQL 概念。
4. 尝试其他示例。有关更多信息，请参阅 适用于 SQL 的 Kinesis Data Analytics 示例。

适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理

> **Note**
>
> 2023 年 9 月 12 日之后，如果您尚未使用适用于 SQL 的 Kinesis Data Analytics，则将无法使用 Kinesis Data Firehose 作为来源创建新应用程序。有关更多信息，请参阅限制。

应用程序是可以在您的账户中创建的 Amazon Kinesis Data Analytics 中的主要资源。您可以使用 AWS Management Console 或 Kinesis Data Analytics API 创建和管理应用程序。Kinesis Data Analytics 提供 API 操作以管理应用程序。有关 API 操作的列表，请参阅操作。

Kinesis Data Analytics 应用程序持续实时读取和处理流数据。您可以使用 SQL 编写应用程序代码来处理传入的流数据并生成输出。然后，Kinesis Data Analytics 会将输出写入到配置的目标中。以下示意图说明典型的应用程序架构。



每个应用程序都有名称、描述、版本 ID 和状态。Amazon Kinesis Data Analytics 会在您首次创建应用程序时分配一个版本 ID。在您更新任何应用程序配置时，此版本 ID 会更新。例如，如果您添加输入配置，添加或删除引用数据源，添加或删除输出配置，或者更新应用程序代码，Kinesis Data Analytics 都会更新当前的应用程序版本 ID。另外，Kinesis Data Analytics 也会维护应用程序的创建和上次更新时间戳。

除了这些基本属性外，每个应用程序还包括：

- Input - 应用程序的流式传输源。您可以选择 Kinesis 数据流或 Firehose 数据传输流作为流媒体源。在输入配置中，您可以将流式传输源映射到应用程序内部输入流。应用程序内部流类似于可以对其执行 SELECT 和 INSERT SQL 操作的连续更新表。在您的应用程序代码中，可以创建其他应用程序内部流来存储中间查询结果。

  您可以选择将单个流式传输源分区到多个应用程序内部输入流以提高吞吐量。有关更多信息，请参阅 Limits 和 配置应用程序输入。

  Amazon Kinesis Data Analytics 在每个应用程序流中提供一个名为 时间戳和 ROWTIME 列 的时间戳列。您可以在基于时间的窗口式查询中使用该列。有关更多信息，请参阅窗口式查询。

  您可以选择配置引用数据源，以便丰富您在应用程序中的输入数据流。这会生成应用程序内部引用表。您必须将引用数据存储为 S3 存储桶中的对象。在应用程序启动时，Amazon Kinesis Data Analytics 会读取 Amazon S3 对象并创建应用程序内部表。有关更多信息，请参阅配置应用程序输入。

- 应用程序代码 - 处理输入和生成输出的一系列 SQL 语句。您可以针对应用程序内部流和引用表编写 SQL 语句。您还可以编写 JOIN 查询以组合来自这两个源的数据。

  有关 支持的 SQL 语言元素的信息，请参阅Amazon Kinesis Data Analytics SQL 参考。

  在采用最简单的形式时，应用程序代码可以是单个 SQL 语句，它从流式输入中选择并将结果插入到流式输出中。它还可以是一系列 SQL 语句，将一个 SQL 语句的输出馈送到下一个 SQL 语句的输入。此外，您可以编写应用程序代码以将输入流拆分为多个流。然后，您可以应用其他查询来处理这些流。有关更多信息，请参阅应用程序代码。

- 输出 - 在应用程序代码中，查询结果将保存到应用程序内部流中。在您的应用程序代码中，您可以创建一个或多个应用程序内部流保存中间结果。然后，您可以选择配置应用程序输出以在应用程序内部流中永久保存数据，而这些应用程序内部流将应用程序输出（也称为应用程序内部输出流）保存到外部目标中。外部目标可以是 Firehose 传输流或 Kinesis 数据流。请注意有关这两种目标的以下信息：

  - 您可以配置 Firehose 传输流以将结果写入亚马逊 S3、亚马逊 Redshift 或 OpenSearch 亚马逊服务（服务OpenSearch ）。

  - 您也可以将应用程序输出写入到自定义目标，而不是 Amazon S3 或 Amazon Redshift。为此，请指定一个 Kinesis 数据流以作为输出配置中的目标。然后，您可以配置 AWS Lambda 为轮询流并调用您的 Lambda 函数。您的 Lambda 函数代码接收流数据以作为输入。在您的 Lambda 函数代码中，您可以将传入数据写入到自定义目标中。有关更多信息，请参阅AWS Lambda 与亚马逊 Kinesis Data Analytics 配合使用。

  有关更多信息，请参阅配置应用程序输出。

此外，请注意以下情况：

- Amazon Kinesis Data Analytics 需要具有相应的权限，以从流式传输源中读取记录并将应用程序输出写入到外部目标中。您可以使用 IAM 角色以授予这些权限。

- Kinesis Data Analytics 自动为每个应用程序提供应用程序内部错误流。如果您的应用程序在处理某些记录时出现问题（例如由于类型不匹配或者到达延迟），记录将写入错误流。您可以配置应用程序输出，以指示 Kinesis Data Analytics 将错误流数据永久保存到外部目标以进行进一步的评估。有关更多信息，请参阅错误处理。

- Amazon Kinesis Data Analytics 确保您的应用程序输出记录写入到配置的目标中。它“至少一次”使用处理和传输模型，即使在您遇到应用程序中断的情况下也是如此。有关更多信息，请参阅 将应用程序输出永久保存到外部目标的传输模型。

主题

- 配置应用程序输入
- 应用程序代码

配置应用程序输入

您的 Amazon Kinesis Data Analytics 应用程序可以从单个流式传输源中接收输入，并且可以选择使用一个引用数据源。有关更多信息，请参阅 适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理。本主题的此部分介绍了应用程序输入源。

主题

配置流式传输源

当您创建应用程序时，可以指定流式传输源。您还可以在创建应用程序后修改输入。Amazon Kinesis Data Analytics 支持应用程序的以下流式源：

- Kinesis 数据流
- Firehose 的传送直播

> **Note**
>
> 2023 年 9 月 12 日之后，如果您尚未使用适用于 SQL 的 Kinesis Data Analytics，则将无法使用 Kinesis Data Firehose 作为来源创建新应用程序。使用 `KinesisFirehoseInput` 对 Kinesis Data Analytics for SQL 应用程序进行操作的现有客户可以继续使用

`KinesisFirehoseInput` 在使用 Kinesis Data Analytics 的现有账户内添加应用程序。如果您是现有客户，并希望使用 `KinesisFirehoseInput` 在适用于 SQL 应用程序的 Kinesis Data Analytics 中创建一个新的账户，则可以通过服务限制增加表创建案例。有关更多信息，请参阅 AWS Support 中心。在将所有新应用程序推广到生产环境之前，务必对其进行测试。

> **Note**
>
> 如果 Kinesis 数据流是加密的，Kinesis Data Analytics 会无缝地访问加密流中的数据，无需进一步配置。Kinesis Data Analytics 不存储从 Kinesis 数据流读取的未加密数据。有关更多信息，请参阅什么是 Kinesis 数据流的服务器端加密？。

Kinesis Data Analytics 持续轮询流式传输源以查找新数据，并根据输入配置在应用程序内部流中提取该数据。

> **Note**
>
> 添加 Kinesis 流作为应用程序的输入不会影响流中的数据。如果其他资源（例如 Firehose 交付流）也访问了相同的 Kinesis 流，则 Firehose 交付流和 Kinesis Data Analytics 应用程序都将收到相同的数据。但是，吞吐量和限制可能会受到影响。

您的应用程序代码可以查询应用程序内部流。作为输入配置的一部分，您需要提供以下内容：

- 流式传输源 – 您提供流的 Amazon 资源名称 (ARN) 以及 Kinesis Data Analytics 可担任的 IAM 角色以代表您访问流。
- 应用程序内部流名称前缀 – 在启动应用程序时，Kinesis Data Analytics 创建指定的应用程序内部流。在您的应用程序代码中，可以使用此名称访问应用程序内部流。

  您可以选择将一个流式传输源映射到多个应用程序内部流。有关更多信息，请参阅Limits。在这种情况下，Amazon Kinesis Data Analytics 使用如下名称创建指定数量的应用程序内部流：*prefix*_001、*prefix*_002 和 *prefix*_003。默认情况下，Kinesis Data Analytics 将流式传输源映射到一个名为 *prefix* _001的应用程序内部流。

  在应用程序内部流中插入行时有速度限制。因此，Kinesis Data Analytics 支持多个此类应用程序内部流，以便以快得多的速度将记录添加到应用程序中。如果发现应用程序无法及时处理流式传输源中的数据，您可以添加并行度单元以提高性能。

- 映射架构 – 您描述流式传输源上的记录格式 (JSON、CSV)。您还描述流上的每个记录如何映射到创建的应用程序内部流中的列。您可以在此处提供列名和数据类型。

> **Note**
>
> 在创建输入应用程序内部流时，Kinesis Data Analytics 使用引号将标识符 (流名称和列名称) 引起来。在查询该流和列时，您必须在引号内使用相同的大小写指定它们 (小写和大写字母完全匹配)。有关标识符的更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的标识符。

您可以通过 Amazon Kinesis Data Analytics 控制台创建一个应用程序并配置输入。然后，控制台执行必需的 API 调用。创建新应用程序 API 或者将输入配置添加到现有应用程序时，可以配置应用程序输入。有关更多信息，请参阅 CreateApplication 和 AddApplicationInput。下面是 `Createapplication` API 请求正文的输入配置部分：

```
"Inputs": [
       {
           "InputSchema": {
               "RecordColumns": [
                   {
                       "Mapping": "string",
                       "Name": "string",
                       "SqlType": "string"
                   }
               ],
               "RecordEncoding": "string",
               "RecordFormat": {
                   "MappingParameters": {
                       "CSVMappingParameters": {
                           "RecordColumnDelimiter": "string",
                           "RecordRowDelimiter": "string"
                       },
                       "JSONMappingParameters": {
                           "RecordRowPath": "string"
                       }
                   },
                   "RecordFormatType": "string"
               }
           },
           "KinesisFirehoseInput": {
```

```
                "ResourceARN": "string",
                "RoleARN": "string"
            },
            "KinesisStreamsInput": {
                "ResourceARN": "string",
                "RoleARN": "string"
            },
            "Name": "string"
        }
    ]
```

配置引用源

您还可以选择将引用数据源添加到现有应用程序中，以便扩充来自流式传输源的数据。您必须将引用数据存储为 Amazon S3 存储桶中的对象。在应用程序启动时，Amazon Kinesis Data Analytics 读取 Amazon S3 对象并创建应用程序内部引用表。然后，您的应用程序代码可以将其与应用程序内部流联接。

您可以使用支持的格式 (CSV、JSON) 在 Amazon S3 对象中存储引用数据。例如，假设您的应用程序对股票订单执行分析。对流式传输源采用以下记录格式：

```
Ticker, SalePrice, OrderId

AMZN     $700        1003
XYZ      $250        1004
...
```

在这种情况下，您可能考虑维护引用数据源，以提供有关每个股票行情机的详细信息，如公司名称。

```
Ticker, Company
AMZN, Amazon
XYZ, SomeCompany
...
```

您可以使用 API 或控制台添加应用程序引用数据源。Amazon Kinesis Data Analytics 提供以下 API 操作来管理引用数据源：

- AddApplicationReferenceDataSource
- UpdateApplication

有关使用控制台添加引用数据的信息，请参阅示例：在 应用程序中添加引用数据。

请注意以下几点：

- 如果应用程序正在运行，则 Kinesis Data Analytics 创建一个应用程序内部引用表，然后立即加载引用数据。
- 如果应用程序未运行 (例如，处于就绪状态)，则 Kinesis Data Analytics 仅保存更新的输入配置。在应用程序开始运行时，Kinesis Data Analytics 在应用程序中将引用数据作为表进行加载。

假设您希望在 Kinesis Data Analytics 创建应用程序内部引用表后刷新数据。您可能更新了 Amazon S3 对象，或者要使用不同的 Amazon S3 对象。在这种情况下，您可以显式调用 UpdateApplication，或在控制台中依次选择操作、Synchronize reference data table (同步引用数据表)。Kinesis Data Analytics 不会自动刷新应用程序内部引用表。

可作为引用数据源创建的 Amazon S3 对象具有大小限制。有关更多信息，请参阅Limits。如果对象大小超出该限制，则 Kinesis Data Analytics 无法加载数据。应用程序状态显示为正在运行，但是不读取数据。

当添加引用数据源时，您需要提供以下信息：

- S3 存储桶和对象键名称 – 除了存储桶名称和对象键以外，您还需要提供 Kinesis Data Analytics 可担任的 IAM 角色以代表您读取对象。
- 应用程序内部引用表名称 – Kinesis Data Analytics 创建该应用程序内部表并读取 Amazon S3 对象以填充该表。这是您在应用程序代码中指定的表名称。
- 映射架构 - 您描述记录格式 (JSON、CSV)，即 Amazon S3 对象中存储的数据的编码。您还可以描述每个对象元素如何映射到应用程序内部引用表中的列。

下面显示 AddApplicationReferenceDataSource API 请求中的请求正文。

```
{
    "applicationName": "string",
    "CurrentapplicationVersionId": number,
    "ReferenceDataSource": {
        "ReferenceSchema": {
            "RecordColumns": [
                {
                    "IsDropped": boolean,
                    "Mapping": "string",
                    "Name": "string",
```

```
                    "SqlType": "string"
                }
            ],
            "RecordEncoding": "string",
            "RecordFormat": {
                "MappingParameters": {
                    "CSVMappingParameters": {
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                    },
                    "JSONMappingParameters": {
                        "RecordRowPath": "string"
                    }
                },
                "RecordFormatType": "string"
            }
        },
        "S3ReferenceDataSource": {
            "BucketARN": "string",
            "FileKey": "string",
            "ReferenceRoleARN": "string"
        },
        "TableName": "string"
    }
}
```

使用 JSONPath

> **Note**
>
> 2023 年 9 月 12 日之后，如果您尚未使用适用于 SQL 的 Kinesis Data Analytics，则将无法使用 Kinesis Data Firehose 作为来源创建新应用程序。有关更多信息，请参阅限制。

JSONPath 是查询 JSON 对象的元素的标准化方式。JSONPath 使用表达式在 JSON 文档中的元素、嵌套元素和数组之间导航。有关 JSON 的更多信息，请参阅 JSON 简介。

Amazon Kinesis Data Analytics 使用应用程序源架构中的 JSONPath 表达式来识别包含 JSON 格式数据的流式源中的数据元素。

有关如何将流式数据映射到应用程序输入流的更多信息，请参阅the section called “将流式传输源元素映射到 SQL 输入列”。

使用 JSONPath 访问 JSON 元素

接下来，您将了解如何使用 JSONPath 表达式访问 JSON 格式的数据中的各种元素。对于此部分中的示例，请假设源流包含以下 JSON 记录：

```
{
  "customerName":"John Doe",
  "address":
  {
    "streetAddress":
    [
      "number":"123",
      "street":"AnyStreet"
    ],
    "city":"Anytown"
  }
  "orders":
  [
    { "orderId":"23284", "itemName":"Widget", "itemPrice":"33.99" },
    { "orderId":"63122", "itemName":"Gadget", "itemPrice":"22.50" },
    { "orderId":"77284", "itemName":"Sprocket", "itemPrice":"12.00" }
  ]
}
```

访问 JSON 元素

要使用 JSONPath 查询 JSON 数据中的元素，请使用以下语法。在此处，$ 表示数据层次结构的根，elementName 是要查询的元素节点的名称。

```
$.elementName
```

以下表达式将查询前面的 JSON 示例中的 customerName 元素。

```
$.customerName
```

前面的表达式将从前面的 JSON 记录返回以下内容。

```
John Doe
```

> **Note**
>
> 路径表达式区分大小写。表达式 `$.customername` 将从前面的 JSON 示例返回 `null`。

> **Note**
>
> 如果路径表达式指定的位置没有出现任何元素，则表达式返回 `null`。以下表达式将从前面的 JSON 示例返回 `null`，因为没有匹配元素。
>
> ```
> $.customerId
> ```

访问嵌套 JSON 元素

要查询嵌套 JSON 元素，请使用以下语法。

```
$.parentElement.element
```

以下表达式将查询前面的 JSON 示例中的 `city` 元素。

```
$.address.city
```

前面的表达式将从前面的 JSON 记录返回以下内容。

```
Anytown
```

您可以使用以下语句进一步查询子元素。

```
$.parentElement.element.subElement
```

以下表达式将查询前面的 JSON 示例中的 `street` 元素。

```
$.address.streetAddress.street
```

前面的表达式将从前面的 JSON 记录返回以下内容。

```
AnyStreet
```

访问数组

您可以通过以下方式访问 JSON 数组中的数据：

- 将数组中的所有元素作为单一的行进行检索。
- 将数组中的每个元素作为单独的行进行检索。

以单个行检索数组中的所有元素

要以单个行查询一个数组的全部内容，请使用以下语法。

```
$.arrayObject[0:]
```

以下表达式将查询本部分前面所用 JSON 示例中的 orders 元素的全部内容。它将返回单个行的单个列中的数组内容。

```
$.orders[0:]
```

上述表达式从本部分所用的示例 JSON 记录返回如下内容。

```
[{"orderId":"23284","itemName":"Widget","itemPrice":"33.99"},
{"orderId":"61322","itemName":"Gadget","itemPrice":"22.50"},
{"orderId":"77284","itemName":"Sprocket","itemPrice":"12.00"}]
```

以单独的行分别检索数组中的所有元素

要以单独的行分别查询一个数组中的各个元素，请使用以下语法。

```
$.arrayObject[0:].element
```

以下表达式将查询前面的 JSON 示例中的 orderId 元素，并将每个数组元素作为一个单独的行返回。

```
$.orders[0:].orderId
```

前面的表达式将从前面的 JSON 记录返回以下内容，其中每个数据项都将作为一个单独的行返回。

23284
63122
77284

Note

如果查询非数组元素的表达式包含在查询各个数组元素的架构中，则非数组元素将针对数组中的每个元素重复。例如，假设前面的 JSON 示例的架构包含以下表达式：

- $.customerName
- $.orders[0:].orderId

在这种情况下，从示例输入流元素返回的数据行将类似于以下内容 (其中的 `name` 元素将针对每个 `orderId` 元素重复)。

John Doe	23284
John Doe	63122
John Doe	77284

Note

以下限制适用于 Amazon Kinesis Data Analytics 中的数组表达式：

- 数组表达式中仅支持一个级别的解除引用。不支持以下表达式格式。

```
$.arrayObject[0:].element[0:].subElement
```

- 一个架构中仅可平展一个数组。可以引用多个数组——作为一个包含数组中的所有元素的行返回。但是，只有一个数组可以让其每个元素都作为单个行返回。

  包含以下格式的元素的架构有效。此格式会将第二个数组的内容作为单个列返回，该内容针对第一个数组中的每一个元素重复。

```
$.arrayObjectOne[0:].element
$.arrayObjectTwo[0:]
```

包含以下格式的元素的架构无效。

```
$.arrayObjectOne[0:].element
$.arrayObjectTwo[0:].element
```

其他考虑因素

使用 JSONPath 的其他注意事项如下所示：

- 如果应用程序架构中的 JSONPath 表达式中的单个元素没有访问任何数组，则会在应用程序的输入流中为处理的每个 JSON 记录创建一行。
- 当数组被平展后（即，它的元素以单独的行返回），任何缺失的元素都会导致在应用程序内部流中创建一个 null 值。
- 一个数组将始终被平展为至少一行。如果不会返回任何值 (即，数组为空或没有查询数组的任何元素)，则会返回包含所有 null 值的单个行。

  以下表达式将从前面的 JSON 示例返回包含 null 值的记录，因为在指定的路径没有匹配的元素。

  ```
  $.orders[0:].itemId
  ```

  前面的表达式将从前面的 JSON 示例记录返回以下内容。

null
null
null

相关主题

- 介绍 JSON

将流式传输源元素映射到 SQL 输入列

> **Note**
>
> 2023 年 9 月 12 日之后，如果您尚未使用适用于 SQL 的 Kinesis Data Analytics，则将无法使用 Kinesis Data Firehose 作为来源创建新应用程序。有关更多信息，请参阅限制。

通过使用 Amazon Kinesis Data Analytics，您可以使用标准 SQL 处理和分析 JSON 或 CSV 格式的流数据。

- 要处理和分析流 CSV 数据，您可以为输入流的列分配列名称和数据类型。您的应用程序将按顺序从每个列定义的输入流中导入一个列。

  您不需要在应用程序输入流中包含所有列，但您不能跳过源流中的列。例如，您可以从包含五个列的一个输入流中导入前三个列，但不能进导入列 1、2 和 4。
- 要处理和分析流 JSON 数据，您可以使用 JSONPath 表达式将流式传输源中的 JSON 元素映射到输入流中的 SQL 列。有关通过 Amazon Kinesis Data Analytics 使用 JSONPath 的更多信息，请参阅 使用 JSONPath。SQL 表中的列具有从 JSON 类型中映射的数据类型。有关支持的数据类型，请参阅数据类型。有关将 JSON 数据转换成 SQL 数据的详细信息，请参阅将 JSON 数据类型映射到 SQL 数据类型。

有关如何配置输入流的详细信息，请参阅配置应用程序输入。

将 JSON 数据映射到 SQL 列

您可以使用 AWS Management Console 或 Kinesis Data Analytics API 将 JSON 元素映射到输入列。

- 要使用控制台将元素映射到列，请参阅使用架构编辑器。
- 如需使用 Kinesis Data Analytics API 将元素映射到列，请参阅以下部分。

要将 JSON 元素映射到应用程序内部输入流中的列，您需要使用在每个列中包含以下信息的架构：

- 源表达式：指定列的数据位置的 JSONPath 表达式。
- 列名称：您的 SQL 查询用于引用数据的名称。
- 数据类型：列的 SQL 数据类型。

使用 API

如需将流式源中的元素映射到输入列，您可以使用 Kinesis Data Analytics API CreateApplication 操作。要创建应用程序内部流，请指定一个架构以将您的数据转换为 SQL 中使用的架构化版本。CreateApplication 操作会将您的应用程序配置为接收来自单个流式传输源的输入。要将 JSON 元素或 CSV 列映射到 SQL 列，您应在 SourceSchema `RecordColumns` 数组中创建一个 RecordColumn 对象。RecordColumn 对象具有以下架构：

```
{
    "Mapping": "String",
    "Name": "String",
    "SqlType": "String"
}
```

RecordColumn 对象中的字段具有以下值：

- `Mapping`：用于识别列输入源记录中数据的位置的 JSONPath 表达式。采用 CSV 格式的源流的输入架构不存在此值。
- `Name`：应用程序内 SQL 数据流中的列名称。
- `SqlType`：应用程序内 SQL 数据流中的数据的数据类型。

JSON 输入架构示例

以下示例展示了 JSON 架构的 `InputSchema` 值的格式。

```
"InputSchema": {
    "RecordColumns": [
        {
            "SqlType": "VARCHAR(4)",
            "Name": "TICKER_SYMBOL",
            "Mapping": "$.TICKER_SYMBOL"
        },
        {
            "SqlType": "VARCHAR(16)",
            "Name": "SECTOR",
            "Mapping": "$.SECTOR"
        },
        {
```

```
            "SqlType": "TINYINT",
            "Name": "CHANGE",
            "Mapping": "$.CHANGE"
        },
        {
            "SqlType": "DECIMAL(5,2)",
            "Name": "PRICE",
            "Mapping": "$.PRICE"
        }
    ],
    "RecordFormat": {
        "MappingParameters": {
            "JSONMappingParameters": {
                "RecordRowPath": "$"
            }
        },
        "RecordFormatType": "JSON"
    },
    "RecordEncoding": "UTF-8"
}
```

CSV 输入架构示例

以下示例展示了采用逗号分隔值 (CSV) 格式的架构的 `InputSchema` 值的格式。

```
"InputSchema": {
    "RecordColumns": [
        {
            "SqlType": "VARCHAR(16)",
            "Name": "LastName"
        },
        {
            "SqlType": "VARCHAR(16)",
            "Name": "FirstName"
        },
        {
            "SqlType": "INTEGER",
             "Name": "CustomerId"
        }
    ],
    "RecordFormat": {
        "MappingParameters": {
            "CSVMappingParameters": {
```

```
                "RecordColumnDelimiter": ",",
                "RecordRowDelimiter": "\n"
            }
        },
        "RecordFormatType": "CSV"
    },
    "RecordEncoding": "UTF-8"
}
```

将 JSON 数据类型映射到 SQL 数据类型

JSON 数据类型将根据应用程序的输入架构转换为相应的 SQL 输入类型。有关支持的 SQL 数据类型的信息，请参阅数据类型。Amazon Kinesis Data Analytics 将根据以下规则将 JSON 数据类型转换为 SQL 数据类型。

Null 文本

无论目标数据类型如何，JSON 输入流中的 null 文本 (即，`"City":null`) 都将转换为 SQL null。

布尔文本

JSON 输入流中的布尔文本 (即 `"Contacted":true`) 将按以下形式转换为 SQL 数据：

- 数值 (DECIMAL、INT 等)：`true` 转换为 1；`false` 转换为 0。
- 二进制 (BINARY 或 VARBINARY)：
  - `true`：结果已设置最低位并清除剩余位。
  - `false`：结果已清除所有位。

  转换为 VARBINARY 将产生长度为 1 个字节的值。
- 布尔值：转换为相应的 SQL 布尔值。
- 字符 (CHAR 或 VARCHAR)：转换为相应的字符串值 (`true` 或 `false`)。值将被截断以适应字段的长度。
- 日期时间 (DATE、TIME 或 TIMESTAMP)：转换将失败，并且一个强制转换错误将写入到错误流。

数字

JSON 输入流中的数字文本 (即 `"CustomerId":67321`) 将按以下形式转换为 SQL 数据：

- 数值 (DECIMAL、INT 等)：直接转换。如果转换后的值超过目标数据类型 (即，将 123.4 转换为 INT) 的大小或精度，转换将失败，并且一个强制转换错误将写入到错误流。
- 二进制 (BINARY 或 VARBINARY)：转换将失败，并且一个强制转换错误将写入到错误流。
- BOOLEAN:
  - 0：转换为 false。
  - 所有其他数字：转换为 true。
- 字符 (CHAR 或 VARCHAR)：转换为数字的字符串表示形式。
- 日期时间 (DATE、TIME 或 TIMESTAMP)：转换将失败，并且一个强制转换错误将写入到错误流。

字符串

JSON 输入流中的字符串值 (即 "CustomerName":"John Doe") 将按以下形式转换为 SQL 数据：

- 数字 (DECIMAL、INT 等)：Amazon Kinesis Data Analytics 尝试将值转换为目标数据类型。如果该值无法转换，则转换将失败，并且一个强制转换错误将写入到错误流。
- 二进制 (BINARY 或 VARBINARY)：如果源字符串是有效的二进制文本 (即包含偶数数量的 f 的 X'3F67A23A')，则该值将转换为目标数据类型。否则，转换将失败，并且一个强制转换错误将写入到错误流。
- 布尔值：如果源字符串为 "true"，则转换为 true。此比较不区分大小写。否则，转换为 false。
- 字符 (CHAR 或 VARCHAR)：转换为输入中的字符串值。如果该值长于目标数据类型，那么它将被截断，并且任何错误都不会将写入到错误流。
- 日期时间 (DATE、TIME 或 TIMESTAMP)：如果源字符串采用了可转换为目标值的格式，则转换该值。否则，转换将失败，并且一个强制转换错误将写入到错误流。

  有效的日期时间格式包括：
  - “1992-02-14”
  - “1992-02-14 18:35:44.0”

数组或对象

JSON 输入流中的数组或对象将按以下形式转换为 SQL 数据：

- 字符 (CHAR 或 VARCHAR)：转换为数组或对象的源文本。请参阅 访问数组。
- 所有其他数据类型：转换将失败，并且一个强制转换错误将写入到错误流。

有关 JSON 数组的示例，请参阅使用 JSONPath。

相关主题

- 配置应用程序输入
- 数据类型
- 使用架构编辑器
- CreateApplication
- RecordColumn
- SourceSchema

针对流数据使用架构发现功能

> **Note**
>
> 2023 年 9 月 12 日之后，如果您尚未使用适用于 SQL 的 Kinesis Data Analytics，则将无法使用 Kinesis Data Firehose 作为来源创建新应用程序。有关更多信息，请参阅限制。

提供输入架构以描述流输入中的记录映射到应用程序内部流可能非常麻烦，并且容易出错。可以使用 DiscoverInputSchema API（称为发现 API）来推断架构。API 通过使用有关流式传输源的记录的随机示例，可以推断架构（即列名、数据类型和传入数据中数据元素的位置）。

> **Note**
>
> 如需使用发现 API 以通过 Amazon S3 中存储的文件生成架构，请参阅 针对静态数据使用架构发现功能。

控制台使用发现 API 来生成指定流式传输源的架构。利用控制台，您还可以更新架构，包括添加或删除列、更改列名称或数据类型等。不过，请仔细进行更改以确保不会创建无效的架构。

在为应用程序内部流完成架构后，您可以使用一些函数来处理字符串和日期时间值。在生成的应用程序内部流中使用行时，您可以在应用程序代码中利用这些函数。有关更多信息，请参阅示例：转换 DateTime 值。

架构发现期间的列命名

在架构发现期间，Amazon Kinesis Data Analytics 尽可能多地保留流输入源中的原始列名称，但以下情况除外：

- 源流列名称是预留 SQL 关键字，例如 TIMESTAMP、USER、VALUES 或 YEAR。
- 源流列名称包含不受支持的字符。只有字母、数字和下划线字符 (_) 受支持。
- 源流列名称以数字开头。
- 源流列名称的长度超过 100 个字符。

如果重命名了某个列，则重命名的架构列名称将以 COL_ 开头。在某些情况下，不能保留任何原始列名称——例如当整个名称都是不受支持的字符时。在这种情况下，该列将被命名为 COL_#，其中 # 是一个数字，表示该列在列顺序中的位置。

发现完成后，您可以使用控制台更新架构以添加或删除列，也可以更改列名称、数据类型或数据大小。

发现建议的列名称的示例

源流列名称	发现建议的列名称
USER	COL_USER
USER@DOMAIN	COL_USERDOMAIN
@@	COL_0

架构发现问题

如果 Kinesis Data Analytics 无法推断给定流式传输源的架构，会发生什么情况？

Kinesis Data Analytics 可以推断常见格式的架构，如使用 UTF-8 编码的 CSV 和 JSON。Kinesis Data Analytics 支持任何 UTF-8 编码的记录 (包括原始文本，如应用程序日志和记录) 并带有自定义列和行分隔符。如果 Kinesis Data Analytics 无法推断架构，您可以在控制台上使用架构编辑器手动定义架构 (或使用 API)。

如果您的数据没有遵循模式 (可使用架构编辑器指定)，您可以将架构定义为单个 VARCHAR(N) 类型的列，其中 N 是您希望记录包含的最大字符数。在此处，当数据位于应用程序内部流中后，您可以使用字符串和日期-时间操作来构造数据。有关示例，请参阅示例：转换 DateTime 值。

针对静态数据使用架构发现功能

> **Note**
>
> 2023 年 9 月 12 日之后，如果您尚未使用适用于 SQL 的 Kinesis Data Analytics，则将无法使用 Kinesis Data Firehose 作为来源创建新应用程序。有关更多信息，请参阅限制。

架构发现功能可以根据流中的数据或 Amazon S3 存储桶上存储的静态文件中的数据生成架构。假设您要为 Kinesis Data Analytics 应用程序生成架构以进行引用，或者实时流数据不可用。您可以对包含示例数据 (采用流或引用数据的预期格式) 的静态文件使用架构发现功能。Kinesis Data Analytics 可针对 Amazon S3 存储桶中所存的 JSON 或 CSV 文件中的示例数据运行架构发现。要针对数据文件使用架构发现功能，需使用控制台或指定了 DiscoverInputSchema 参数的 `S3Configuration` API。

使用控制台运行架构发现

要使用控制台对静态文件运行发现功能，请执行以下操作：

1. 向 S3 存储桶添加引用数据对象。
2. 在 Kinesis Data Analytics 控制台的应用程序主页面中选择 连接引用数据。
3. 提供存储桶、路径和 IAM 角色数据以访问包含引用数据的 Amazon S3 对象。
4. 选择 发现架构。

有关如何在控制台中添加引用数据和发现架构的更多信息，请参阅示例：在 应用程序中添加引用数据。

使用 API 运行架构发现

要使用 API 针对静态文件运行发现，您需要为 API 提供具有以下信息的 `S3Configuration` 结构：

- `BucketARN`：包含该文件的 Amazon S3 存储桶的 Amazon 资源名称 (ARN)。对于 Amazon S3 存储桶 ARN 的格式，请参阅 Amazon 资源名称 (ARN) 和 Amazon 服务命名空间：Amazon Simple Storage Service (Amazon S3)。
- `RoleARN`：具有 `AmazonS3ReadOnlyAccess` 策略的 IAM 角色的 ARN。有关如何将策略添加到角色的信息，请参阅修改角色。
- `FileKey`：对象的文件名称。

使用 **DiscoverInputSchema** API 根据 Amazon S3 对象生成架构

1. 确保您已完成 AWS CLI 设置。有关更多信息，请参阅“入门”部分中的步骤 2：设置 AWS Command Line Interface (AWS CLI)。
2. 使用以下内容创建名为 `data.csv` 的文件：

```
year,month,state,producer_type,energy_source,units,consumption
2001,1,AK,TotalElectricPowerIndustry,Coal,ShortTons,47615
2001,1,AK,ElectricGeneratorsElectricUtilities,Coal,ShortTons,16535
2001,1,AK,CombinedHeatandPowerElectricPower,Coal,ShortTons,22890
2001,1,AL,TotalElectricPowerIndustry,Coal,ShortTons,3020601
2001,1,AL,ElectricGeneratorsElectricUtilities,Coal,ShortTons,2987681
```

3. 登录到 Amazon S3 控制台，网址：https://console.aws.amazon.com/s3/。
4. 创建一个 Amazon S3 存储桶，并上传您创建的 `data.csv` 文件。请记下所创建存储桶的 ARN。有关创建 Amazon S3 存储桶并上传文件的信息，请参阅 Amazon Simple Storage Service 入门。
5. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。创建具有 `AmazonS3ReadOnlyAccess` 策略的角色。记下新角色的 ARN。有关创建角色的更多信息，请参阅创建向 Amazon 服务委托权限的角色。有关如何将策略添加到角色的信息，请参阅修改角色。
6. 在中运行以下`DiscoverInputSchema`命令 AWS CLI，将 ARN 替换为您的 Amazon S3 存储桶和 IAM 角色：

```
$aws kinesisanalytics discover-input-schema --s3-configuration '{ "RoleARN":
 "arn:aws:iam::123456789012:role/service-role/your-IAM-role", "BucketARN":
 "arn:aws:s3:::your-bucket-name", "FileKey": "data.csv" }'
```

7. 该响应应该类似于下列内容：

```
{
    "InputSchema": {
        "RecordEncoding": "UTF-8",
        "RecordColumns": [
            {
                "SqlType": "INTEGER",
                "Name": "COL_year"
            },
            {
                "SqlType": "INTEGER",
                "Name": "COL_month"
            },
```

```
            {
                "SqlType": "VARCHAR(4)",
                "Name": "state"
            },
            {
                "SqlType": "VARCHAR(64)",
                "Name": "producer_type"
            },
            {
                "SqlType": "VARCHAR(4)",
                "Name": "energy_source"
            },
            {
                "SqlType": "VARCHAR(16)",
                "Name": "units"
            },
            {
                "SqlType": "INTEGER",
                "Name": "consumption"
            }
        ],
        "RecordFormat": {
            "RecordFormatType": "CSV",
            "MappingParameters": {
                "CSVMappingParameters": {
                    "RecordRowDelimiter": "\r\n",
                    "RecordColumnDelimiter": ","
                }
            }
        }
    },
    "RawInputRecords": [
        "year,month,state,producer_type,energy_source,units,consumption
\r\n2001,1,AK,TotalElectricPowerIndustry,Coal,ShortTons,47615\r
\n2001,1,AK,ElectricGeneratorsElectricUtilities,Coal,ShortTons,16535\r
\n2001,1,AK,CombinedHeatandPowerElectricPower,Coal,ShortTons,22890\r
\n2001,1,AL,TotalElectricPowerIndustry,Coal,ShortTons,3020601\r
\n2001,1,AL,ElectricGeneratorsElectricUtilities,Coal,ShortTons,2987681"
    ],
    "ParsedInputRecords": [
        [
            null,
            null,
            "state",
```

```
            "producer_type",
            "energy_source",
            "units",
            null
        ],
        [
            "2001",
            "1",
            "AK",
            "TotalElectricPowerIndustry",
            "Coal",
            "ShortTons",
            "47615"
        ],
        [
            "2001",
            "1",
            "AK",
            "ElectricGeneratorsElectricUtilities",
            "Coal",
            "ShortTons",
            "16535"
        ],
        [
            "2001",
            "1",
            "AK",
            "CombinedHeatandPowerElectricPower",
            "Coal",
            "ShortTons",
            "22890"
        ],
        [
            "2001",
            "1",
            "AL",
            "TotalElectricPowerIndustry",
            "Coal",
            "ShortTons",
            "3020601"
        ],
        [
            "2001",
            "1",
```

```
            "AL",
            "ElectricGeneratorsElectricUtilities",
            "Coal",
            "ShortTons",
            "2987681"
        ]
    ]
}
```

使用 Lambda 函数预处理数据

> **Note**
>
> 2023 年 9 月 12 日之后，如果您尚未使用适用于 SQL 的 Kinesis Data Analytics，则将无法使用 Kinesis Data Firehose 作为来源创建新应用程序。有关更多信息，请参阅限制。

如果您的流中的数据需要格式转换、转换、丰富或过滤，则可以使用 AWS Lambda 函数对数据进行预处理。您可以在执行应用程序 SQL 代码或应用程序通过数据流创建架构之前执行此操作。

在以下情况下，Lambda 函数有助于预处理记录：

- 将其他格式 (例如 KPL 或 GZIP) 的记录转换为 Kinesis Data Analytics 可以分析的格式。Kinesis Data Analytics 目前支持 JSON 或 CSV 数据格式。
- 将数据扩展为聚合或异常检测等操作更易访问的格式。例如，如果多个数据值存储在同一字符串中，您可以将数据展开为多个分开的列。
- 利用其他 Amazon 服务进行数据扩充，例如外推或错误更正。
- 将复杂的字符串转换应用于记录字段。
- 用于整理数据的数据筛选。

使用 Lambda 函数预处理记录

在创建 Kinesis Data Analytics 应用程序时，您可在 连接到源 页面上启用 Lambda 预处理。

使用 Lambda 函数在 Kinesis Data Analytics 应用程序中预处理记录

1. 登录 AWS Management Console 并打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。

2. 在应用程序的 连接到源 页面上，在 使用 预处理记录 AWS Lambda部分选择 已启用。
3. 如需使用已创建的 Lambda 函数，请在 Lambda 函数 下拉列表中选择该函数。
4. 如需通过某个 Lambda 预处理模板创建新的 Lambda 函数，请从下拉列表中选择该模板。然后，选择 View <template name> in Lambda (在 Lambda 中查看 <模板名称>) 以编辑该函数。
5. 如需新建 Lambda 函数，请选择 新建。有关创建 Lambda 函数的信息，请参阅开发人员指南中的创建 HelloWorld Lambda 函数和浏览控制台。AWS Lambda
6. 选择要使用的 Lambda 函数的版本。要使用最新版本，请选择 $LATEST。

在选择或创建 Lambda 函数以预处理记录时，将在执行应用程序 SQL 代码或应用程序通过记录生成架构之前预处理记录。

Lambda 预处理权限

使用 Lambda 进行预处理，应用程序的 IAM 角色需要具有以下权限策略：

```
    {
      "Sid": "UseLambdaFunction",
      "Effect": "Allow",
      "Action": [
          "lambda:InvokeFunction",
          "lambda:GetFunctionConfiguration"
      ],
      "Resource": "<FunctionARN>"
  }
```

Lambda 预处理指标

您可以使用 Amazon CloudWatch 来监控 Lambda 调用的次数、处理的字节数、成功和失败次数等。有关 Kinesis Data Analytics Lambda 预处理发出的 CloudWatch 指标的信息，请参阅亚马逊 Kinesis Analytics 指标。

AWS Lambda 与 Kinesis 制作人库配合使用

Kinesis Producer Library（KPL）将较小的用户格式化记录聚合为较大的记录（最大为 1 MB），以更好地利用 Amazon Kinesis Data Streams 吞吐量。用于 Java 的 Kinesis 客户端库 (KCL) 支持取消聚合这些记录。但是，当你用作直播的使用者时，必须使用 AWS Lambda 特殊模块来解聚记录。

要获取必要的项目代码和说明，请参阅 Kinesis Producer 库解聚模块以获取相关信息。 AWS LambdaGitHub您可以使用此项目中的组件在 Java、Node.js 和 Python AWS Lambda 中处理 KPL 序列化数据。您也可以将这些组件用在多语言 KCL 应用程序中。

数据预处理事件输入数据模型/记录响应模型

要预处理记录，您的 Lambda 函数必须符合所需的事件输入数据和记录响应模型。

事件输入数据模型

Kinesis Data Analytics 会持续从你的 Kinesis 数据流或 Firehose 传输流中读取数据。对于检索的每一批记录，此服务管理如何将每个批次传送到您的 Lambda 函数。您的函数将接收到的记录列表作为输入。在您的函数中，您对列表进行迭代，并应用业务逻辑来完成您的预处理要求 (如数据格式转换或扩充)。

预处理函数的输入模型略有不同，具体取决于数据是从 Kinesis 数据流还是 Firehose 传输流接收。

如果源是 Firehose 传送流，则事件输入数据模型如下所示：

Kinesis Data Firehose 请求数据模型

字段	描述
`invocationId`	Lambda 调用 ID (随机 GUID)。
`applicationArn`	Kinesis Data Analytics 应用程序 Amazon 资源名称 (ARN)
`streamArn`	传输流 ARN
记录	

字段	描述	
`recordId`	记录 ID (随机 GUID)	
`kinesisFirehoseRecordMetadata`		

字段	描述
`approximateArriva`	传输流记录大致到达时间

字段	描述
字段	描述
字段	描述
Timestan	
data	Base64 编码的源记录负载

以下示例显示来自 Firehose 传输流的输入：

```
{
   "invocationId":"00540a87-5050-496a-84e4-e7d92bbaf5e2",
   "applicationArn":"arn:aws:kinesisanalytics:us-east-1:12345678911:application/lambda-
test",
   "streamArn":"arn:aws:firehose:us-east-1:AAAAAAAAAAAA:deliverystream/lambda-test",
   "records":[
      {
         "recordId":"49572672223665514422805246926656954630972486059535892482",
         "data":"aGVsbG8gd29ybGQ=",
         "kinesisFirehoseRecordMetadata":{
            "approximateArrivalTimestamp":1520280173
         }
      }
   ]
}
```

如果源是 Kinesis 数据流，事件输入数据模型如下所示：

Kinesis 流请求数据模型

字段	描述
invocationId	Lambda 调用 ID (随机 GUID)。
applicationArn	Kinesis Data Analytics 应用程序 ARN
streamArn	传输流 ARN

字段	描述
记录	

字段	描述
recordId	基于 Kinesis 记录序列号的记录 ID
kinesisStreamRecordMetadata	字段 描述 sequenceNumber 从 Kinesis 流记录中得到的序列号 partitionKey 从 Kinesis 流记录中得到的分区键 shardId 从 Kinesis 流记录中得到的 ShardId approximateArrivalTimestamp 传输流记录大致到达时间
数据	Base64 编码的源记录负载

以下示例显示来自 Kinesis 数据流的输入：

```
{
  "invocationId": "00540a87-5050-496a-84e4-e7d92bbaf5e2",
  "applicationArn": "arn:aws:kinesisanalytics:us-east-1:12345678911:application/lambda-test",
  "streamArn": "arn:aws:kinesis:us-east-1:AAAAAAAAAAAA:stream/lambda-test",
  "records": [
    {
      "recordId": "49572672223665514422805246926656954630972486059535892482",
      "data": "aGVsbG8gd29ybGQ=",
```

```
      "kinesisStreamRecordMetadata":{
            "shardId" :"shardId-000000000003",
            "partitionKey":"7400791606",

 "sequenceNumber":"49572672223665514422805246926656954630972486059535892482",
            "approximateArrivalTimestamp":1520280173
         }
    }
  ]
}
```

记录响应模型

必须返回发送到 Lambda 函数，并从您的 Lambda 预处理函数返回的所有记录 (带记录 ID)。这些记录必须包含以下参数，否则，Kinesis Data Analytics 将拒绝这些记录，并将其视为失败的数据预处理。可对记录的数据负载部分进行转换，以满足预处理要求。

响应数据模型

记录

字段	描述
`recordId`	在调用期间，记录 ID 从 Kinesis Data Analytics 传送到 Lambda。转换后的记录必须包含相同记录 ID。原始记录的 ID 和转换记录的 ID 之间如果有不匹配，将被视为数据预处理失败。
`result`	记录的数据转换的状态。可能的值包括： • `Ok`：记录已成功转换。Kinesis Data Analytics 会摄取 SQL 处理记录。 • `Dropped`：您的处理逻辑有意丢弃了此记录。Kinesis Data Analytics 会丢弃 SQL 处理记录。对于 `Dropped` 记录，数据负载字段是可选的。 • `ProcessingFailed` ：无法转换记录。Kinesis Data Analytics 认为您的 Lambda 函数未成功处理它，并在错误流中写入一条错误。有关错误流的更多信息，请参阅错误处

字段	描述
	理。对于 ProcessingFailed 记录，数据负载字段是可选的。
data	转换后的数据负载（使用 base64 编码之后）。如果应用程序提取数据格式为 JSON，则每个数据负载可能包含多个 JSON 文档。或者，如果应用程序提取数据格式为 CSV，则每个数据负载可能包含多个 CSV 行 (在每一行中指定行分隔符)。Kinesis Data Analytics 服务将成功分析并处理同一数据负载中的多个 JSON 文档或 CSV 行数据。

以下示例显示来自 Lambda 函数的输出：

```
{
  "records": [
    {
      "recordId": "49572672223665514422805246926656954630972486059535892482",
      "result": "Ok",
      "data": "SEVMTE8gV09STEQ="
    }
  ]
}
```

常见的数据预处理失败情况

以下是预处理失败的常见原因。

- 并非批次中发送到 Lambda 函数的所有记录（具有记录 ID）都会返回到 Kinesis Data Analytics 服务。
- 响应中缺少记录 ID、状态或数据负载字段。对于 Dropped 或 ProcessingFailed 记录，数据负载字段是可选的。
- Lambda 函数的超时时间不足以预处理数据。
- Lambda 函数的响应时间超出了 AWS Lambda 服务施加的响应限制。

如果数据预处理失败，Kinesis Data Analytics 会继续对同一组记录重试 Lambda 调用，直到成功为止。您可以监控以下 CloudWatch 指标以深入了解故障。

- Kinesis Data Analytics 应用程序 `MillisBehindLatest`：指示应用程序从流式传输源中读取时的滞后时间。
- Kinesis Data Analytics `InputPreprocessing` CloudWatch 应用程序指标：表示成功和失败的数量以及其他统计数据。有关更多信息，请参阅 Amazon Kinesis Analytics 指标。
- AWS Lambda 函数 CloudWatch 指标和日志。

创建 Lambda 函数以进行预处理

在将记录提取到应用程序时，您的 Amazon Kinesis Data Analytics 应用程序可以使用 Lambda 函数预处理记录。Kinesis Data Analytics 在控制台上提供以下模板以作为数据预处理起点。

主题

- 使用 Node.js 创建预处理 Lambda 函数
- 使用 Python 创建预处理 Lambda 函数
- 使用 Java 创建预处理 Lambda 函数
- 使用 .NET 创建预处理 Lambda 函数

使用 Node.js 创建预处理 Lambda 函数

在 Kinesis Data Analytics 控制台上提供了以下模板以使用 Node.js 创建预处理 Lambda 函数：

Lambda 蓝图	语言和版本	描述
通用 Kinesis Data Analytics 输入处理	Node.js 6.10	Kinesis Data Analytics 记录预处理器，它将 JSON 或 CSV 记录作为输入接收，然后返回这些记录以及处理状态。使用此处理器作为自定义转换逻辑的起点。
压缩输入处理	Node.js 6.10	Kinesis Data Analytics 记录处理器，它接收压缩（GZIP 或 Deflate 压缩）JSON 或 CSV 记录以作为输入，并返回解压缩的记录以及处理状态。

使用 Python 创建预处理 Lambda 函数

在控制台上提供了以下模板以使用 Python 创建预处理 Lambda 函数：

Lambda 蓝图	语言和版本	描述
通用 Kinesis Analytics 输入处理	Python 2.7	Kinesis Data Analytics 记录预处理器，它将 JSON 或 CSV 记录作为输入接收，然后返回这些记录以及处理状态。使用此处理器作为自定义转换逻辑的起点。
KPL 输入处理	Python 2.7	Kinesis Data Analytics 记录处理器，它接收 JSON 或 CSV 记录的 Kinesis 创建器库 (KPL) 聚合以作为输入，并返回取消聚合的记录以及处理状态。

使用 Java 创建预处理 Lambda 函数

要使用 Java 创建 Lambda 函数以预处理记录，请使用 Java 事件类。

以下代码说明了一个使用 Java 的示例 Lambda 函数（用于预处理记录）：

```
public class LambdaFunctionHandler implements
        RequestHandler<KinesisAnalyticsStreamsInputPreprocessingEvent,
 KinesisAnalyticsInputPreprocessingResponse> {

    @Override
    public KinesisAnalyticsInputPreprocessingResponse handleRequest(
            KinesisAnalyticsStreamsInputPreprocessingEvent event, Context context) {
        context.getLogger().log("InvocatonId is : " + event.invocationId);
        context.getLogger().log("StreamArn is : " + event.streamArn);
        context.getLogger().log("ApplicationArn is : " + event.applicationArn);

        List<KinesisAnalyticsInputPreprocessingResponse.Record> records = new
 ArrayList<KinesisAnalyticsInputPreprocessingResponse.Record>();
        KinesisAnalyticsInputPreprocessingResponse response = new
 KinesisAnalyticsInputPreprocessingResponse(records);

        event.records.stream().forEach(record -> {
            context.getLogger().log("recordId is : " + record.recordId);
            context.getLogger().log("record aat is :" +
 record.kinesisStreamRecordMetadata.approximateArrivalTimestamp);
             // Add your record.data pre-processing logic here.
```

```
            // response.records.add(new Record(record.recordId,
 KinesisAnalyticsInputPreprocessingResult.Ok, <preprocessedrecordData>));
        });
        return response;
    }

}
```

使用 .NET 创建预处理 Lambda 函数

要使用 .NET 创建 Lambda 函数以预处理记录，请使用 .NET 事件类。

以下代码说明了一个使用 C# 的示例 Lambda 函数（用于预处理记录）：

```
public class Function
    {
        public KinesisAnalyticsInputPreprocessingResponse
 FunctionHandler(KinesisAnalyticsStreamsInputPreprocessingEvent evnt, ILambdaContext
 context)
        {
            context.Logger.LogLine($"InvocationId: {evnt.InvocationId}");
            context.Logger.LogLine($"StreamArn: {evnt.StreamArn}");
            context.Logger.LogLine($"ApplicationArn: {evnt.ApplicationArn}");

            var response = new KinesisAnalyticsInputPreprocessingResponse
            {
                Records = new List<KinesisAnalyticsInputPreprocessingResponse.Record>()
            };

            foreach (var record in evnt.Records)
            {
                context.Logger.LogLine($"\tRecordId: {record.RecordId}");
                context.Logger.LogLine($"\tShardId: {record.RecordMetadata.ShardId}");
                context.Logger.LogLine($"\tPartitionKey:
 {record.RecordMetadata.PartitionKey}");
                context.Logger.LogLine($"\tRecord ApproximateArrivalTime:
 {record.RecordMetadata.ApproximateArrivalTimestamp}");
                context.Logger.LogLine($"\tData: {record.DecodeData()}");

                // Add your record preprocessig logic here.

                var preprocessedRecord = new
 KinesisAnalyticsInputPreprocessingResponse.Record
                {
```

```
                    RecordId = record.RecordId,
                    Result = KinesisAnalyticsInputPreprocessingResponse.OK
                };
                preprocessedRecord.EncodeData(record.DecodeData().ToUpperInvariant());
                response.Records.Add(preprocessedRecord);
            }
            return response;
        }
    }
```

有关使用 .NET 创建 Lambda 函数以进行预处理或作为目标的更多信息，请参阅 `Amazon.Lambda.KinesisAnalyticsEvents`。

并行处理输入流以增加吞吐量

> **Note**
>
> 2023 年 9 月 12 日之后，如果您尚未使用适用于 SQL 的 Kinesis Data Analytics，则将无法使用 Kinesis Data Firehose 作为来源创建新应用程序。有关更多信息，请参阅限制。

Amazon Kinesis Data Analytics 应用程序可以支持多个应用程序内部输入流，以将应用程序扩展到超出单个应用程序内部输入流的吞吐量。有关应用程序内部输入流的更多信息，请参阅 适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理。

几乎在所有情况下，Amazon Kinesis Data Analytics 都会扩展您的应用程序，以处理馈送到您的应用程序的 Kinesis 流或 Firehose 源流的容量。但是，如果您的源流的吞吐量超出单个应用程序内部输入流的吞吐量，您可显式增加您的应用程序使用的应用程序内部输入流的数量。您需使用 `InputParallelism` 参数执行此操作。

如果 `InputParallelism` 参数大于 1，则 Amazon Kinesis Data Analytics 在应用程序内部流之间平均拆分源流的分区。例如，如果您的源流具有 50 个分片，并且您已将 `InputParallelism` 设置为 2，则每个应用程序内部输入流都将收到来自 25 个源流分片的输入。

当您增加应用程序内部流的数量后，您的应用程序必须显式访问每个流中的数据。有关通过代码访问多个应用程序内部流的信息，请参阅在 Amazon Kinesis Data Analytics 应用程序中访问单独的应用程序内部流。

尽管 Kinesis Data Streams 和 Firehose 流分片都以相同的方式划分到应用程序内部流，但它们在应用程序中的显示方式有所不同：

- Kinesis 数据流中的记录包含一个 shard_id 字段，可用于指定记录的源分片。
- 来自 Firehose 传输流的记录不包含用于标识记录源分片或分区的字段。这是因为 Firehose 会将这些信息从您的应用程序中抽象出来。

评估是否增加您的应用程序内部输入流的数量

在大多数情况下，单个应用程序内部输入流可处理单个源流的吞吐量，具体取决于输入流的复杂度和数据大小。要确定是否需要增加应用程序内输入流的数量，您可以在 Amazon CloudWatch 中监控InputBytes和MillisBehindLatest指标。

如果 InputBytes 指标大于 100 MB/秒（或您预期该指标将大于此速率），这可能会使 MillisBehindLatest 增大并导致应用程序问题的影响扩大。为解决此问题，我们建议您为应用程序选择以下语言：

- 如果您的应用程序的扩展需求超过 100 MB/秒，请使用多个流和适用于 SQL 应用程序的 Kinesis Data Analytics。
- 如果您希望使用单个流和应用程序，请使用适用于 Java 应用程序的 Kinesis Data Analytics。

如果 MillisBehindLatest 指标具有以下任一特征，则应调高您的应用程序的 InputParallelism 设置：

- MillisBehindLatest 指标逐渐增大，指示您的应用程序落后于流中的最新数据。
- MillisBehindLatest 指标始终高于 1000 (1 秒)。

如果满足以下条件，您无需调高您的应用程序的 InputParallelism 设置：

- MillisBehindLatest 指标逐渐减小，指示您的应用程序跟上了流中的最新数据。
- MillisBehindLatest 指标小于 1000 (1 秒)。

有关使用的更多信息 CloudWatch，请参阅《CloudWatch 用户指南》。

实施多个应用程序内部输入流

如果应用程序是使用 CreateApplication 创建的，您可以设置应用程序内部输入流的数量。您应在使用UpdateApplication创建应用程序之后设置此数量。

> **Note**
>
> 你只能使用 Amazon Kinesis Data Analytics API 或 AWS CLI设置 `InputParallelism`。您不能使用来设置此设置 AWS Management Console。有关设置的信息 AWS CLI，请参阅步骤 2：设置 AWS Command Line Interface (AWS CLI)。

设置新应用程序的输入流计数

以下示例说明了如何使用 `CreateApplication` API 操作将新应用程序的输入流计数设置为 2。

有关 `CreateApplication` 的更多信息，请参阅CreateApplication。

```
{
   "ApplicationCode": "<The SQL code the new application will run on the input
 stream>",
   "ApplicationDescription": "<A friendly description for the new application>",
   "ApplicationName": "<The name for the new application>",
   "Inputs": [
    {
      "InputId": "ID for the new input stream",
      "InputParallelism": {
        "Count": 2
    }],
   "Outputs": [ ... ],
 }]
}
```

设置现有应用程序的输入流计数

以下示例说明了如何使用 `UpdateApplication` API 操作将现有应用程序的输入流计数设置为 2。

有关 `Update_Application` 的更多信息，请参阅UpdateApplication。

```
{
   "InputUpdates": [
      {
         "InputId": "yourInputId",
         "InputParallelismUpdate": {
            "CountUpdate": 2
         }
```

```
      }
   ],
}
```

在 Amazon Kinesis Data Analytics 应用程序中访问单独的应用程序内部流

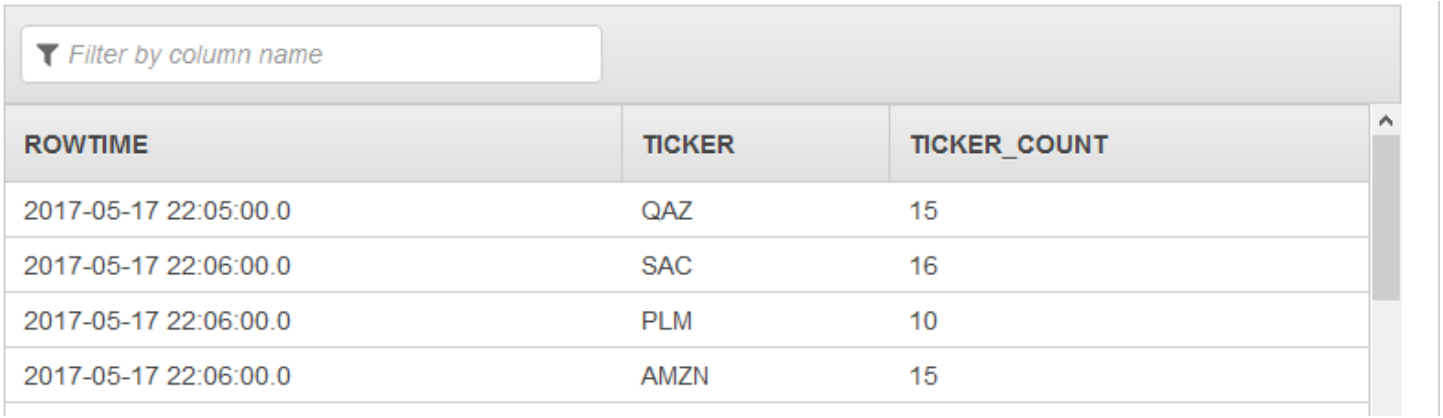
要使用您的应用程序中的多个应用程序内部输入流，您必须从不同的流中显式选择。以下代码示例说明了如何在创建于“入门”教程中的应用程序中查询多个输入流。

在以下示例中，每个源流在组合到名为 [的单个应用程序内部流之前先通过](#) COUNTin_application_stream001 进行了聚合。预先聚合源流有助于确保组合的应用程序内部流可处理来自多个流的流量而不会过载。

> **Note**
>
> 要运行此示例并获得来自应用程序内部输入流的结果，请更新源流中的分片数和应用程序中的 InputParallelism 参数。

```
CREATE OR REPLACE STREAM in_application_stream_001 (
    ticker VARCHAR(64),
    ticker_count INTEGER
);

CREATE OR REPLACE PUMP pump001 AS
INSERT INTO in_application_stream_001
SELECT STREAM ticker_symbol, COUNT(ticker_symbol)
FROM source_sql_stream_001
GROUP BY STEP(source_sql_stream_001.rowtime BY INTERVAL '60' SECOND),
    ticker_symbol;

CREATE OR REPLACE PUMP pump002 AS
INSERT INTO in_application_stream_001
SELECT STREAM ticker_symbol, COUNT(ticker_symbol)
FROM source_sql_stream_002
GROUP BY STEP(source_sql_stream_002.rowtime BY INTERVAL '60' SECOND),
    ticker_symbol;
```

前面的代码示例将在 in_application_stream001 中生成类似于下面的输出：



其他注意事项

在使用多个输入流时，请注意以下事项：

- 应用程序内部输入流的最大数量为 64。
- 应用程序内部输入流在应用程序的输入流的分片之间均匀分配。
- 通过添加应用程序内部流获得的性能改进无法线性扩展。也就是说，使应用程序内部流的数量加倍不会使吞吐量加倍。对于典型行大小，每个应用程序内部流可实现每秒大约 5,000 到 15,000 行的吞吐量。通过将应用程序内部流计数增加到 10，您可以实现每秒 20,000 到 30,000 行的吞吐量。吞吐量流速取决于输入流中的字段的计数、数据类型和数据大小。
- 某些聚合函数（如 AVG）在应用于分区到不同分片中的输入流时可能生成意外结果。由于您需要在将各个分片组合到聚合流中之前对它们运行聚合操作，因此结果可能向包含更多记录的流加权。
- 如果您增加了输入流的数量后，应用程序性能仍然很差 (反映在较高的 `MillisBehindLatest` 指标上)，您可能已达到 Kinesis 处理单元 (KPU) 的限制。有关更多信息，请参阅 自动扩展应用程序以提高吞吐量。

应用程序代码

应用程序代码是处理输入和生产输出的一系列 SQL 语句。这些 SQL 语句运行于应用程序内部流和引用表中。有关更多信息，请参阅 适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理。

有关 Kinesis Data Analytics 支持的 SQL 语言元素的信息，请参阅 Amazon Kinesis Data Analytics SQL 参考。

在关系数据库中，可使用 INSERT 语句添加记录和使用 SELECT 语句查询数据来处理表。在 Amazon Kinesis Data Analytics 中，您可以处理流。可以编写 SQL 语句来查询这些流。查询一个应用程序内部流所获得的结果始终将发送到另一个应用程序内部流。在执行复杂分析时，您可以创建多个应用程序内

部流来保存中间分析的结果。最后，将应用程序输出配置为将最终分析的结果（来自一个或多个应用程序内部流）保存到外部目标。概括来说，以下是用于编写应用程序代码的典型模式：

- SELECT 语句始终用于 INSERT 语句的上下文中。即，在选择行时，您将结果插入另一个应用程序内部流中。
- INSERT 语句始终用于数据泵的上下文中。即，您使用数据泵对应用程序内部流进行写入。

以下示例应用程序代码读取一个应用程序内部流 (SOURCE_SQL_STREAM_001) 中的记录，并将其写入另一个应用程序内部流 (DESTINATION_SQL_STREAM)。您可以使用数据泵将记录插入应用程序内部流中，如下所示：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
                                                   change DOUBLE,
                                                   price DOUBLE);
-- Create a pump and insert into output stream.
CREATE OR REPLACE PUMP "STREAM_PUMP" AS

  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM ticker_symbol, change,price
    FROM   "SOURCE_SQL_STREAM_001";
```

> **Note**
>
> 为流名称和列名称指定的标识符需遵循标准 SQL 约定。例如，如果您为标识符加上引号，则可使标识符区分大小写。如果没有这样做，则标识符将默认为大写。有关标识符的更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的标识符。

您的应用程序代码可包含多个 SQL 语句。例如：

- 您可以按顺序编写 SQL 查询，其中一个 SQL 语句的结果将馈送到下一个 SQL 语句。
- 您也可以编写彼此单独运行的 SQL 查询。例如，您可以编写两个 SQL 语句，它们查询同一应用程序内部流，但将输出发送到不同的应用程序内部流。随后，您可以单独查询新创建的应用程序内部流。

您可以创建应用程序内部流来保存中间结果。可使用数据泵将数据插入应用程序内部流。有关更多信息，请参阅 应用程序内部流和数据泵。

如果您添加一个应用程序内部引用表，则可编写 SQL 以联接应用程序内部流和引用表中的数据。有关更多信息，请参阅 示例：在 应用程序中添加引用数据。

根据应用程序的输出配置，Amazon Kinesis Data Analytics 将特定应用程序内部流中的数据写入到外部目标中。确保您的应用程序代码写入输出配置中指定的应用程序内部流。

有关更多信息，请参阅以下主题：

- 流式 SQL 概念
- Amazon Kinesis Data Analytics SQL 参考

配置应用程序输出

在您的应用程序代码中，将 SQL 语句的输出写入一个或多个应用程序内部流。您可以选择向应用程序添加输出配置。以将写入应用程序内流的所有内容保存到外部目标，例如 Amazon Kinesis 数据流、Firehose 传输流或函数。 AWS Lambda

可以用来永久保存应用程序输出的外部目标的数量有限制。有关更多信息，请参阅Limits。

> **Note**
>
> 建议用一个外部目标来永久保存应用程序内部错误流数据，以方便调查错误。

在所有这些输出配置中，可以提供以下内容：

- 应用程序内部流名称 - 要永久保存到外部目标的流。

  Kinesis Data Firehose 查找在输出配置中指定的应用程序内部流。（流名称区分大小写，并且必须完全匹配）。确保应用程序代码创建了这一应用程序内部流。
- 外部目标-您可以将数据保存到 Kinesis 数据流、Firehose 传输流或 Lambda 函数中。您需要提供流或函数的 Amazon 资源名称 (ARN)。您还要提供 Kinesis data Analytics 代表您写入到流或函数时代入的 IAM 角色。您描述在写入到外部目标时 Kinesis Data Analytics 使用的记录格式（JSON、CSV）。

如果 Kinesis Data Analytics 无法写入到流或 Lambda 目标，该服务将继续无限期地进行尝试。这将产生反向压力，导致您的应用程序滞后。如果不能解决这一问题，您的应用程序最终将停止处理新数据。

您可以监控 Kinesis Data Analytics 指标并设置故障警报。有关指标和警报的更多信息，请参阅使用亚马逊 CloudWatch指标和创建亚马逊 CloudWatch警报。

您可以使用 AWS Management Console配置应用程序输出。控制台将调用 API 以保存配置。

使用创建输出 AWS CLI

此部分介绍如何为 `CreateApplication` 或 `AddApplicationOutput` 操作创建请求正文的 `Outputs` 部分。

创建 Kinesis 流输出

以下 JSON 代码段显示用于创建 Amazon Kinesis data stream 目标的 `CreateApplication` 请求正文中的 `Outputs` 部分。

```
"Outputs": [
   {
       "DestinationSchema": {
           "RecordFormatType": "string"
       },
       "KinesisStreamsOutput": {
           "ResourceARN": "string",
           "RoleARN": "string"
       },
       "Name": "string"
   }

]
```

创建 Firehose 传送流输出

以下 JSON 片段显示了`CreateApplication`请求正`Outputs`文中用于创建 Amazon Data Firehose 传输流目标的部分。

```
"Outputs": [
   {
       "DestinationSchema": {
           "RecordFormatType": "string"
       },
       "KinesisFirehoseOutput": {
           "ResourceARN": "string",
           "RoleARN": "string"
```

```
        },
        "Name": "string"
    }
]
```

创建 Lambda 函数输出

以下 JSON 片段显示了`CreateApplication`请求正文中用于创建 AWS Lambda 函数目标的`Outputs`部分。

```
"Outputs": [
    {
        "DestinationSchema": {
            "RecordFormatType": "string"
        },
        "LambdaOutput": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "Name": "string"
    }
]
```

使用 Lambda 函数作为输出

使用 AWS Lambda 作为目标可以更轻松地对 SQL 结果进行后处理，然后再将其发送到最终目标。常见的后处理任务包括：

- 将多行聚合为一条记录
- 将当前结果与过去的结果相结合以解决迟到数据的问题
- 根据信息类型传输到不同的目标
- 记录格式转换 (如转换为 Protobuf)
- 字符串操作或转换
- 分析处理后的数据扩充
- 地理空间使用案例的自定义处理
- 数据加密

Lambda 函数可以向各种 AWS 服务和其他目标提供分析信息，包括：

- Amazon Simple Storage Service (Amazon S3)
- 自定义 API
- Amazon DynamoDB
- Apache Aurora
- Amazon Redshift
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- Amazon CloudWatch

有关创建 Lambda 应用程序的更多信息，请参阅AWS Lambda入门。

主题

- Lambda 作为输出权限
- Lambda 作为输出指标
- Lambda 作为输出事件输入数据模型和记录响应模型
- Lambda 输出调用频率
- 添加 &LAM; 函数作为输出
- 常见的 Lambda 作为输出的故障
- 为应用程序目标创建 &LAM; 函数

Lambda 作为输出权限

要使用 Lambda 作为输出，应用程序的 Lambda 输出 IAM 角色需要以下权限策略：

```
{
   "Sid": "UseLambdaFunction",
   "Effect": "Allow",
   "Action": [
       "lambda:InvokeFunction",
       "lambda:GetFunctionConfiguration"
   ],
   "Resource": "FunctionARN"
}
```

Lambda 作为输出指标

您可以使用 Amazon CloudWatch 监控发送的字节数、成功和失败等。有关 Kinesis Data Analytics 使用 Lambda 作为输出发出的 CloudWatch 指标的信息，请参阅亚马逊 Kinesis Analytics 指标。

Lambda 作为输出事件输入数据模型和记录响应模型

要发送 Kinesis Data Analytics 输出记录，您的 Lambda 函数必须符合所需的事件输入数据和记录响应模型。

事件输入数据模型

使用以下请求模型持续将输出记录从应用程序发送到 以作为输出函数。在您的函数中，遍历列表并应用业务逻辑来完成输出要求 (例如，在将数据发送到最终目标之前先进行数据转换)。

字段	描述
`invocationId`	&LAM; 调用 ID (随机 GUID)。
`applicationArn`	数据分析应用程序 Amazon 资源名称 (ARN)。
记录	

字段	描述
`recordId`	记录 ID (随机 GUID)
`lambdaDeliveryRecordMetadata`	

字段	描述
`retryHir`	传输重试次数

数据	Base64 编码的输出记录负载

> **Note**
>
> `retryHint` 是一个每次传输失败时都会增加的值。该值不会持久不变，并在应用程序中断时重置。

记录响应模型

作为输出函数发送到您的 Lambda 的每个记录（具有记录 ID）必须使用 `Ok` 或 `DeliveryFailed` 进行确认，并且必须包含以下参数。否则， 将其视为传输失败。

记录

字段	描述
`recordId`	在调用期间，记录 ID 从 Kinesis Data Analytics 传送到 Lambda。如果原始记录的 ID 和确认记录的 ID 之间不匹配，就会被视为传输失败。
`result`	记录的传输状态。以下是可能的值： • `Ok`：记录成功转换并发送到最终目标。Kinesis Data Analytics 会摄取 SQL 处理记录。 • `DeliveryFailed` ：&LAM; 作为输出函数未将记录成功地传输到最终目标。不断重试将传输失败的记录发送到 作为输出函数。

Lambda 输出调用频率

Kinesis Data Analytics 数据分析应用程序缓存输出记录，并频繁调用 AWS Lambda 目标函数。

- 如果将记录作为翻滚窗口发送到数据分析应用程序内的目标应用程序内流，则每次翻滚窗口触发器都会调用 AWS Lambda 目标函数。例如，如果使用 60 秒的滚动窗口将记录发送到目标应用程序内部流，则每 60 秒调用一次 函数。
- 如果在该应用程序中使用持续查询或滑动窗口将记录发送到目标应用程序内部流，则大约每秒调用一次 目标函数。

> **Note**
>
> 每 &LAM; 函数调用请求负载大小限制适用。超出这些限制将导致拆分输出记录，并在多个 函数调用中进行发送。

添加 &LAM; 函数作为输出

以下过程说明了如何添加 Lambda 函数以作为 Kinesis Data Analytics 应用程序输出。

1. 登录 AWS Management Console 并打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择列表中的应用程序，然后选择 Application details。
3. 在 Destination 部分，选择 Connect new destination。
4. 对于 Destination 项，选择 AWS Lambda function。
5. 在向 AWS Lambda传递记录部分中，选择现有Lambda 函数或创建新函数。
6. 如果您正在创建一个新的 &LAM; 函数，请执行以下操作：
   a. 选择提供的模板之一。有关更多信息，请参阅 为应用程序目标创建 &LAM; 函数。
   b. 将在新的浏览器选项卡中打开 Create Function (创建函数) 页。在 Name (名称) 框中，为函数指定一个有意义的名称（例如，**`myLambdaFunction`**）。
   c. 针对您的应用程序用后处理功能更新模板。有关创建 Lambda 函数的信息，请参阅AWS Lambda 开发人员指南中的入门。
   d. 在 Kinesis Data Analytics 控制台上的 Lambda 函数列表中，选择刚创建的 Lambda 函数。Lambda 函数版本选择最新。
7. 在 In-application stream 部分，选择 Choose an existing in-application stream。对于 In-application stream name，选择应用程序的输出流。选定输出流的结果将发送到 输出函数。
8. 保持表单其余部分为默认值，然后选择 Save and continue。

您的应用程序现在将记录从应用程序内部流发送到 函数。您可以在 Amazon CloudWatch 控制台中查看默认模板的结果。监控 `AWS/KinesisAnalytics/LambdaDelivery.OkRecords` 指标，查看传输给 &LAM; 函数的记录数。

常见的 Lambda 作为输出的故障

传输到 &LAM; 函数失败的常见原因如下。

- 并非批次中发送到 Lambda 函数的所有记录（具有记录 ID）都会返回到 服务。
- 响应中缺少记录 ID 或状态字段。
- &LAM; 函数超时不足以在 &LAM; 函数内完成业务逻辑。
- Lambda 函数中的业务逻辑不会捕获所有错误，导致因未处理的异常而产生超时和反向压力。这些消息通常称为“毒丸”消息。

如果数据传输失败， 继续对同一组记录重试 调用，直到成功为止。要深入了解故障，您可以监控以下 CloudWatch 指标：

- Kinesis Data Analytics 应用程序 Lambda 作为 CloudWatch 输出指标：表示成功和失败的数量以及其他统计数据。有关更多信息，请参阅 Amazon Kinesis Analytics 指标。
- AWS Lambda 函数 CloudWatch 指标和日志。

为应用程序目标创建 &LAM; 函数

您的 Kinesis Data Analytics 应用程序 AWS Lambda 可以使用函数作为输出。 提供了一些模板以创建用作应用程序目标的 函数。可以将这些模板作为应用程序输出后处理的起点。

主题

- 使用 Node.js 创建 &LAM; 函数目标
- 使用 Python 创建 &LAM; 函数目标
- 使用 Java 创建 &LAM; 函数目标
- 使用 .NET 创建 &LAM; 函数目标

使用 Node.js 创建 &LAM; 函数目标

在控制台上提供了以下模板以使用 Node.js 创建目标 函数：

&LAM; 作为输出蓝图	语言和版本	描述
`kinesis-analytics-output`	Node.js 12.x	将输出记录从 应用程序传输到自定义目标。

使用 Python 创建 &LAM; 函数目标

在控制台上提供了以下模板以使用 Python 创建目标 函数：

&LAM; 作为输出蓝图	语言和版本	描述
`kinesis-analytics-output-sns`	Python 2.7	将 Kinesis Data Analytics 应用程序的输出记录传送到 Amazon SNS。
`kinesis-analytics-output-ddb`	Python 2.7	将 Kinesis Data Analytics 应用程序的输出记录传送到 Amazon DynamoDB。

使用 Java 创建 &LAM; 函数目标

要使用 Java 创建目标 &LAM; 函数，请使用 Java 事件类。

以下代码说明了一个使用 Java 的示例目标 &LAM; 函数：

```
public class LambdaFunctionHandler
        implements RequestHandler<KinesisAnalyticsOutputDeliveryEvent,
 KinesisAnalyticsOutputDeliveryResponse> {

    @Override
    public KinesisAnalyticsOutputDeliveryResponse
 handleRequest(KinesisAnalyticsOutputDeliveryEvent event,
            Context context) {
        context.getLogger().log("InvocatonId is : " + event.invocationId);
        context.getLogger().log("ApplicationArn is : " + event.applicationArn);

        List<KinesisAnalyticsOutputDeliveryResponse.Record> records = new
 ArrayList<KinesisAnalyticsOutputDeliveryResponse.Record>();
        KinesisAnalyticsOutputDeliveryResponse response = new
 KinesisAnalyticsOutputDeliveryResponse(records);

        event.records.stream().forEach(record -> {
            context.getLogger().log("recordId is : " + record.recordId);
            context.getLogger().log("record retryHint is :" +
 record.lambdaDeliveryRecordMetadata.retryHint);
```

```
            // Add logic here to transform and send the record to final destination of
 your choice.
            response.records.add(new Record(record.recordId,
 KinesisAnalyticsOutputDeliveryResponse.Result.Ok));
        });
        return response;
    }

}
```

使用 .NET 创建 &LAM; 函数目标

要使用 .NET 创建目标 &LAM; 函数，请使用 .NET 事件类。

以下代码说明了一个使用 C# 的示例目标 &LAM; 函数：

```
public class Function
    {
        public KinesisAnalyticsOutputDeliveryResponse
 FunctionHandler(KinesisAnalyticsOutputDeliveryEvent evnt, ILambdaContext context)
        {
            context.Logger.LogLine($"InvocationId: {evnt.InvocationId}");
            context.Logger.LogLine($"ApplicationArn: {evnt.ApplicationArn}");

            var response = new KinesisAnalyticsOutputDeliveryResponse
            {
                Records = new List<KinesisAnalyticsOutputDeliveryResponse.Record>()
            };

            foreach (var record in evnt.Records)
            {
                context.Logger.LogLine($"\tRecordId: {record.RecordId}");
                context.Logger.LogLine($"\tRetryHint:
 {record.RecordMetadata.RetryHint}");
                context.Logger.LogLine($"\tData: {record.DecodeData()}");

                // Add logic here to send to the record to final destination of your
 choice.

                var deliveredRecord = new KinesisAnalyticsOutputDeliveryResponse.Record
                {
                    RecordId = record.RecordId,
                    Result = KinesisAnalyticsOutputDeliveryResponse.OK
```

```
                };
                response.Records.Add(deliveredRecord);
            }
            return response;
        }
    }
```

有关使用 .NET 创建 Lambda 函数以进行预处理或作为目标的更多信息，请参阅[Amazon.Lambda.KinesisAnalyticsEvents](#)。

将应用程序输出永久保存到外部目标的传输模型

Amazon Kinesis Data Analytics 使用“至少一次”传输模型将应用程序输出传输到配置的目标。在应用程序运行时，Kinesis Data Analytics 使用内部检查点。这些检查点是指将输出记录传输到目标并且未丢失数据的时间点。该服务根据需要使用检查点以确保至少向配置的目标传输一次应用程序输出。

在正常情况下，您的应用程序会持续处理传入的数据。Kinesis Data Analytics 将输出写入配置的目的地，例如 Kinesis 数据流或 Firehose 传输流。不过，您的应用程序偶尔可能会中断，例如：

- 您选择停止应用程序并稍后重新启动。
- 您删除 Kinesis Data Analytics 在将应用程序输出写入到配置的目标时所需的 IAM 角色。在没有 IAM 角色的情况下，Kinesis Data Analytics 无权代表您写入到外部目标。
- 网络中断或其他内部服务故障导致应用程序暂时停止运行。

在您的应用程序重新启动时，Kinesis Data Analytics 确保它从发生故障之前或发生故障时起继续处理和写入输出。这有助于确保它将所有应用程序输出传输到配置的目标，而不会遗漏任何内容。

假设您从同一应用程序内部流中配置多个目标。在应用程序从故障恢复后，Kinesis Data Analytics 会从传输到最慢目标的最后一条记录开始继续将输出永久保存到配置的目标中。这可能会导致将同一输出记录多次传输到其他目标。在这种情况下，您必须在外部处理目标中的潜在重复项。

错误处理

Amazon Kinesis Data Analytics 会直接向您返回 API 或 SQL 错误。有关 API 操作的更多信息，请参阅[操作](#)。有关处理 SQL 错误的更多信息，请参阅 [Amazon Kinesis Data Analytics SQL 参考](#)。

Amazon Kinesis Data Analytics 使用名为 `error_stream` 的应用程序内部错误流报告运行时错误。

使用应用程序内部错误流报告错误

Amazon Kinesis Data Analytics 向名为 `error_stream` 的应用程序内部错误流报告运行时错误。下面是可能出现的错误的示例：

- 从流式传输源读取的一个记录不符合输入架构。
- 您的应用程序代码指定被零除。
- 行顺序错误（例如，流中出现的一个记录具有 `ROWTIME` 值，一个用户修改该值后导致记录顺序错误）。
- 源流中的数据无法转换为架构中指定的数据类型 (强制转换错误)。有关可转换的数据类型的信息，请参阅将 JSON 数据类型映射到 SQL 数据类型。

建议以编程方式在 SQL 代码中处理这些错误或将错误流上的数据永久保存到外部目标。这需要将输出配置添加到您的应用程序 (请参阅配置应用程序输出)。要查看应用程序内错误流工作方式的示例，请参阅示例：探索应用程序内部错误流。

> **Note**
>
> 由于错误流是使用系统账户创建的，因此，Kinesis Data Analytics 应用程序无法以编程方式访问或修改错误流。必须使用错误输出来确定应用程序可能遇到的错误。然后，编写应用程序的 SQL 代码来处理预期的错误情况。

错误流架构

错误流具有以下架构：

Field	数据类型	备注
ERROR_TIME	TIMESTAMP	错误出现的时间
ERROR_LEVEL	VARCHAR(10)	
ERROR_NAME	VARCHAR(32)	
MESSAGE	VARCHAR(4096)	
DATA_ROWTIME	TIMESTAMP	传入记录的行时间

DATA_ROW	VARCHAR(49152)	原始行中的十六进制编码数据。您可以使用标准库对此值进行十六进制解码，也可以使用 Web 资源，如 Hex to String Converter。
PUMP_NAME	VARCHAR(128)	利用 `CREATE PUMP` 定义的原始泵

自动扩展应用程序以提高吞吐量

Amazon Kinesis Data Analytics 可以灵活地扩展您的应用程序，以适应源数据流的数据吞吐量，以及大多数情况下的查询复杂性。Kinesis Data Analytics 以 Kinesis 处理单元 (KPU) 的形式预置容量。一个 KPU 可为您提供内存 (4 GB) 和相应的计算和联网能力。

应用程序的默认 KPU 限制为 64 个。有关申请提高此限制的说明，请参阅 Amazon 服务限制中的申请提高限制。

使用标记

本节介绍如何将密钥值元数据标签添加到 Kinesis Data Analytics 应用程序。这些标签可用于以下目的：

- 确定单个 Kinesis Data Analytics 应用程序的计费。有关更多信息，请参阅 AWS 账单和成本管理指南中的使用成本分配标签。
- 根据标签控制对应用程序资源的访问。有关更多信息，请参阅《用户指南》中的使用标签控制访问。
- 用户定义的目的。您可以根据用户标签定义应用程序的功能。

请注意与标记相关的以下信息：

- 应用程序标签的最大数量包括系统标签。用户定义的应用程序标签的最大数量为 50。
- 如果某项操作包含的标签列表存在重复的 `Key` 值，服务将提示 `InvalidArgumentException`。

本主题包含下列部分：

- 创建应用程序时添加标签

创建应用程序时添加标签

在创建应用程序时，您可以使用 CreateApplication 操作的 tags 参数添加标签。

以下示例请求显示了 CreateApplication 请求的 Tags 节点：

```
"Tags": [
    {
        "Key": "Key1",
        "Value": "Value1"
    },
    {
        "Key": "Key2",
        "Value": "Value2"
    }
]
```

为现有应用程序添加或更新标签

您可以使用 TagResource 操作将标签添加到应用程序中。您无法使用 UpdateApplication 操作将标签添加到应用程序中。

要更新现有标签，可添加一个与现有标签的键相同的标签。

针对 TagResource 操作的以下示例请求可添加新标签或更新现有标签：

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "NewTagKey",
         "Value": "NewTagValue"
      },
      {
         "Key": "ExistingKeyOfTagToUpdate",
         "Value": "NewValueForExistingTag"
```

```
      }
   ]
}
```

列出应用程序的标签

要列出现有的标签，您可以使用 ListTagsForResource 操作。

针对 ListTagsForResource 操作的以下示例请求可列出应用程序的标签：

```
{
   "ResourceARN": "arn:aws:kinesisanalytics:us-west-2:012345678901:application/
MyApplication"
}
```

从应用程序删除标签

要从应用程序中删除标签，您可以使用 UntagResource 操作。

针对 UntagResource 操作的以下示例请求可从应用程序中删除标签：

```
{
   "ResourceARN": "arn:aws:kinesisanalytics:us-west-2:012345678901:application/
MyApplication",
   "TagKeys": [ "KeyOfFirstTagToRemove", "KeyOfSecondTagToRemove" ]
}
```

适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 入门

在下文中，可以找到帮助您开始使用 Amazon Kinesis Data Analytics for SQL 应用程序的主题。如果您是 Kinesis Data Analytics for SQL 应用程序的新用户，我们建议您查看 适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理 中提供的概念和术语，然后再完成“入门”部分中的步骤。

主题

注册 AWS 账户

如果您还没有 AWS 账户，请完成以下步骤来创建一个。

注册 AWS 账户

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

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册 AWS 账户时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

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

创建管理用户

注册 AWS 账户 后，保护您的 AWS 账户根用户，启用 AWS IAM Identity Center，创建一个管理用户，以避免使用根用户执行日常任务。

保护您的 AWS 账户根用户

1. 选择根用户并输入您的 AWS 账户电子邮件地址，以账户所有者身份登录 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. 对您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅《IAM 用户指南》中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建管理用户

1. 启用 IAM Identity Center

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

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

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

作为管理用户登录

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

  要获取使用 IAM Identity Center 用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的登录 AWS 访问门户。

步骤 1：设置 账户并创建管理员用户

首次使用 Amazon Kinesis Data Analytics 之前，请完成以下任务。

1. 注册 AWS

2. 创建 IAM 用户

注册 AWS

当您注册 Amazon Web Services 时，您的 AWS 账户 会自动注册 AWS 中的所有服务，包括 Amazon Kinesis Data Analytics。您只需为使用的服务付费。

借助 Kinesis Data Analytics，您仅需为实际使用的资源付费。如果您是 AWS 新客户，还可以免费试用 Kinesis Data Analytics。有关更多信息，请参阅 AWS 免费使用套餐。

如果您已有 AWS 账户，请跳到下一个任务。如果您还没有 AWS 账户，请按照以下过程中的步骤创建。

创建 AWS 账户

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

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册 AWS 账户时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

请记下您的 AWS 账户 账号，因为在下一个任务中需要使用该账号。

创建 IAM 用户

AWS 中的服务（例如 Amazon Kinesis Data Analytics）要求您在访问时提供凭证，以便让服务确定您是否有权访问服务所拥有的资源。控制台要求您的密码。您可以为您的 AWS 账户 创建访问密钥以访问 AWS CLI 或 API。但是，我们不建议使用您的 AWS 账户 的凭证访问 AWS。相反，我们建议您使用 AWS Identity and Access Management (IAM)。创建 IAM 用户，将该用户添加到具有管理权限的 IAM 组，然后向您创建的 IAM 用户授予管理权限。您随后便可以使用一个特殊的 URL 和该 IAM 用户的凭证访问 AWS。

如果您已注册 AWS，但尚未为自己创建 IAM 用户，则可以使用 IAM 控制台自行创建。

本指南中的入门练习假定您拥有具有管理权限的用户 (`adminuser`)。请按照以下过程在您的账户中创建 `adminuser`。

创建管理员用户和登录控制台

1. 在您的 AWS 账户 中创建名为 `adminuser` 的管理员用户。有关说明，请参考《IAM 用户指南》中的创建您的第一个 IAM 用户和管理员组。
2. 用户可使用特殊 URL 登录 AWS Management Console。有关更多信息，请参阅《IAM 用户指南》中的用户如何登录您的账户。

有关 IAM 的更多信息，请参阅以下文档：

- AWS Identity and Access Management (IAM)
- 入门
- IAM 用户指南

下一个步骤

步骤 2：设置 AWS Command Line Interface (AWS CLI)

注册 AWS 账户

如果您还没有 AWS 账户，请完成以下步骤来创建一个。

注册 AWS 账户

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

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册 AWS 账户时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

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

创建管理用户

注册 AWS 账户 后，保护您的 AWS 账户根用户，启用 AWS IAM Identity Center，创建一个管理用户，以避免使用根用户执行日常任务。

保护您的 AWS 账户根用户

1. 选择根用户并输入您的 AWS 账户电子邮件地址，以账户所有者身份登录 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. 对您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅《IAM 用户指南》中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建管理用户

1. 启用 IAM Identity Center

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

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

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

作为管理用户登录

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

  要获取使用 IAM Identity Center 用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的登录 AWS 访问门户。

步骤 2：设置 AWS Command Line Interface (AWS CLI)

按照以下步骤下载和配置 AWS Command Line Interface (AWS CLI)。

> **Important**
>
> 您不需要使用 AWS CLI 以执行入门练习中的步骤。但是，本指南中的某些练习会用到 AWS CLI。您可以跳过此步骤转至 步骤 3：创建您的初学者 Amazon Kinesis Data Analytics 应用程序，并在稍后需要时设置 AWS CLI。

设置 AWS CLI

1. 下载并配置 AWS CLI。有关说明，请参阅《AWS Command Line Interface 用户指南》中的以下主题：

   - 开始设置 AWS Command Line Interface
   - 配置 AWS Command Line Interface

2. 在 AWS CLI 配置文件中为管理员用户添加一个命名配置文件。在执行 AWS CLI 命令时，您将使用此配置文件。有关命名配置文件的更多信息，请参阅《AWS Command Line Interface 用户指南》中的命名配置文件。

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   有关可用 AWS 区域的列表，请参阅《Amazon Web Services 一般参考》中的区域和端点。

3. 在命令提示符处输入以下帮助命令来验证设置：

   ```
   aws help
   ```

下一个步骤

步骤 3：创建您的初学者 Amazon Kinesis Data Analytics 应用程序

步骤 3：创建您的初学者 Amazon Kinesis Data Analytics 应用程序

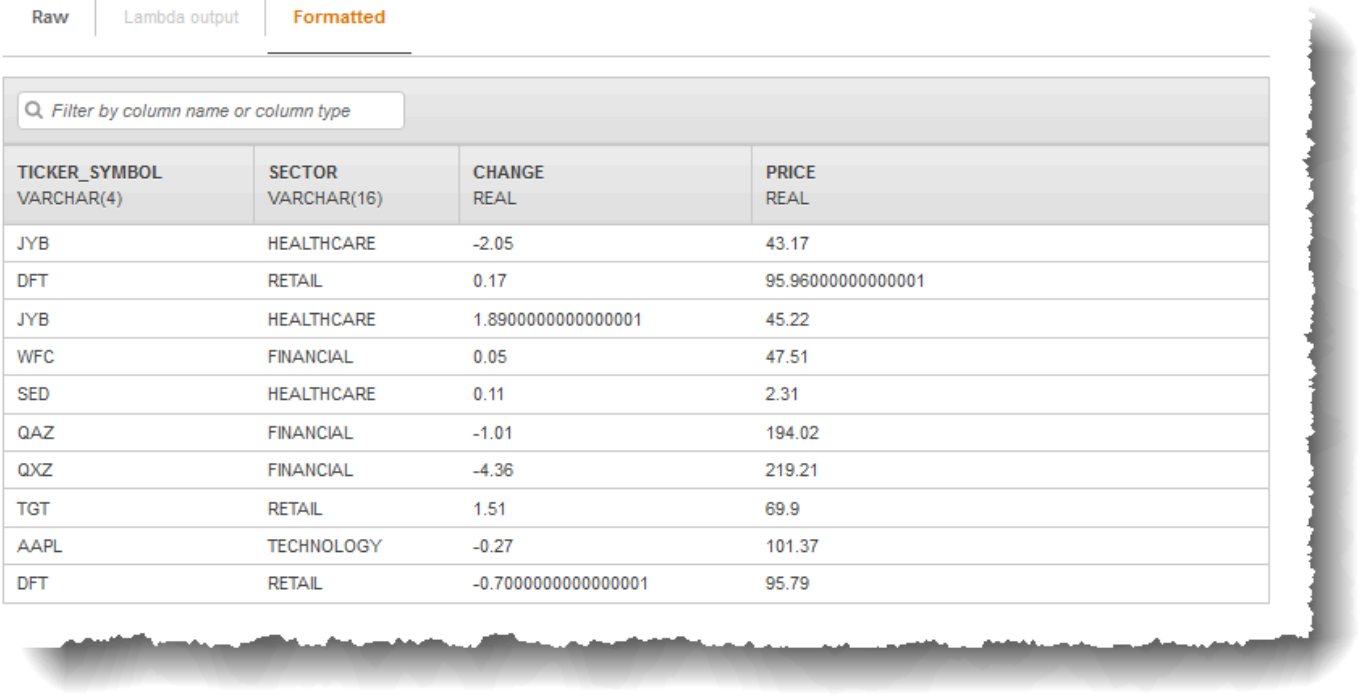
通过执行此部分中的步骤，您可以使用控制台创建第一个 Kinesis Data Analytics 应用程序。

> **Note**
>
> 在尝试入门练习之前，我们建议您先查看 适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理。

在本入门练习中，您可以通过控制台来使用演示流或包含应用程序代码的模板。

- 如果您选择使用演示流，控制台将在您的账户中创建一个名为 `kinesis-analytics-demo-stream` 的 Kinesis 数据流。

  Kinesis Data Analytics 应用程序需要使用流式传输源。对于此源，本指南中的几个 SQL 示例使用演示流 `kinesis-analytics-demo-stream`。控制台还会运行持续向此流添加示例数据 (模拟的股票交易记录) 的脚本，如下所示。



TICKER_SYMBOL VARCHAR(4)	SECTOR VARCHAR(16)	CHANGE REAL	PRICE REAL
JYB	HEALTHCARE	-2.05	43.17
DFT	RETAIL	0.17	95.96000000000001
JYB	HEALTHCARE	1.8900000000000001	45.22
WFC	FINANCIAL	0.05	47.51
SED	HEALTHCARE	0.11	2.31
QAZ	FINANCIAL	-1.01	194.02
QXZ	FINANCIAL	-4.36	219.21
TGT	RETAIL	1.51	69.9
AAPL	TECHNOLOGY	-0.27	101.37
DFT	RETAIL	-0.7000000000000001	95.79

在本练习中，您可以使用 `kinesis-analytics-demo-stream` 作为应用程序的流式传输源。

> **Note**
>
> 演示流会保留在账户中。您可以使用它测试本指南中的其他示例。但是，如果您退出控制台，控制台使用的脚本会停止填充数据。控制台可在需要时提供重新开始填充流的选项。

- 如果您选择使用包含示例应用程序代码的模板，请使用控制台提供的模板代码对演示流执行简单分析。

您可以使用如下这些功能快速设置您的首个应用程序：

1. 创建应用程序 - 您只需提供一个名称。控制台创建应用程序，服务将应用程序状态设置为 READY。

2. 配置输入 - 首先，您将添加流式传输源，即演示流。您必须先在控制台中创建演示流才能使用它。然后，控制台对演示流中的记录进行随机采样，并推断创建的应用程序内部输入流的架构。控制台将应用程序内部流命名为 SOURCE_SQL_STREAM_001。

   控制台使用发现 API 来推断架构。如果需要，可以编辑推断的架构。有关更多信息，请参阅 DiscoverInputSchema。Kinesis Data Analytics 使用此架构创建应用程序内部流。

   在启动应用程序时，Kinesis Data Analytics 代表您持续读取演示流，并在 SOURCE_SQL_STREAM_001 应用程序内部输入流中插入行。

3. 指定应用程序代码 - 您可以使用提供以下代码的模板（称为连续式筛选器）：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
  (symbol VARCHAR(4), sector VARCHAR(12), CHANGE DOUBLE, price DOUBLE);

-- Create pump to insert into output.
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM ticker_symbol, sector, CHANGE, price
      FROM "SOURCE_SQL_STREAM_001"
      WHERE sector SIMILAR TO '%TECH%';
```

   应用程序代码将查询应用程序内部流 SOURCE_SQL_STREAM_001。之后，该代码将使用泵将生成的行插入到另一个应用程序内部流 DESTINATION_SQL_STREAM。有关此编码模式的更多信息，请参阅应用程序代码。

   有关 Kinesis Data Analytics 支持的 SQL 语言元素的信息，请参阅 Amazon Kinesis Data Analytics SQL 参考。

4. 配置输出 - 在此练习中，您不会配置任何输出。也就是说，您不会将应用程序创建的应用程序内部流中的数据永久保存到任何外部目标，而应在控制台中验证查询结果。本指南中的其他示例演示了如何配置输出。要查看其中一个示例，请参阅示例：创建简单警报。

> **Important**
>
> 本练习使用美国东部（弗吉尼亚州北部）区域 (us-east-1) 设置应用程序。您可以使用支持的任意 AWS 区域。

下一个步骤

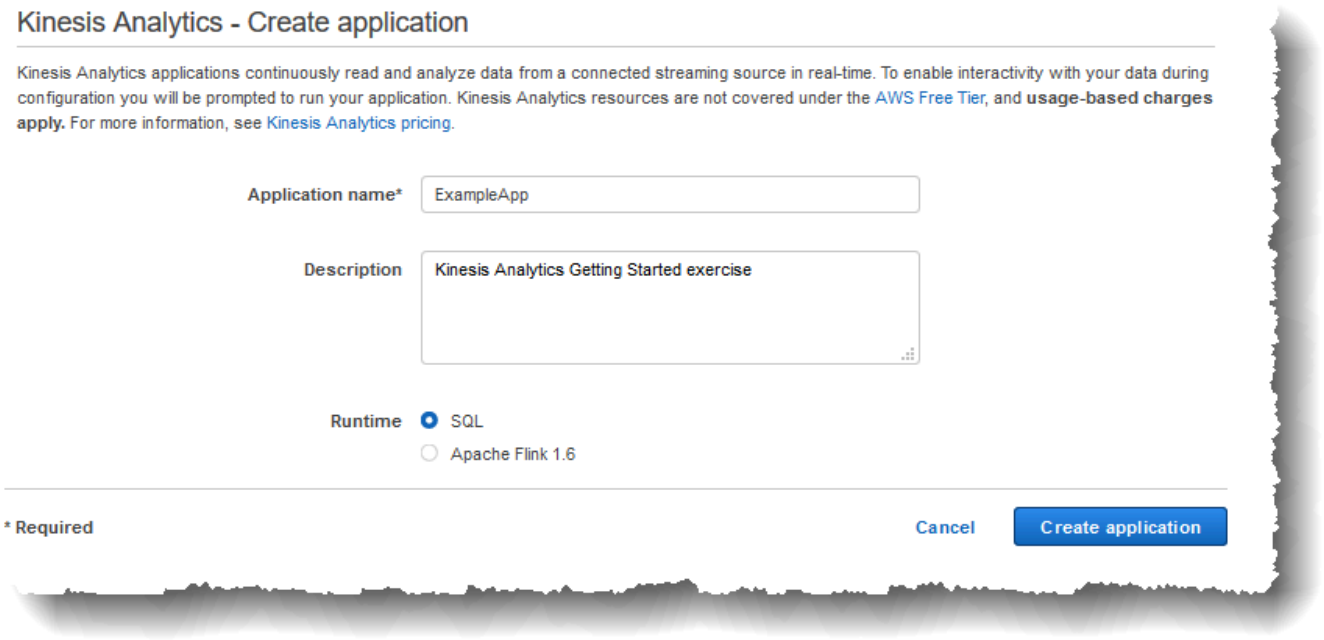
步骤 3.1：创建应用程序

步骤 3.1：创建应用程序

在此部分中，您创建一个 Amazon Kinesis Data Analytics 应用程序。您将在下一步骤中配置应用程序输入。

创建数据分析应用程序

1. 登录 AWS Management Console，然后打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择创建应用程序。
3. 在创建应用程序页面上，键入应用程序名称，键入描述，为应用程序的运行时设置选择 SQL，然后选择创建应用程序。



执行此操作将创建一个具有 READY 状态的 Kinesis Data Analytics 应用程序。控制台显示您可在其中配置输入和输出的应用程序中心。

> **Note**
>
> 要创建应用程序，CreateApplication 操作只需要应用程序名称。您可以在控制台中创建应用程序后添加输入和输出配置。

您可以在下一步中为应用程序配置输入。在输入配置中，您将为应用程序添加一个流式数据源，并通过流式传输源数据采样来发现应用程序内部输入流的架构。

下一个步骤

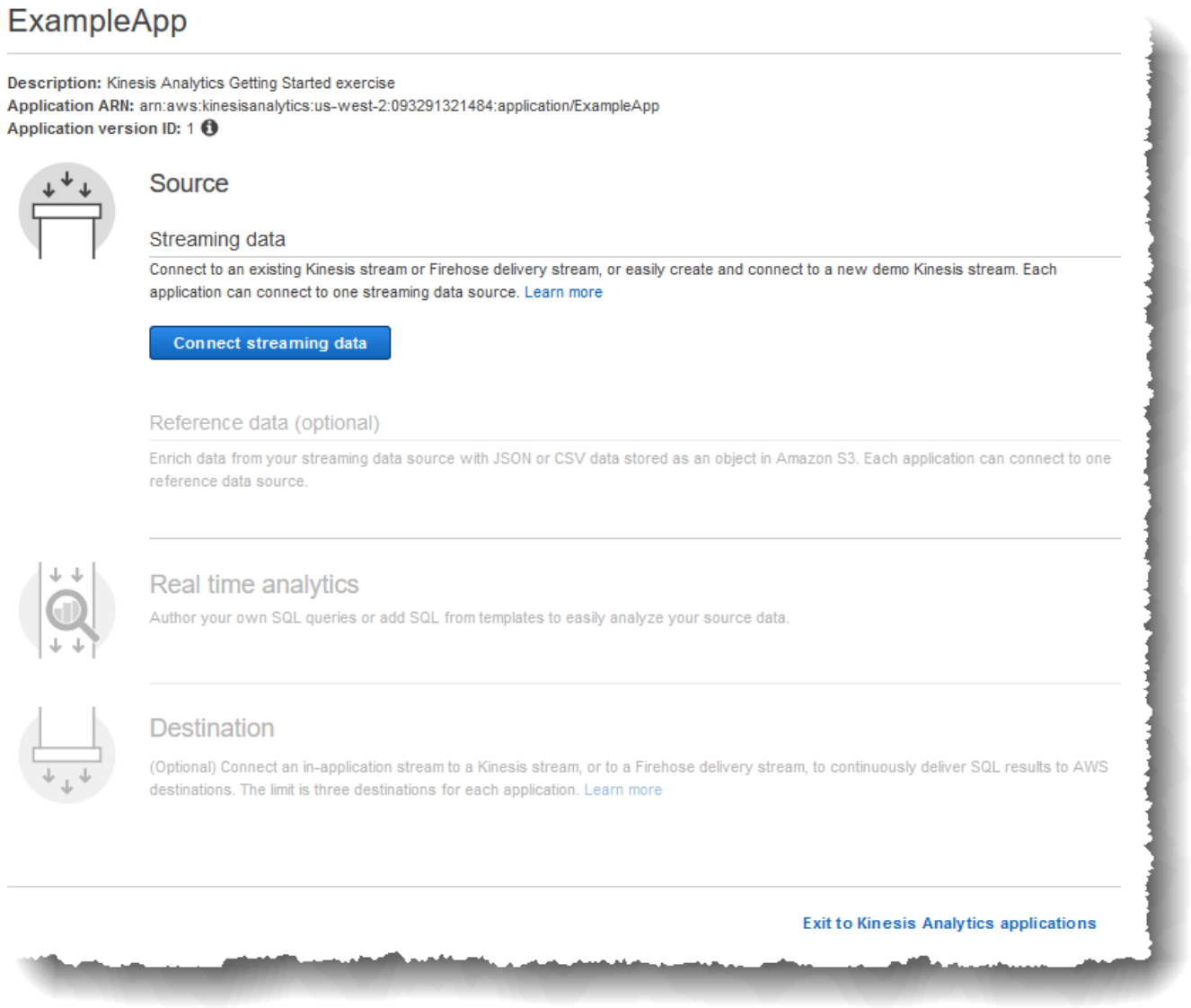
步骤 3.2：配置输入

步骤 3.2：配置输入

您的应用程序需要流式传输源。控制台创建了演示流（名为 `kinesis-analytics-demo-stream`）以帮助您入门。控制台还会运行在流中填充记录的脚本。

为应用程序添加流式传输源

1. 在控制台中的应用程序中心页面上，选择 Connect streaming data (连接流数据)。



2. 在显示的页面上查看以下内容：

   - Source 部分，您可在其中指定应用程序的流式传输源。您可以选择现有流式传输源或创建流式传输源。在本练习中，您将新建一个流，即演示流。

     默认情况下，控制台将创建的应用程序内部输入流命名为 `INPUT_SQL_STREAM_001`。在本练习中，请按原样保留该名称。

- Stream reference name - 此选项显示创建的应用程序内输入流的名称 `SOURCE_SQL_STREAM_001`。您可以更改此名称，但本练习将保留此名称。

  在输入配置中，您可以将演示流映射到创建的应用程序内部输入流。在启动应用程序时，Amazon Kinesis Data Analytics 持续读取演示流，并在应用程序内部输入流中插入行。您可以在应用程序代码中查询此应用程序内部输入流。

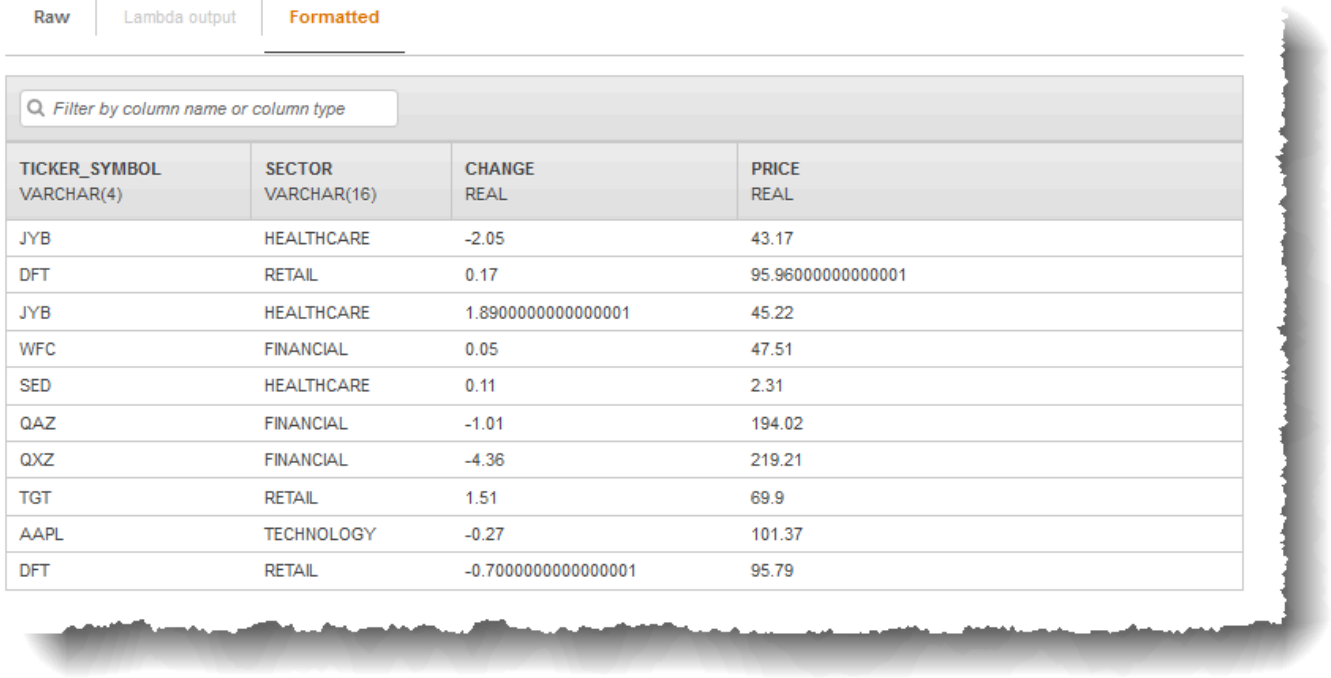
- Record pre-processing with AWS Lambda：(使用 预处理记录)：您可以在此选项中指定一个 AWS Lambda 表达式，以便在应用程序代码执行之前修改输入流中的记录。在此练习中，选中 Disabled 选项。有关 Lambda 预处理的更多信息，请参阅 使用 Lambda 函数预处理数据。

在本页上提供所有信息后，控制台会发送更新请求（请参阅UpdateApplication）以便将输入配置添加到应用程序。

3. 在 Source 页面上选择 Configure a new stream、
4. 选择 Create demo stream。控制台通过执行以下操作来配置应用程序输入：

   - 控制台创建一个名为 `kinesis-analytics-demo-stream` 的 Kinesis 数据流。
   - 控制台将使用示例股票行情机数据填充流。
   - 利用 DiscoverInputSchema 输入操作，控制台将通过读取流上的示例记录来推断架构。推断出的架构是适用于创建的应用程序内部输入流的架构。有关更多信息，请参阅 配置应用程序输入。
   - 控制台将显示推断的架构和从流式传输源读取的用来推断架构的示例数据。

   控制台显示流式传输源中的示例记录。



以下内容将显示在 Stream sample 控制台页面上：

- Raw stream sample 选项卡显示 DiscoverInputSchema API 操作为推断架构而采用的抽样原始流记录。
- Formatted stream sample 选项卡显示 Raw stream sample 选项卡中的数据的表格式版本。
- 如果您选择 Edit schema，则可以编辑推断的架构。在本练习中，请不要更改推断的架构。有关编辑架构的更多信息，请参阅使用架构编辑器。

  如果您选择 Rediscover schema，则可以请求控制台再次运行 DiscoverInputSchema 并推断架构。

5. 选择 保存并继续。

现在，您已具有一个已添加输入配置的应用程序。在下一步中，您将添加 SQL 代码以便对应用程序内部输入流中的数据执行一些分析。

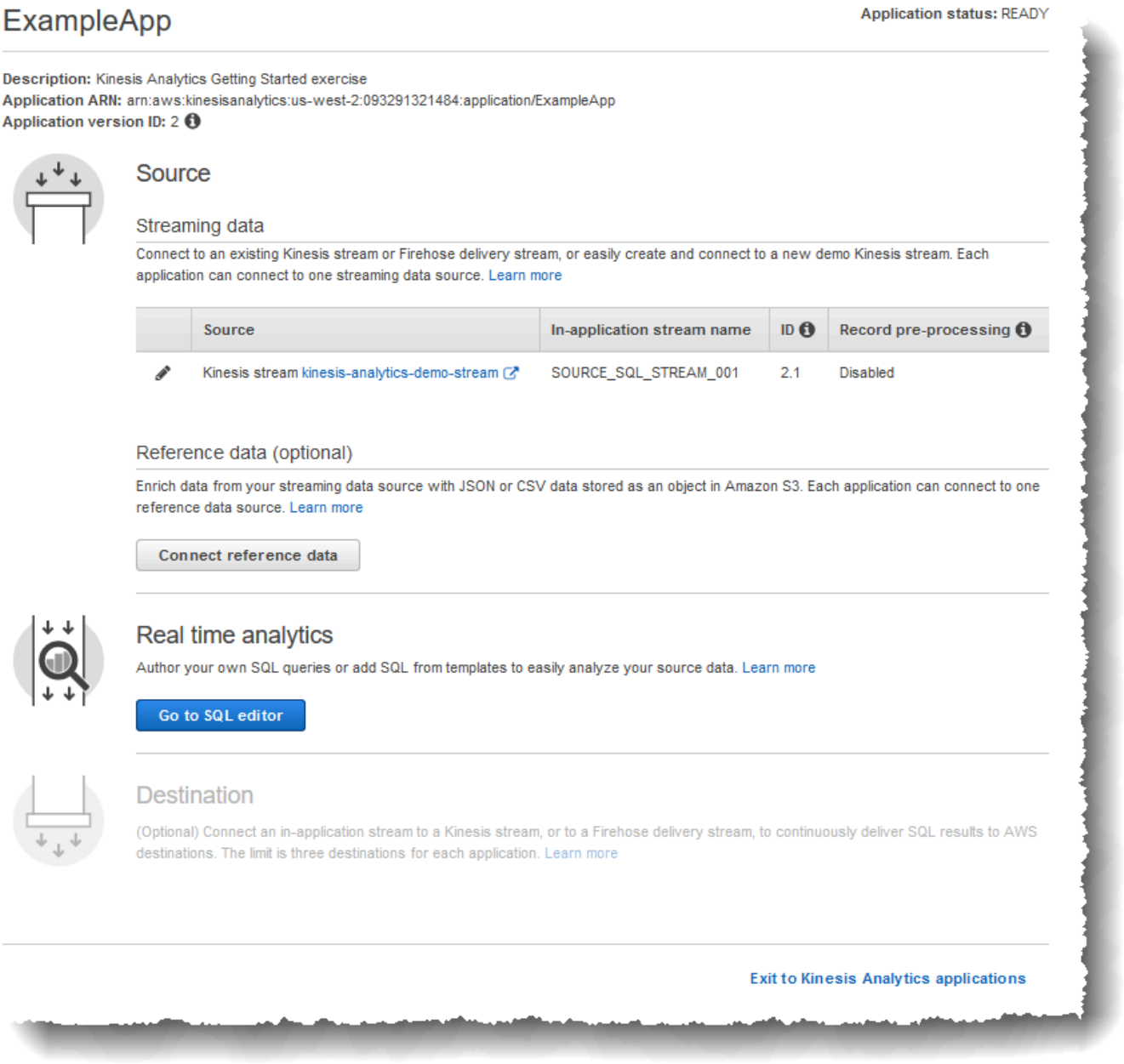
下一个步骤

步骤 3.3：添加实时分析（添加应用程序代码）

步骤 3.3：添加实时分析（添加应用程序代码）

您可以针对应用程序内部流编写您自己的 SQL 查询，但在下一步中，您将使用一个提供示例代码的模板。

1. 在应用程序中心页面上，选择 Go to SQL editor。



2. 在“你想开始跑步吗” ExampleApp “？ 对话框中，选择是，启动应用程序。

   控制台会发送启动应用程序（请参阅StartApplication）的请求，然后会显示 SQL 编辑器页面。

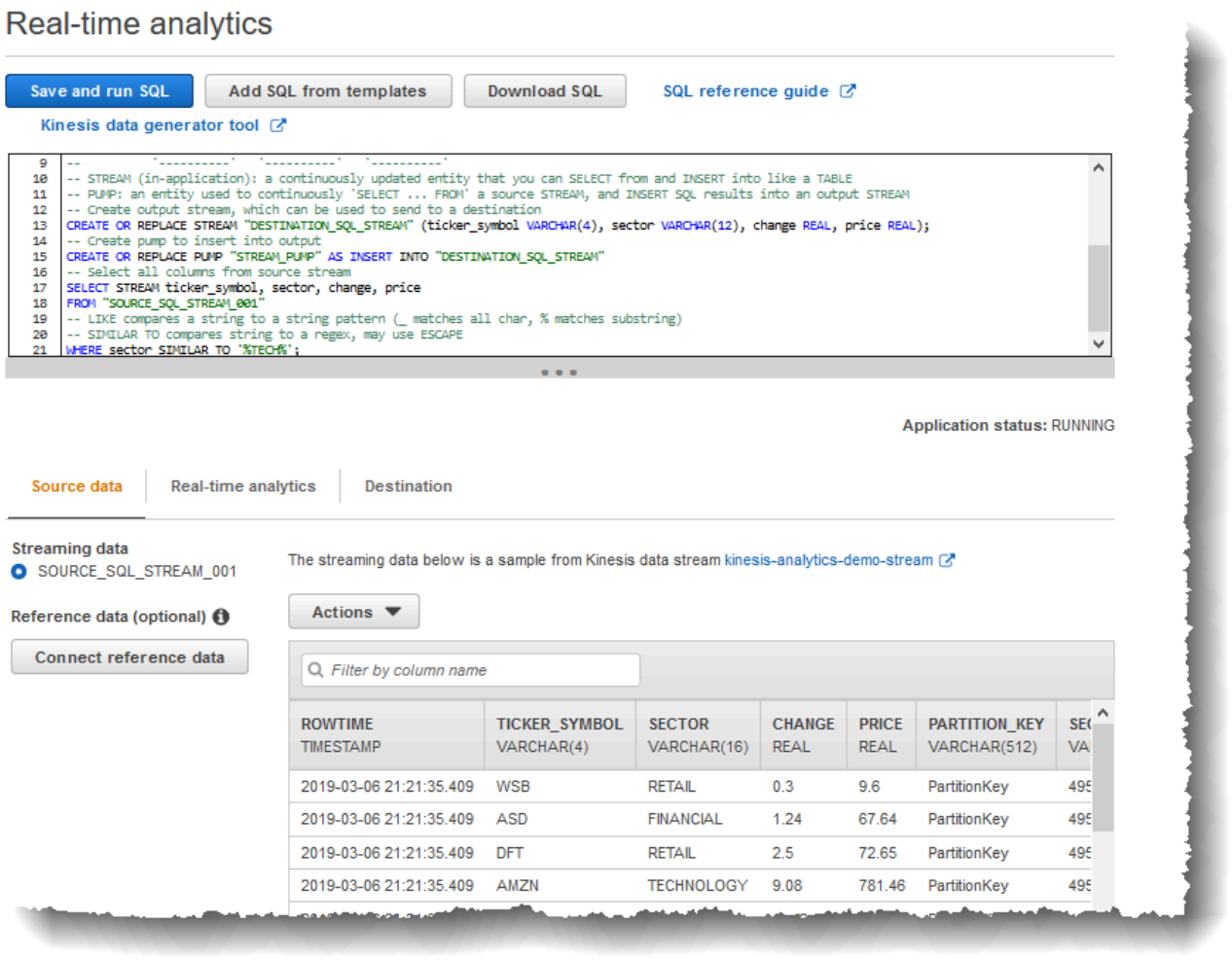
3. 控制台将打开 SQL 编辑器页面。查看该页面，其中包含多个按钮（Add SQL from templates、Save and run SQL）和不同选项卡。
4. 在 SQL 编辑器中，选择 Add SQL from templates。
5. 从可用模板列表中，选择 Continuous filter。示例代码读取来自一个应用程序内部流的数据 (WHERE 子句将筛选行)，并将数据插入到另一个应用程序内部流，如下所示：

   - 它将创建应用程序内部流 DESTINATION_SQL_STREAM。
   - 它将创建泵 STREAM_PUMP，并使用此泵从 SOURCE_SQL_STREAM_001 中选择行，并将这些行插入到 DESTINATION_SQL_STREAM。

6. 选择 Add this SQL to editor。
7. 按如下方式测试应用程序代码：

   请记住，您已启动应用程序（状态为 RUNNING）。因此，Amazon Kinesis Data Analytics 已在持续从流式传输源中读取数据，并将行添加到应用程序内部流 SOURCE_SQL_STREAM_001 中。

   a. 在 SQL 编辑器中，选择 Save and run SQL。控制台首先会发送保存应用程序代码的更新请求。然后，代码会持续执行。
   b. 您可以在 Real-time analytics 选项卡中查看结果。



SQL 编辑器包含以下选项卡：

- Source data 选项卡显示映射到流式传输源的应用程序内部输入流。选择应用程序内部流，您可以看到数据不断传入。请注意应用程序内部输入流中未在输入配置中指定的其他列。其中包括以下时间戳列：

  - ROWTIME - 应用程序内部流中的每一行都具有一个名为 ROWTIME 的特殊列。此列是 Amazon Kinesis Data Analytics 在首个应用程序内流 (映射到流式源的应用程序内输入流) 中插入行的时间戳。

  - Approximate_Arrival_Time - 每个 Kinesis Data Analytics 记录都包含一个名为 Approximate_Arrival_Time 的值。此值是流式源成功接收和存储该记录时设置的大

致到达时间戳。从流式传输源中读取记录时，Kinesis Data Analytics 会将该列提取到应用程序内部输入流中。

这些时间戳值在基于时间的窗口式查询中非常有用。有关更多信息，请参阅 窗口式查询。

- Real-time analytics 选项卡显示应用程序代码创建的所有其他应用程序内部流。它还包括错误流。Kinesis Data Analytics 会将它无法处理的任何行发送到错误流。有关更多信息，请参阅 错误处理。

  选择 DESTINATION_SQL_STREAM 可查看应用程序代码插入的行。请注意您的应用程序代码未创建的其他列。这些列包括 ROWTIME 时间戳列。Kinesis Data Analytics 只需从源代码中复制这些值即可 (SOURCE_SQL_STREAM_001)。

- 目标 选项卡显示 Kinesis Data Analytics 将查询结果写入到的外部目标。您尚未为应用程序输出配置任何外部目标。

下一个步骤

步骤 3.4：(可选) 更新应用程序代码

步骤 3.4：(可选) 更新应用程序代码

在此步骤中，您将研究如何更新应用程序代码。

更新应用程序代码

1. 按如下方式创建另一个应用程序内部流：

   - 创建名为 DESTINATION_SQL_STREAM_2 的另一个应用程序内部流。
   - 创建数据泵，然后从 DESTINATION_SQL_STREAM 中选择行，以便使用数据泵在新创建的流中插入这些行。

   在 SQL 编辑器中，将以下代码附加到现有应用程序代码中：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM_2"
           (ticker_symbol VARCHAR(4),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP_2" AS
   INSERT INTO "DESTINATION_SQL_STREAM_2"
      SELECT STREAM ticker_symbol, change, price
      FROM   "DESTINATION_SQL_STREAM";
```

保存并运行代码。Real-time analytics 选项卡上将显示其他应用程序内部流。

2. 创建两个应用程序内部流。根据股票代码筛选 SOURCE_SQL_STREAM_001 中的行，然后将这些行插入到这些单独的流中。

   将以下 SQL 语句附加到您的应用程序代码：

```
CREATE OR REPLACE STREAM "AMZN_STREAM"
           (ticker_symbol VARCHAR(4),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "AMZN_PUMP" AS
   INSERT INTO "AMZN_STREAM"
      SELECT STREAM ticker_symbol, change, price
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  ticker_symbol SIMILAR TO '%AMZN%';

CREATE OR REPLACE STREAM "TGT_STREAM"
           (ticker_symbol VARCHAR(4),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "TGT_PUMP" AS
   INSERT INTO "TGT_STREAM"
      SELECT STREAM ticker_symbol, change, price
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  ticker_symbol SIMILAR TO '%TGT%';
```

保存并运行代码。请注意 Real-time analytics 选项卡上的其他应用程序内部流。

您现已创建第一个正常运行的 Amazon Kinesis Data Analytics 应用程序。在本练习中，您已完成以下操作：

- 创建了第一个 Kinesis Data Analytics 应用程序。
- 配置将演示流标识为流式源的应用程序输入，并将其映射到创建的应用程序内部流 (SOURCE_SQL_STREAM_001)。Kinesis Data Analytics 持续读取演示流，并在应用程序内部流中插入记录。
- 应用程序代码已查询 SOURCE_SQL_STREAM_001，并将输出写入到名为 DESTINATION_SQL_STREAM 的另一个应用程序内部流。

现在，您可以选择性地配置应用程序输出，以便将应用程序输出写入到外部目标。也就是说，您可配置应用程序输出以便将 DESTINATION_SQL_STREAM 中的记录写入到外部目标。在本练习中，此步骤为可选步骤。要了解如何配置目标，请转到下一步。

下一个步骤

步骤 4：(可选) 使用控制台编辑架构和 SQL 代码.

步骤 4：(可选) 使用控制台编辑架构和 SQL 代码

在下文中，您可以找到有关如何编辑推断的架构和如何编辑 Amazon Kinesis Data Analytics 的 SQL 代码的信息。您可以使用 Kinesis Data Analytics 控制台包含的架构编辑器和 SQL 编辑器以执行此操作。

> **Note**
>
> 要在控制台中访问示例数据，您的登录用户角色必须具有 kinesisanalytics:GetApplicationState 权限。有关 Kinesis Data Analytics 应用程序权限的更多信息，请参阅 访问管理概述。

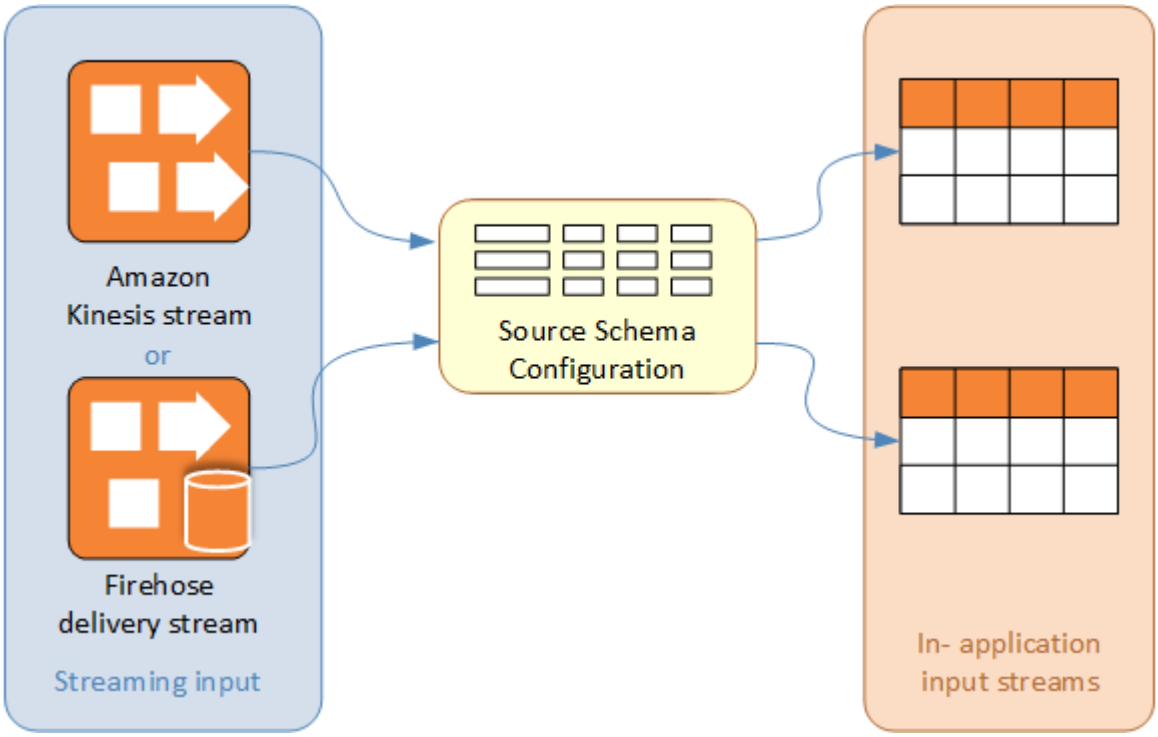
主题

- 使用架构编辑器

- 使用 SQL 编辑器

使用架构编辑器

Amazon Kinesis Data Analytics 应用程序输入流的架构定义了如何为应用程序中的 SQL 查询提供流中的数据。



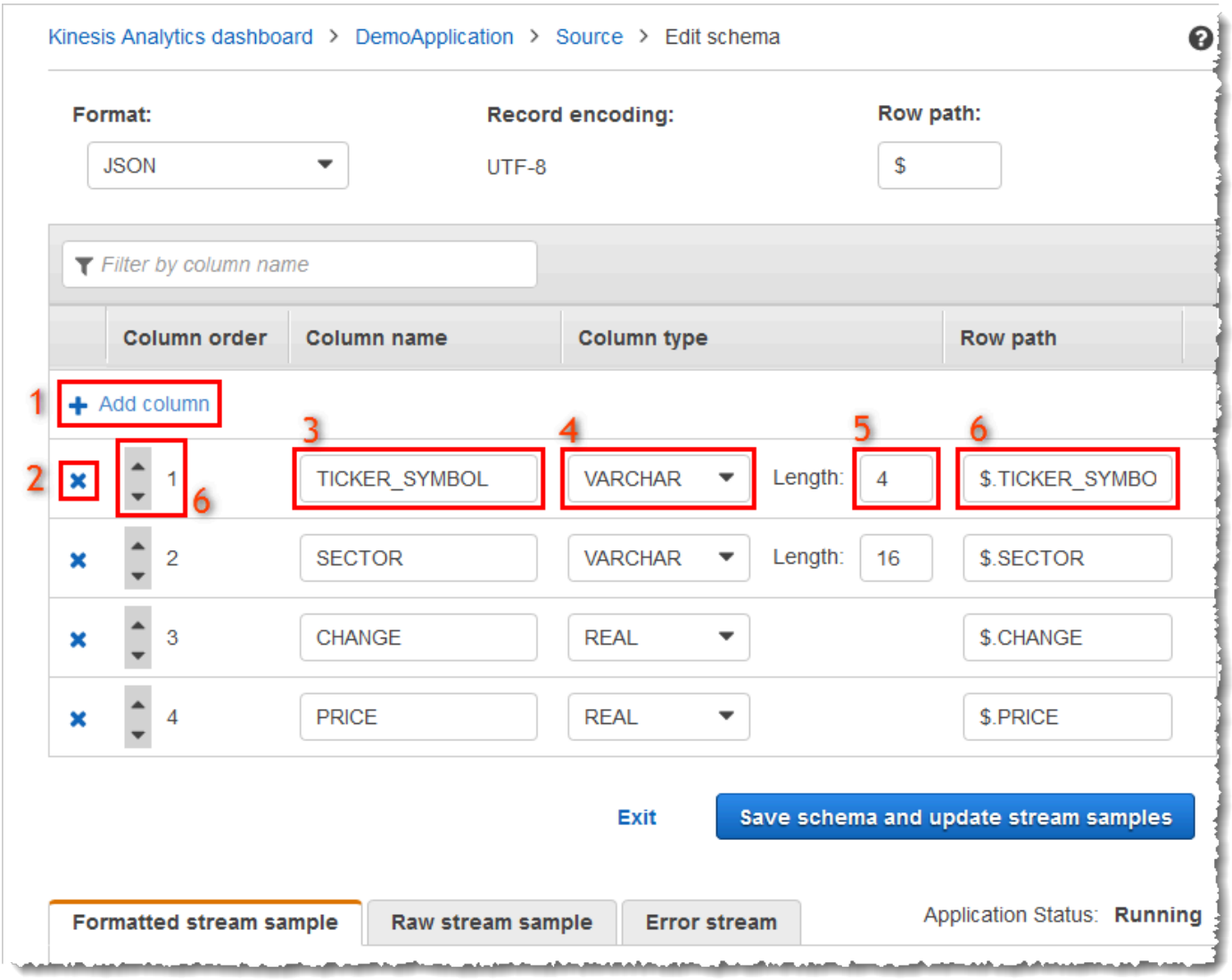
架构包含选择条件，用于确定哪部分流输入将转换为应用程序内部输入流中的数据列。此输入可以是以下项之一：

- JSON 输入流的 JSONPath 表达式。JSONPath 是用于查询 JSON 数据的工具。
- 输入流的列编号 (逗号分隔值 (CSV) 格式)。
- 列名称和用于在应用程序内数据流中呈现数据的 SQL 数据类型。该数据类型还包含字符或二进制数据的长度。

控制台将尝试使用 DiscoverInputSchema 生成架构。如果架构发现失败或返回了不正确或不完整的架构，您必须使用架构编辑器手动编辑架构。

架构编辑器主屏幕

以下屏幕截图显示了架构编辑器的主屏幕。



您可以将下列编辑应用于架构：

- 添加列 (1)：如果未自动检测到数据项，您可能需要添加数据列。
- 删除列 (2)：如果您的应用程序不需要源流中的数据，您可以排除该数据。此排除不会影响源流中的数据。排除数据后，数据将不可供应用程序使用。
- 重命名列 (3)。列名称不能为空，长度必须超过一个字符，并且不得包含保留的 SQL 关键字。该名称还必须符合 SQL 普通标识符的命名标准：名称必须以字母开头，并且只包含字母、下划线字符和数字。
- 更改列的数据类型 (4) 或长度 (5)：您可以为列指定兼容的数据类型。如果您指定了不兼容的数据类型，则将使用 NULL 填充列或完全不填充应用程序内部流。在后一种情况下，错误将写入到错误流。如果您为列指定的长度太小，传入数据将被截断。

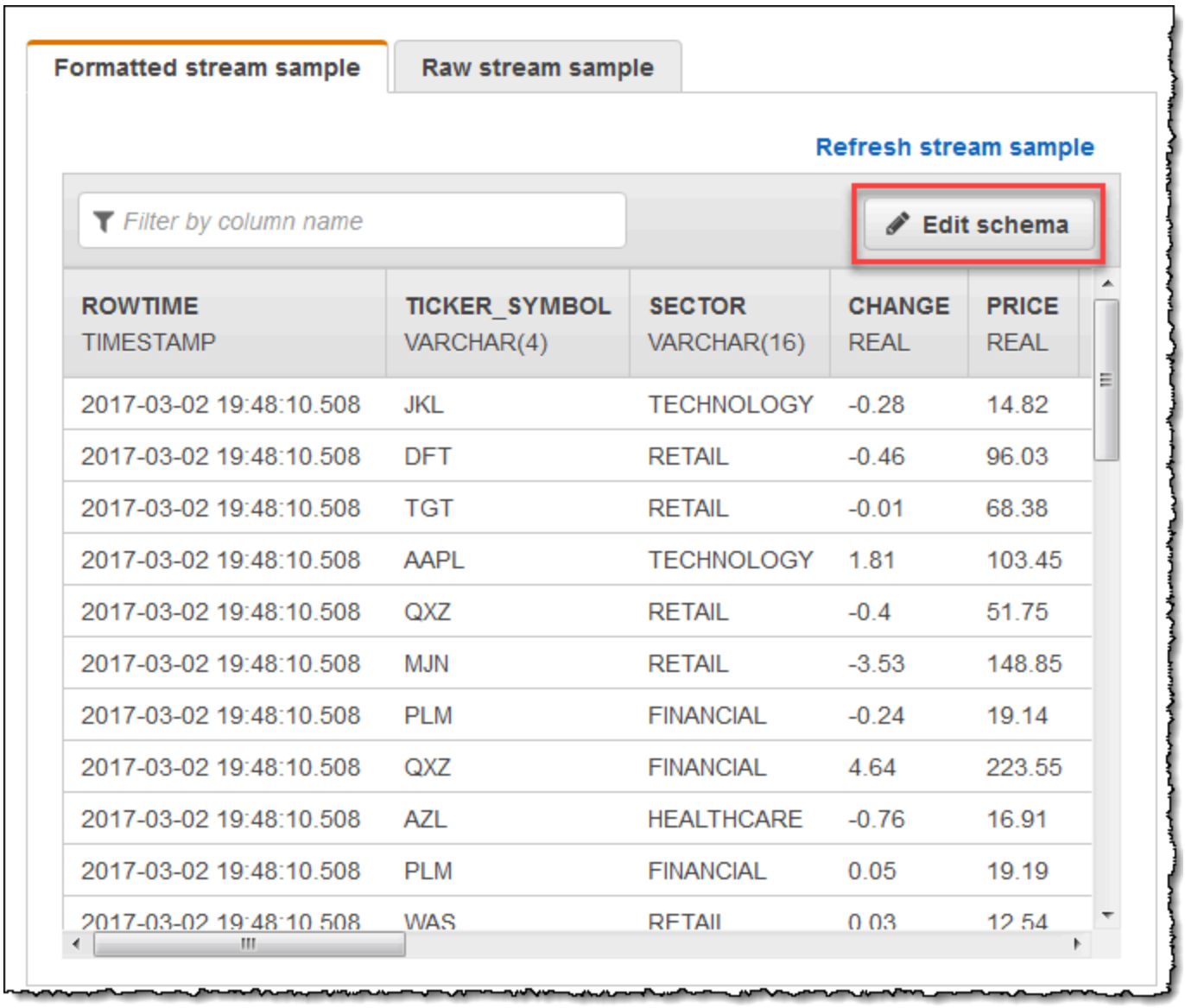
- 更改列的选择条件 (6)：您可以编辑用于确定列中数据源的 JSONPath 表达式或 CSV 列顺序。要更改 JSON 架构的选择条件，请为行路径表达式输入新值。CSV 架构将使用列顺序作为选择条件。要更改 CSV 架构的选择条件，请更改列的顺序。

编辑流式传输源的架构

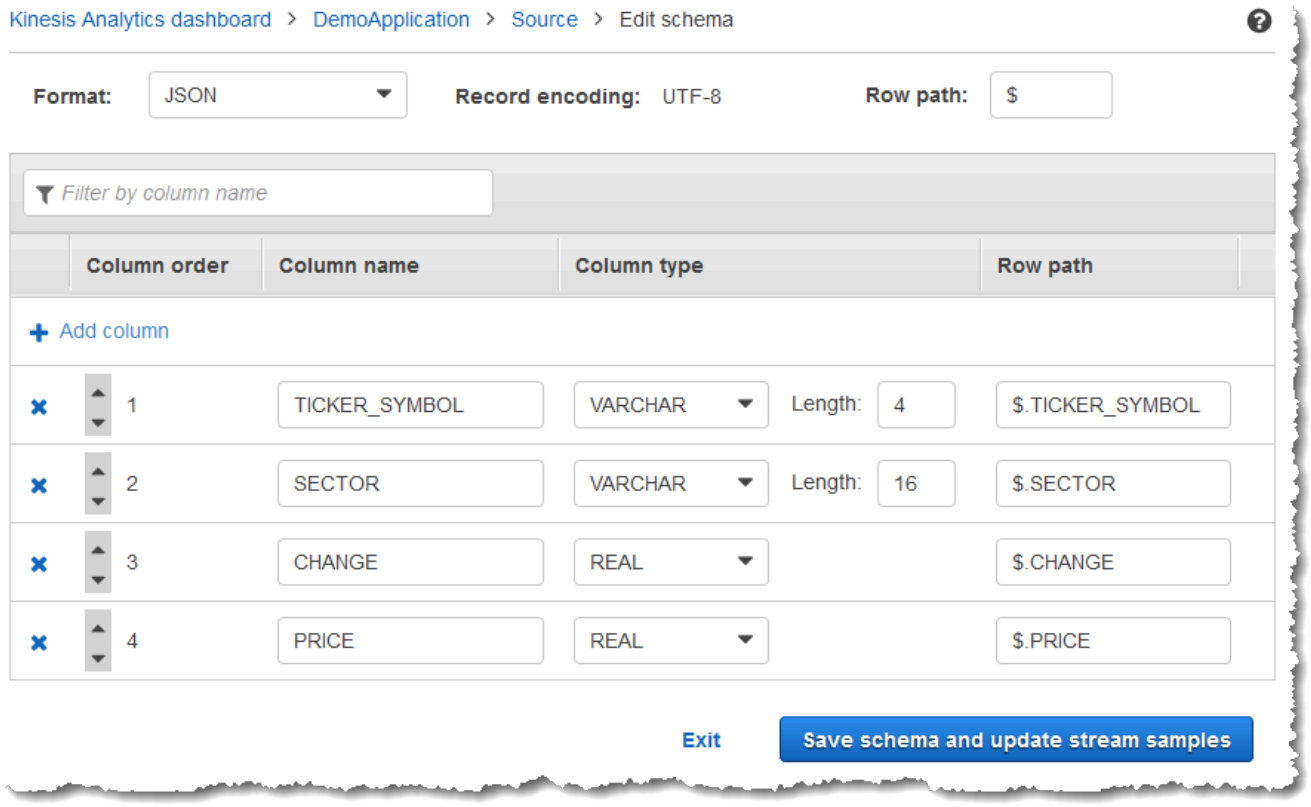
如果您需要编辑流式传输源的架构，请按照以下步骤操作。

编辑流式传输源的架构

1. 在 Source 页上，选择 Edit schema。



2. 在 Edit schema 页上，编辑源架构。



3. 对于 Format，选择 JSON 或 CSV。对于 JSON 或 CSV 格式，支持的编码为 ISO 8859-1。

有关编辑 JSON 或 CSV 格式的架构的更多信息，请参阅后续章节中的过程。

编辑 JSON 架构

您可以通过以下步骤编辑 JSON 架构。

编辑 JSON 架构

1. 在架构编辑器，选择 Add column 以添加列。

   新列将显示在第一个列位置。要更改列顺序，请选择列名称旁边的向上和向下箭头。

   对于新列，请提供以下信息：

   - 对于 Column name，键入一个名称。

     列名称不能为空，长度必须超过一个字符，并且不得包含保留的 SQL 关键字。它还必须符合 SQL 普通标识符的命名标准：必须以字母开头，并且只包含字母、下划线字符和数字。

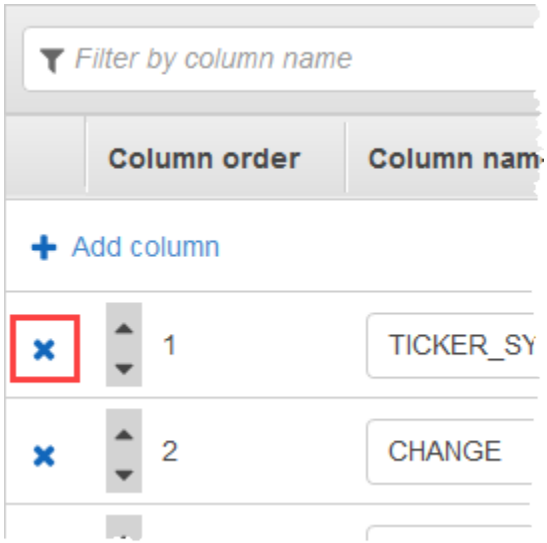
- 对于 Column type，键入一个 SQL 数据类型。

  列类型可以是任何受支持的 SQL 数据类型。如果新数据类型为 CHAR、VARBINARY 或 VARCHAR，请在 Length (长度) 中指定数据长度。有关更多信息，请参阅数据类型。

- 对于 Row path，提供一个行路径。行路径是映射到 JSON 元素的有效 JSONPath 表达式。

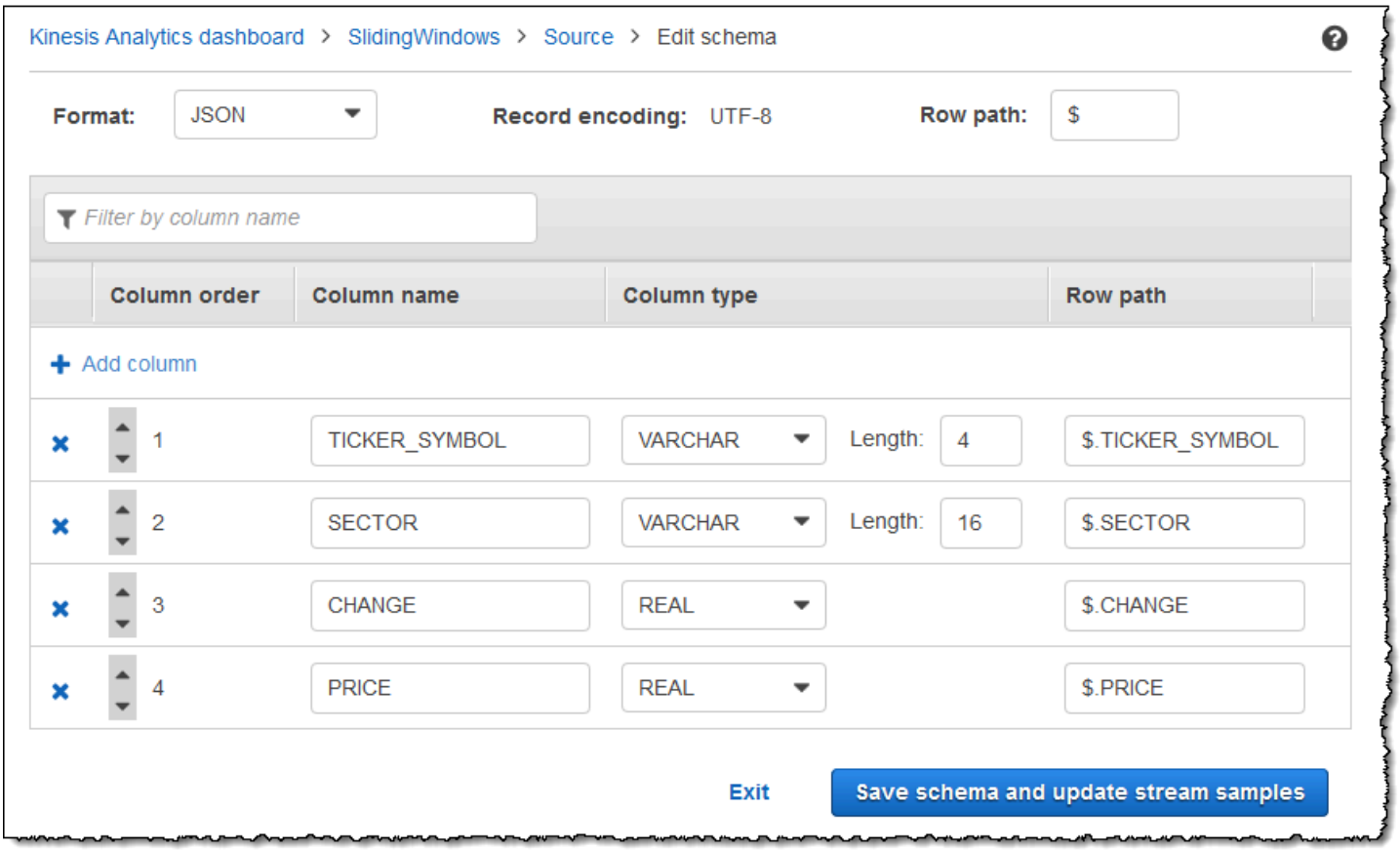
> **Note**
>
> 基础 Row path 值是指向包含要将数据导入到的顶级父项的路径。默认情况下，此值为 $。有关更多信息，请参阅`JSONMappingParameters`中的`RecordRowPath`。

2. 要删除列，请选择列编号旁的 x 图标。



3. 要重命名列，请在 Column name (列名) 中输入新名称。新列名称不能为空，长度必须超过一个字符，并且不得包含保留的 SQL 关键字。它还必须符合 SQL 普通标识符的命名标准：必须以字母开头，并且只包含字母、下划线字符和数字。
4. 要更改列的数据类型，请在 列类型 中选择新数据类型。如果新数据类型为 CHAR、VARBINARY 或 VARCHAR，请在 Length (长度) 中指定数据长度。有关更多信息，请参阅数据类型。
5. 选择 Save schema and update stream 以保存您的更改。

修改后的架构将显示在编辑器中，类似于以下内容。



如果您的架构具有许多行，您可使用 Filter by column name 来筛选行。例如，要编辑以 P 开头的列名称 (如 `Price` 列)，请在 PFilter by column name 框中输入 。

编辑 CSV 架构

您可通过以下步骤编辑 CSV 架构。

编辑 CSV 架构

1. 在架构编辑器中，对于 Row delimiter，选择您的传入数据流使用的分隔符。这是您的流中数据记录之间的分隔符 (如换行符)。
2. 对于 Column delimiter，选择您的传入数据流使用的分隔符。这是您的流中数据字段之间的分隔符 (如逗号)。
3. 要添加列，请选择 Add column。

   新列将显示在第一个列位置。要更改列顺序，请选择列名称旁边的向上和向下箭头。

   对于新列，请提供以下信息：

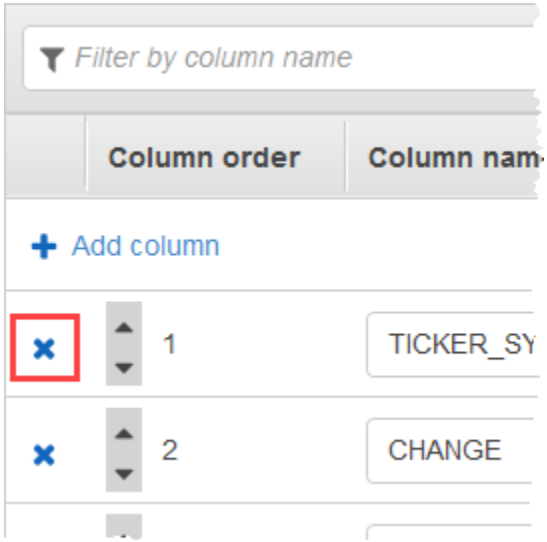
- 对于 Column name (列名)，输入一个名称。

  列名称不能为空，长度必须超过一个字符，并且不得包含保留的 SQL 关键字。它还必须符合 SQL 普通标识符的命名标准：必须以字母开头，并且只包含字母、下划线字符和数字。

- 对于 Column type (列类型)，输入一个 SQL 数据类型。

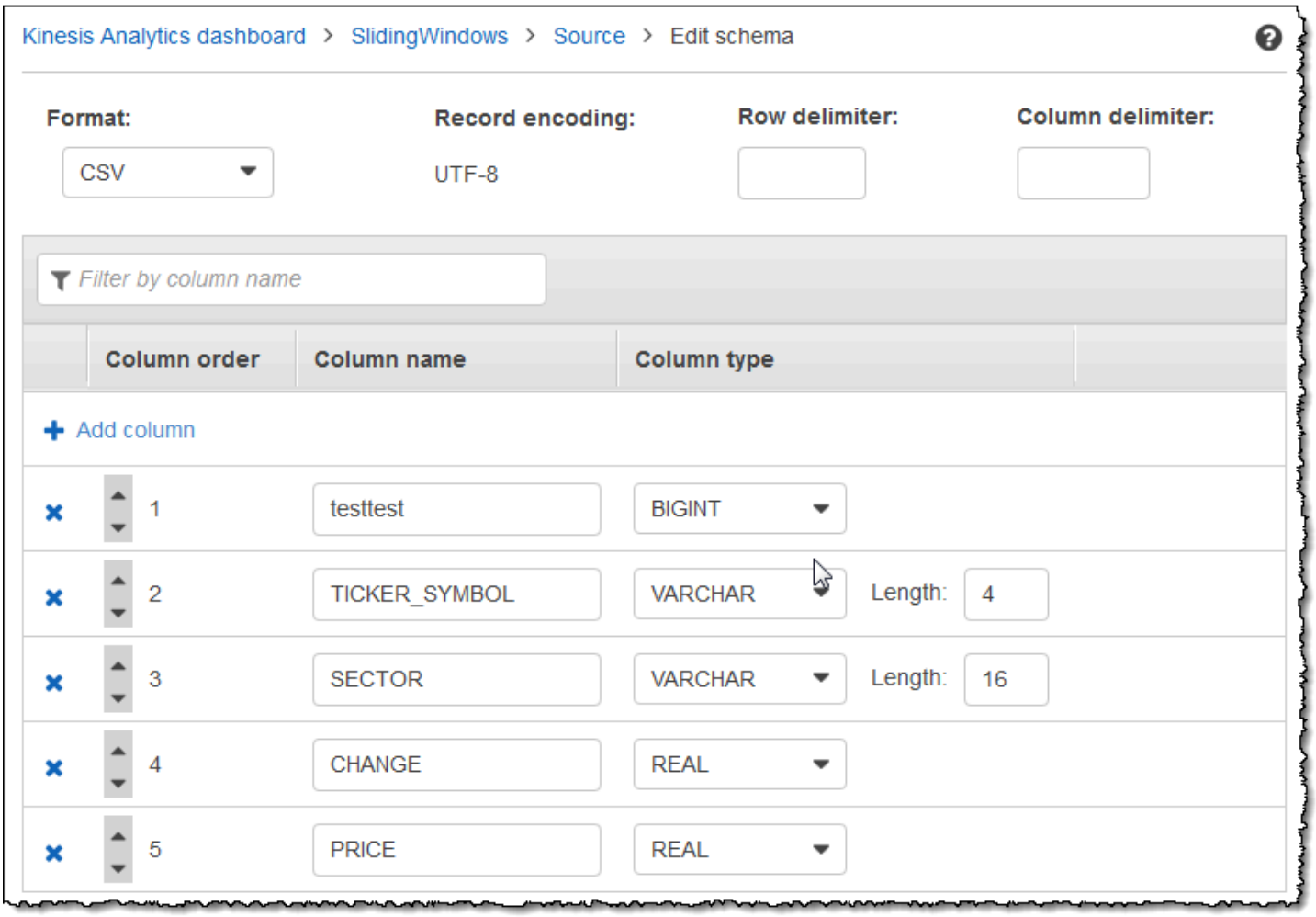
  列类型可以是任何受支持的 SQL 数据类型。如果新数据类型为 CHAR、VARBINARY 或 VARCHAR，请在 Length (长度) 中指定数据长度。有关更多信息，请参阅数据类型。

4. 要删除列，请选择列编号旁的 x 图标。



5. 要重命名列，请在 Column name (列名) 中输入新名称。新列名称不能为空，长度必须超过一个字符，并且不得包含保留的 SQL 关键字。它还必须符合 SQL 普通标识符的命名标准：必须以字母开头，并且只包含字母、下划线字符和数字。
6. 要更改列的数据类型，请在 列类型 中选择新数据类型。如果新数据类型为 CHAR、VARBINARY 或 VARCHAR，请在 Length (长度) 中指定数据长度。有关更多信息，请参阅数据类型。
7. 选择 Save schema and update stream 以保存您的更改。

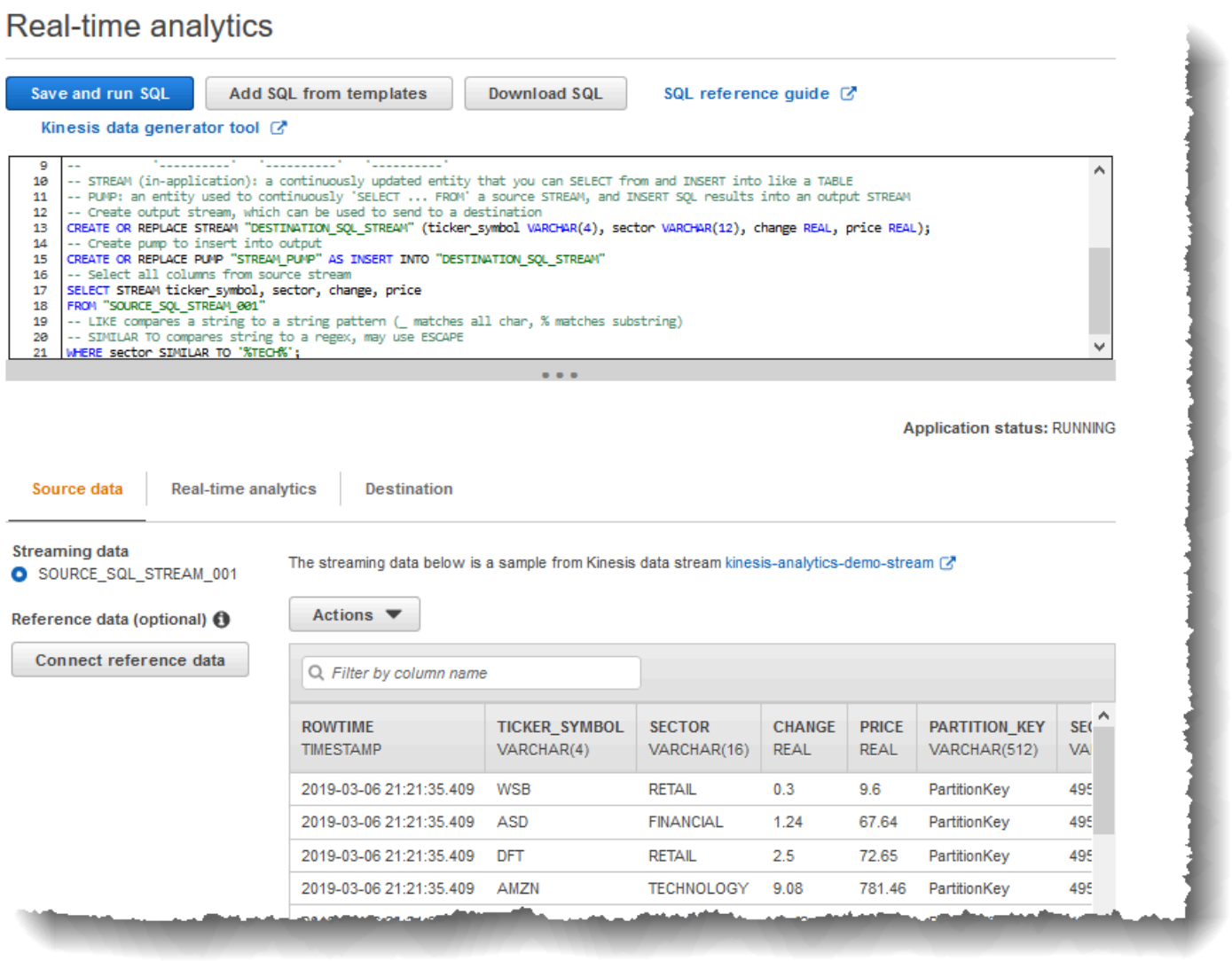
修改后的架构将显示在编辑器中，类似于以下内容。



如果您的架构具有许多行，您可使用 Filter by column name 来筛选行。例如，要编辑以 P 开头的列名称 (如 `Price` 列)，请在 PFilter by column name 框中输入 。

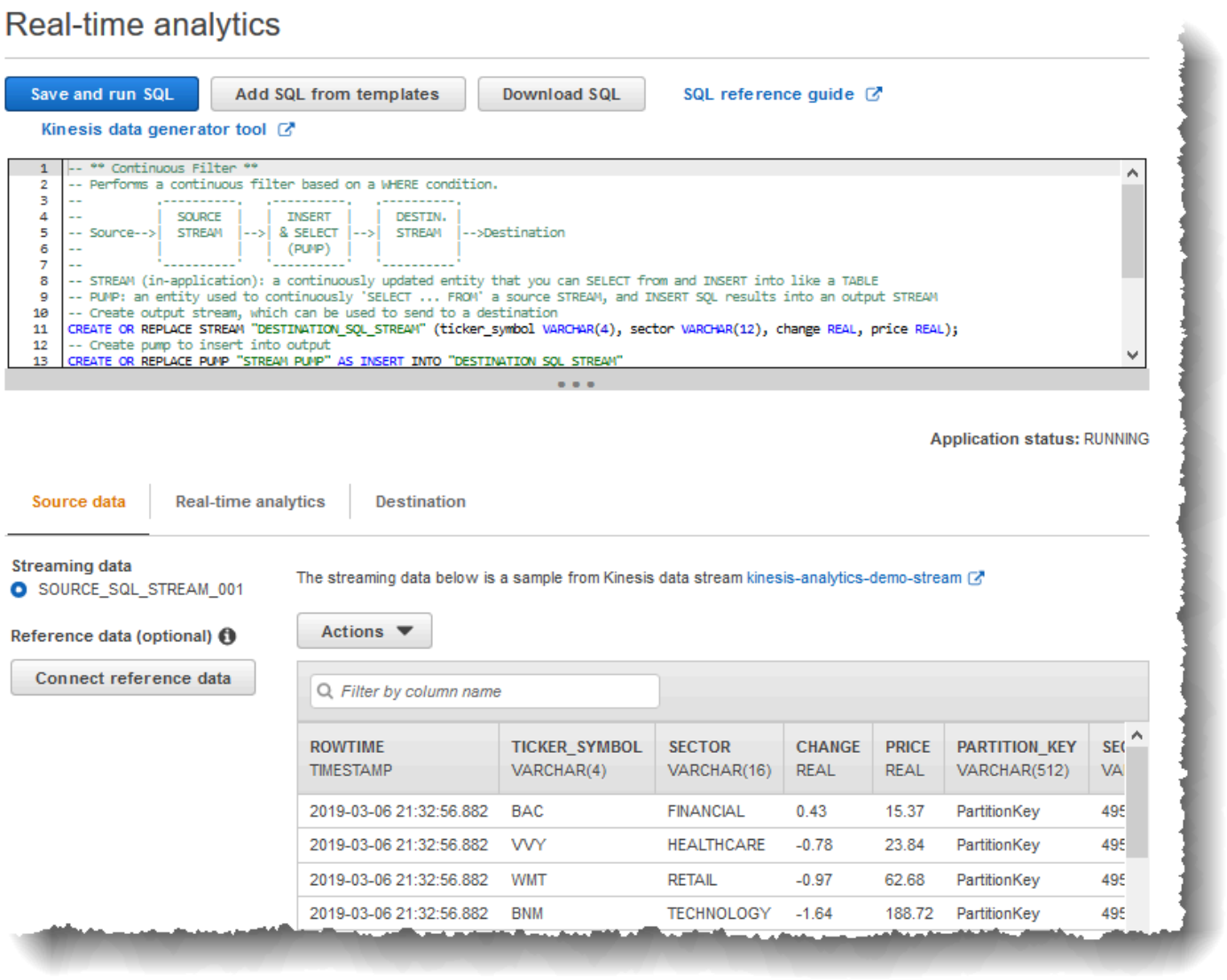
使用 SQL 编辑器

在下文中，您可以找到有关 SQL 编辑器的各个部分及其工作方式的信息。在 SQL 编辑器中，您可以自行编写代码，也可以选择 Add SQL from templates。SQL 模板为您提供了示例 SQL 代码，可以帮助您编写常见的 Amazon Kinesis Data Analytics 应用程序。本指南中的示例应用程序使用了其中的一些模板。有关更多信息，请参阅 [适用于 SQL 的 Kinesis Data Analytics 示例]。



“Source Data”选项卡

Source data 选项卡可标识流式传输源。它还可标识作为此源的映射目标并提供了应用程序输入配置的应用程序内部输入流。



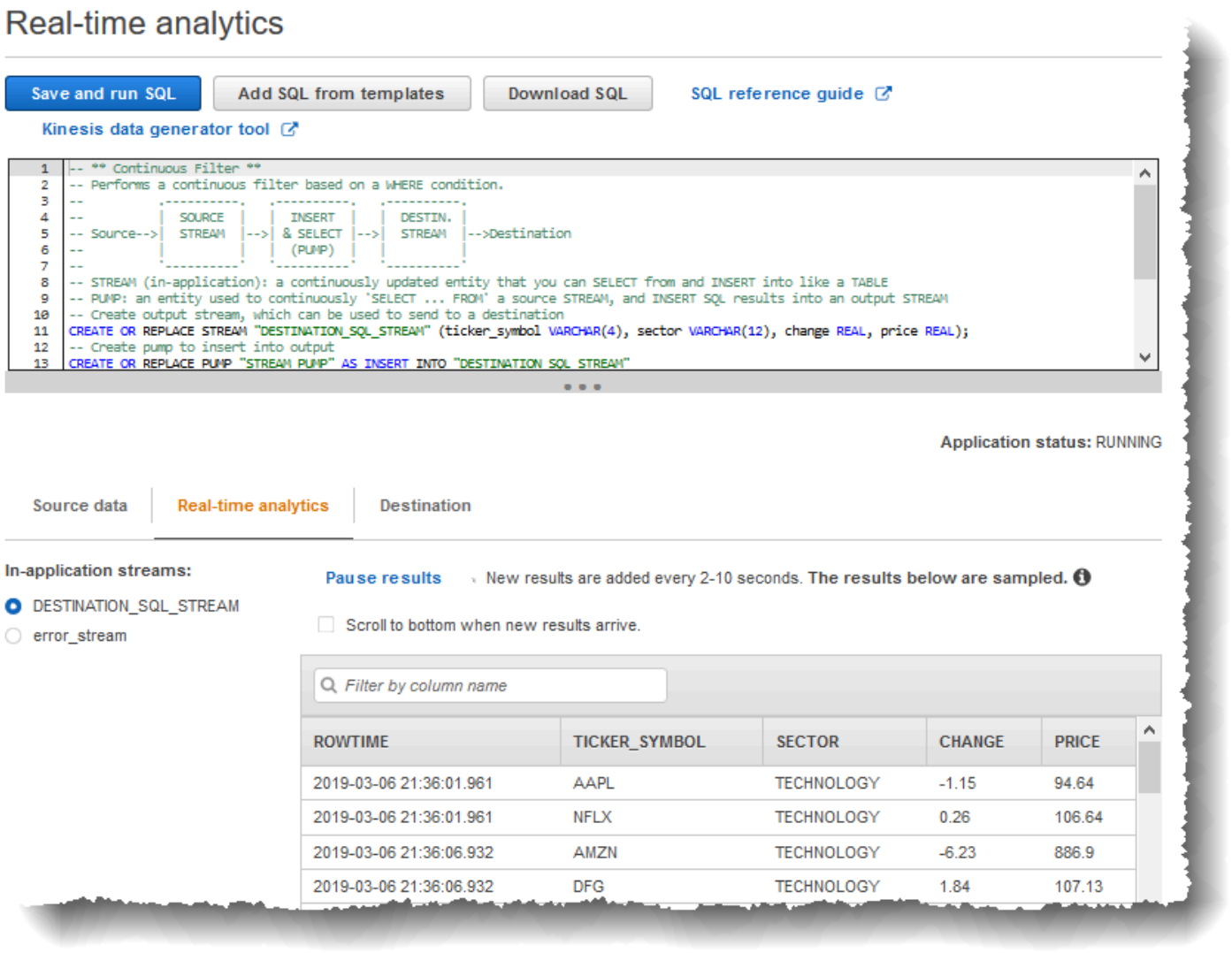
Amazon Kinesis Data Analytics 提供了以下时间戳列，使您无需在输入配置中提供显式映射：

- ROWTIME - 应用程序内部流中的每一行都具有一个名为 `ROWTIME` 的特殊列。此列是 Kinesis Data Analytics 在第一个应用程序内部流中插入行的时间戳。
- Approximate_Arrival_Time - 流式源中的记录包含 `Approximate_Arrival_Timestamp` 列。它是流式源成功接收和存储相关记录时设置的大致到达时间戳。Kinesis Data Analytics 将该列提取到应用程序内部输入流作为 `Approximate_Arrival_Time`。Amazon Kinesis Data Analytics 只在映射到流式源的应用程序内部输入流中提供该列。

这些时间戳值在基于时间的窗口式查询中非常有用。有关更多信息，请参阅 窗口式查询。

“Real-Time Analytics”选项卡

Real-time analytics (实时分析) 选项卡显示了应用程序代码创建的所有应用程序内部流。这组流包含 Amazon Kinesis Data Analytics 为所有应用程序提供的错误流 (`error_stream`)。



“Destination”选项卡

Destination (目标) 选项卡可让您配置应用程序输出，以便将应用程序内部流永久保存到外部目标。您可以配置输出，以便将任何应用程序内部流中的数据保存到外部目标。有关更多信息，请参阅 配置应用程序输出。

流式 SQL 概念

Amazon Kinesis Data Analytics 通过扩展实现了 ANSI 2008 SQL 标准。这些扩展使您可以处理流数据。以下主题介绍了重要的流式 SQL 概念。

主题

应用程序内部流和数据泵

当您配置应用程序输入时，您需要将一个流式传输源映射到已创建的应用程序内部流。数据从流式传输源持续流动到应用程序内部流。应用程序内部流类似于可使用 SQL 语句进行查询的表，但是，它之所以称为流是因为它表示连续的数据流。

> **Note**
>
> 不要将应用程序内流与 Amazon Kinesis 数据流和 Firehose 传输流混为一谈。应用程序内部流仅存在于 Amazon Kinesis Data Analytics 应用程序环境中。Kinesis 数据流和 Firehose 传输流的存在独立于您的应用程序。您可以将它们配置为应用程序输入配置中的流式源，或者配置为输出配置中的一个目标。

您还可以根据需要创建更多的应用程序内部流来存储中间查询结果。创建应用程序内部流是一个两步过程。首先，创建应用程序内部流，然后将数据泵送到其中。例如，假设您应用程序的输入配置创建了名为 INPUTSTREAM 的应用程序内部流。在以下示例中，您将创建另一个流 (TEMPSTREAM)，然后从 INPUTSTREAM 将数据泵送到其中。

1. 按如下所示使用三列创建应用程序内部流 (TEMPSTREAM)：

```
CREATE OR REPLACE STREAM "TEMPSTREAM" (
   "column1" BIGINT NOT NULL,
```

```
    "column2" INTEGER,
    "column3" VARCHAR(64));
```

在引号内指定列名，输入时区分大小写。有关更多信息，请参阅 Amazon Kinesis Data Analytics SQL 参考中的标识符。

2. 使用数据泵将数据插入流。数据泵是连续运行的插入查询，它将数据从一个应用程序内部流插入另一个应用程序内部流。以下语句创建一个数据泵 (SAMPLEPUMP)，然后通过从另一个流 (INPUTSTREAM) 选择记录，将数据插入 TEMPSTREAM。

```
CREATE OR REPLACE PUMP "SAMPLEPUMP" AS
INSERT INTO "TEMPSTREAM" ("column1",
                          "column2",
                          "column3")
SELECT STREAM inputcolumn1,
              inputcolumn2,
              inputcolumn3
FROM "INPUTSTREAM";
```

您可以使多个写入器插入一个应用程序内部流，可以从该流选择多个读取器。可以将应用程序内部流视为实施发布/订阅消息发送范式。在此范式中，数据行（包括创建时和接收时）可以通过一串关联的流式 SQL 语句进行处理、解释和转发，无需存储在传统的 RDBMS 中。

在创建应用程序内部流后，您可以执行普通 SQL 查询。

> **Note**
>
> 查询流时，会使用基于行或基于时间的窗口绑定大多数 SQL 语句。有关更多信息，请参阅窗口式查询。

您还可以联接流。有关联接流的示例，请参阅流数据操作：流联接。

时间戳和 ROWTIME 列

应用程序内部流包含名为 ROWTIME 的特殊列。该列存储 Amazon Kinesis Data Analytics 在第一个应用程序内部流中插入行的时间戳。ROWTIME 反映了 Amazon Kinesis Data Analytics 从流式传输源中读取后将记录插入到第一个应用程序内部流的时间戳。之后，该 ROWTIME 值在您的整个应用程序中进行维护。

> **Note**
>
> 在将一个应用程序内部流中的记录泵取到另一个应用程序内部流时，您不需要明确复制 ROWTIME 列，Amazon Kinesis Data Analytics 将会为您复制该列。

Amazon Kinesis Data Analytics 会确保 ROWTIME 值是单调递增的。您可以在基于时间的窗口式查询中使用此时间戳。有关更多信息，请参阅窗口式查询。

您可以访问 SELECT 语句中的 ROWTIME 列，就像应用程序内部流中的任何其他列一样。例如：

```
SELECT STREAM ROWTIME,
              some_col_1,
              some_col_2
FROM  SOURCE_SQL_STREAM_001
```

了解流式分析中的各种时间

除了 ROWTIME 之外，在实时流式应用程序中还存在其他类型的时间。这些是：

- 事件时间 - 发生事件时的时间戳。它有时也称为客户端时间。经常需要在分析中使用此时间，因为它是事件发生时的时间。但是，许多事件源（例如手机和 Web 客户端）没有可靠的时钟，这可能会导致时间不准确。此外，连接问题可能会导致记录没有按照事件发生顺序出现在流中。

- 接收时间 - 记录添加到流式传输源时的时间戳。Amazon Kinesis Data Streams 在提供此时间戳的每条记录中都提供了一个名为 APPROXIMATE_ARRIVAL_TIME 的字段。有时这也称为服务器端时间。此接收时间通常非常接近事件时间。如果记录接收到流时存在任何种类的延迟，会导致不准确，这种情况通常很少见。此外，接收时间很少出现顺序问题，但由于流数据的分布特点，它也会出现。因此，接收时间通常准确地反映按顺序排列的事件时间。

- 处理时间 – Amazon Kinesis Data Analytics 在第一个应用程序内部流中插入一行的时间戳。Amazon Kinesis Data Analytics 在每个应用程序内部流的 ROWTIME 列中提供此时间戳。处理时间始终是单调递增的。但如果应用程序滞后，则处理时间不准确。（如果应用程序滞后，则处理时间无法准确反映事件时间）。此 ROWTIME 相对于时钟来说很准确，但可能不是事件实际发生的时间。

在基于时间的窗口式查询中使用这些时间有优点也有缺点。我们建议您选择这些时间中的一个或多个，并根据您的使用案例场景选择一种策略来处理相关缺点。

> **Note**
>
> 如果您使用的是基于行的窗口，则时间不是问题，您可以忽略本部分。

我们建议采用双窗口策略，这两个窗口基于不同的时间，即 ROWTIME 和其他时间（接收时间或事件时间）中的一个。

- 使用 ROWTIME 作为第一个窗口，控制查询发送结果的频率，如以下示例所示。它不用作逻辑时间。
- 使用其他时间中您希望与分析关联的逻辑时间。该时间表示事件的发生时间。在以下示例中，分析目标是按股票行情机对记录分组并返回计数。

此策略的优势是它可以使用事件发生时间。它可以轻松处理您的应用程序落后或事件无序到达的情况。当将记录放入应用程序内部流时，如果应用程序落后，记录仍然按第二个窗口中的逻辑时间分组。查询使用 ROWTIME 确保处理顺序。落后的任何记录（与 ROWTIME 值相比，接收时间戳显示的值较早）也会成功处理。

针对入门练习中使用的演示流，考虑以下查询。查询使用 GROUP BY 子句，并在一分钟滚动窗口中发送股票行情机计数。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
    ("ingest_time"    timestamp,
    "APPROXIMATE_ARRIVAL_TIME" timestamp,
    "ticker_symbol"  VARCHAR(12),
    "symbol_count"        integer);


CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND) AS
 "ingest_time",
        STEP("SOURCE_SQL_STREAM_001".APPROXIMATE_ARRIVAL_TIME BY INTERVAL '60' SECOND)
 AS "APPROXIMATE_ARRIVAL_TIME",
        "TICKER_SYMBOL",
        COUNT(*) AS "symbol_count"
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY "TICKER_SYMBOL",
```

```
        STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND),
        STEP("SOURCE_SQL_STREAM_001".APPROXIMATE_ARRIVAL_TIME BY INTERVAL '60' SECOND);
```

在 GROUP BY 中，您首先在一分钟窗口中基于 ROWTIME 对记录分组，然后基于 APPROXIMATE_ARRIVAL_TIME 分组。

结果中的时间戳值已向下舍入为最接近的 60 秒间隔。查询发送的第一组结果显示第一分钟的记录。发送的第二组结果基于 ROWTIME 显示后续分钟内的记录。最后一条记录指示应用程序在应用程序内部流中放入记录是最晚的（与接收时间戳相比，它显示最晚的 ROWTIME 值）。

```
ROWTIME                 INGEST_TIME     TICKER_SYMBOL   SYMBOL_COUNT

--First one minute window.
2016-07-19 17:05:00.0    2016-07-19 17:05:00.0    ABC      10
2016-07-19 17:05:00.0    2016-07-19 17:05:00.0    DEF      15
2016-07-19 17:05:00.0    2016-07-19 17:05:00.0    XYZ      6
--Second one minute window.
2016-07-19 17:06:00.0    2016-07-19 17:06:00.0    ABC      11
2016-07-19 17:06:00.0    2016-07-19 17:06:00.0    DEF      11
2016-07-19 17:06:00.0    2016-07-19 17:05:00.0    XYZ      1  ***

***late-arriving record, instead of appearing in the result of the
first 1-minute windows (based on ingest_time, it is in the result
of the second 1-minute window.
```

您可以将结果推送到下游数据库，以汇总结果来得到每分钟的最终准确计数。例如，您可以将应用程序输出配置为将结果保存到可以写入 Amazon Redshift 表的 Firehose 传输流中。在结果位于 Amazon Redshift 表后，您可以查询该表以计算按 Ticker_Symbol 分组的总数。对于 XYZ，总数是精确的 (6+1)，即使记录延迟到达也是如此。

连续查询

对流数据连续执行流查询。此连续执行使许多方案得以实现，例如应用程序连续查询流和生成警报的能力。

在入门练习中，您创建了一个名为 SOURCE_SQL_STREAM_001 的应用程序内部流。它从演示流（Kinesis 数据流）中持续接收股票价格。架构如下：

```
(TICKER_SYMBOL VARCHAR(4),
 SECTOR varchar(16),
 CHANGE REAL,
```

```
  PRICE REAL)
```

假设您对超过 15% 的股票价格变化感兴趣。您可以在应用程序代码中使用以下查询。此查询连续运行，当检测到大于 15% 的股票价格变化时，此查询会发送记录。

```
SELECT STREAM TICKER_SYMBOL, PRICE
       FROM   "SOURCE_SQL_STREAM_001"
       WHERE  (ABS((CHANGE / (PRICE-CHANGE)) * 100)) > 15
```

使用以下过程设置 Amazon Kinesis Data Analytics 应用程序和测试此查询。

测试查询

1. 按照入门练习创建应用程序。
2. 使用先前的 SELECT 查询替换应用程序代码中的 SELECT 语句。下面显示得到的应用程序代码：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
                                                   price DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM TICKER_SYMBOL,
                    PRICE
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  (ABS((CHANGE / (PRICE-CHANGE)) * 100)) > 15;
```

窗口式查询

对应用程序内部流连续执行应用程序代码中的 SQL 查询。应用程序内部流表示通过您的应用程序持续流动的无边界数据。因此，要从该连续更新输入获取结果集，通常可以使用根据时间或行定义的窗口来限制查询。这些查询也称为窗口式 SQL。

对于基于时间的窗口式查询，需要以时间为单位指定窗口大小（例如，一分钟窗口）。这需要在应用程序内部流中有一个单调递增的时间戳列。(新行的时间戳大于或等于前一行。) Amazon Kinesis Data Analytics 为每个应用程序内部流提供名为 ROWTIME 的时间戳列。在指定基于时间的查询时，可以使用该列。对于您的应用程序，可以选择其他某个时间戳选项。有关更多信息，请参阅时间戳和 ROWTIME 列。

对于基于行的窗口式查询，可以使用行数为单位指定窗口大小。

您可以根据应用程序需求指定查询以滚动窗口方式、滑动窗口方式还是交错窗口方式处理记录。Kinesis Data Analytics 支持以下窗口类型：

- 交错窗口：一个使用基于时间的键控窗口聚合数据的查询，这种窗口在数据到达时打开。这些键允许多个重叠的窗口。这是使用基于时间的窗口聚合数据的推荐方法，因为与 Tumbling 窗口相比，Stagger Windows 可以减少延迟或 out-of-order 数据流量。
- 滚动窗口：一个使用基于时间的不同窗口聚合数据的查询，这些窗口以固定时间间隔打开和关闭。
- 滑动窗口：一个使用固定时间或行计数间隔连续聚合数据的查询。

交错窗口

使用交错窗口 是一种窗口化方法，适用于分析到达时间不一致的数据组。这种方法非常适合任何时间序列分析使用案例，例如一组相关的销售或日志记录。

例如，VPC 流日志具有一个大约 10 分钟的捕获窗口。但是，如果您在客户端上聚合数据，则最多可以具有 15 分钟的捕获窗口。交错窗口非常适合用于聚合这些日志进行分析。

交错窗口解决了多条相关记录不属于同一时间限制窗口的问题，例如在使用了滚动窗口的情况下。

滚动窗口的部分结果

使用滚动窗口聚合延迟或无序数据具有某些限制。

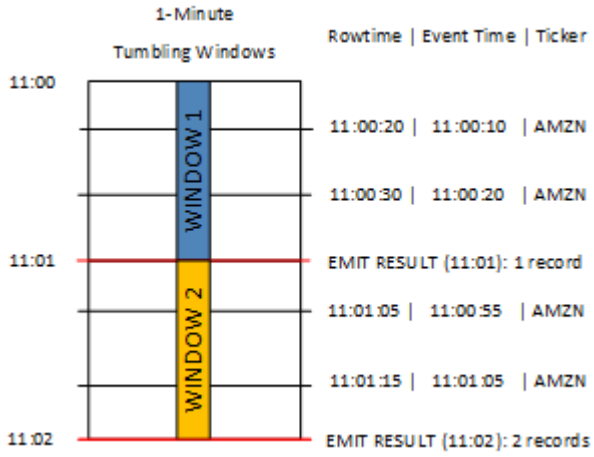
如果使用滚动窗口来分析与时间相关的多组数据，则个别记录可能属于单独的窗口。因此，必须稍后组合每个窗口的部分结果，以便为每组记录生成完整的结果。

在以下滚动窗口查询中，记录按行时间、事件时间和股票代码分组为若干窗口：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    TICKER_SYMBOL VARCHAR(4),
    EVENT_TIME timestamp,
    TICKER_COUNT     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        TICKER_SYMBOL,
        FLOOR(EVENT_TIME TO MINUTE),
        COUNT(TICKER_SYMBOL) AS TICKER_COUNT
    FROM "SOURCE_SQL_STREAM_001"
```

```
    GROUP BY ticker_symbol, FLOOR(EVENT_TIME TO MINUTE),
STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE);
```

在下图中，应用程序根据交易发生的时间（事件时间）以一分钟的粒度计算它收到的交易数量。应用程序可以使用滚动窗口根据行时间和事件时间对数据进行分组。该应用程序接收四条记录，所有记录都在彼此的一分钟之内到达。它按行时间、事件时间和股票代码对记录进行分组。因为一些记录在第一个滚动窗口结束后到达，所以记录并非全部位于同一个一分钟的滚动窗口内。



上图包含以下事件。

ROWTIME	EVENT_TIME	TICKER_SYMBOL
11:00:20	11:00:10	AMZN
11:00:30	11:00:20	AMZN
11:01:05	11:00:55	AMZN
11:01:15	11:01:05	AMZN

滚动窗口应用程序的结果集类似于以下内容。

ROWTIME	EVENT_TIME	TICKER_SYMBOL	COUNT
11:01:00	11:00:00	AMZN	2
11:02:00	11:00:00	AMZN	1

ROWTIME	EVENT_TIME	TICKER_SYMBOL	COUNT
11:02:00	11:01:00	AMZN	1

在前面的结果集中，返回了三个结果：

- ROWTIME 为 11:01:00 的记录，它聚合前两个记录。
- 11:02:00 的记录，它仅聚合第三条记录。此记录在第二个窗口中有一个 ROWTIME，但在第一个窗口中有一个 EVENT_TIME。
- 11:02:00 的记录，它仅聚合第四条记录。

要分析完整的结果集，必须在持久性存储中聚合记录。这给应用程序增加了复杂性和处理要求。

完整结果与交错窗口

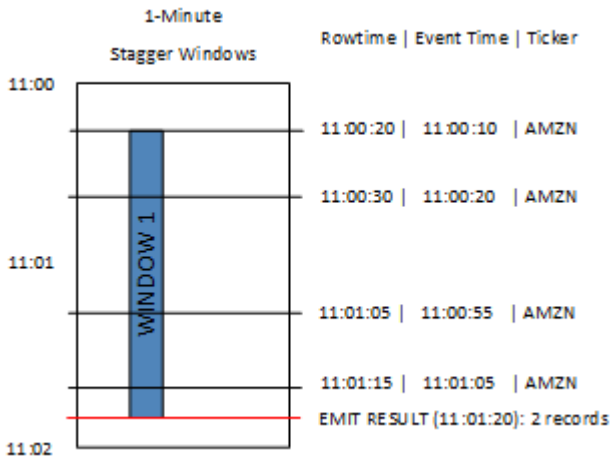
为了提高分析与时间相关的数据记录的准确性，Kinesis Data Analytics 提供了一种名为交错窗口的新窗口类型。在此窗口类型中，窗口在与分区键匹配的第一个事件到达时打开，而不是在固定的时间间隔打开。窗口根据指定的期限关闭，期限是从窗口打开的时间开始计算的。

交错窗口是窗口子句中每个键分组的单独时间限制窗口。应用程序将窗口子句的每个结果聚合在其自己的时间窗口内，而不是对所有结果使用单个窗口。

在以下交错窗口查询中，记录按事件时间和股票代码分组为若干窗口：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol    VARCHAR(4),
    event_time       TIMESTAMP,
    ticker_count     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        TICKER_SYMBOL,
        FLOOR(EVENT_TIME TO MINUTE),
        COUNT(TICKER_SYMBOL) AS ticker_count
    FROM "SOURCE_SQL_STREAM_001"
    WINDOWED BY STAGGER (
            PARTITION BY FLOOR(EVENT_TIME TO MINUTE), TICKER_SYMBOL RANGE INTERVAL '1'
 MINUTE);
```

在下图中，事件按事件时间和股票代码聚合到交错窗口中。



上图包含以下事件，这些事件与分析的滚动窗口应用程序相同：

ROWTIME	EVENT_TIME	TICKER_SYMBOL
11:00:20	11:00:10	AMZN
11:00:30	11:00:20	AMZN
11:01:05	11:00:55	AMZN
11:01:15	11:01:05	AMZN

交错窗口应用程序的结果集类似于以下内容。

ROWTIME	EVENT_TIME	TICKER_SYMBOL	计数
11:01:20	11:00:00	AMZN	3
11:02:15	11:01:00	AMZN	1

返回的记录聚合了前三条输入记录。记录按一分钟的交错窗口分组。当应用程序收到第一条 AMZN 记录（ROWTIME 为 11:00:20）时，交错窗口开始。当 1 分钟交错窗口到期时 (11:01:20)，对于包含位于

交错窗口内的结果（基于 ROWTIME 和 EVENT_TIME）的记录，将被写入输出流。使用交错窗口，在一分钟窗口内具有 ROWTIME 和 EVENT_TIME 的所有记录都将在单个结果中发出。

最后一条记录（其 EVENT_TIME 位于一分钟聚合之外）是单独聚合的。这是因为 EVENT_TIME 是用于将记录分成多个结果集的分区键之一，而第一个窗口的 EVENT_TIME 的分区键是 11:00。

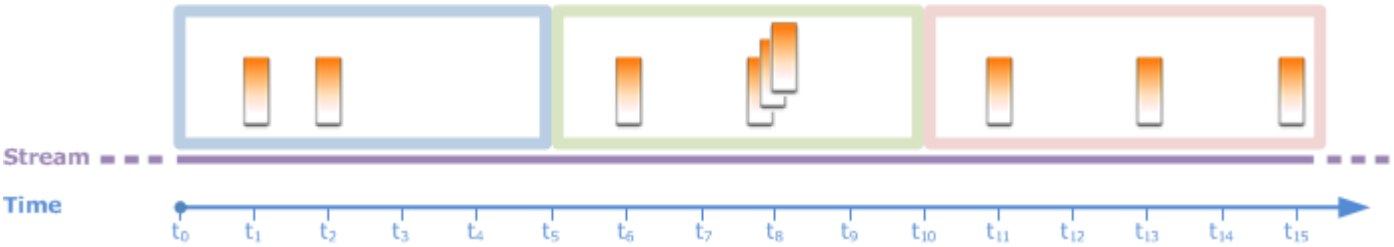
交错窗口的语法在特殊子句 WINDOWED BY 中定义。对于流式处理聚合，使用此子句代替 GROUP BY 子句。该子句紧跟在可选的 WHERE 子句之后和 HAVING 子句之前。

交错窗口在 WINDOWED BY 子句中定义，并带有两个参数：分区键和窗口长度。分区键对传入的数据流进行分区，并定义窗口何时打开。当流中出现具有唯一分区键的第一个事件时，将打开一个交错窗口。在经过由窗口长度定义的固定时间段之后，交错窗口关闭。以下代码示例说明了此语法：

```
...
FROM <stream-name>
WHERE <... optional statements...>
WINDOWED BY STAGGER(
 PARTITION BY <partition key(s)>
 RANGE INTERVAL <window length, interval>
);
```

滚动窗口（使用 GROUP BY 组的聚合）

当一个窗口式查询以非重叠方式处理每个窗口时，这样的窗口称为滚动窗口。在这种情况下，应用程序内部流上的每个记录属于特定窗口。它只处理一次（当查询处理记录所属的窗口时）。



例如，使用 GROUP BY 子句的聚合查询在一个滚动窗口中处理行。入门练习中的演示流接收股票价格数据，而这些数据映射到应用程序中的应用程序内部流 SOURCE_SQL_STREAM_001。这个流具有以下架构。

```
(TICKER_SYMBOL VARCHAR(4),
 SECTOR varchar(16),
```

```
 CHANGE REAL,
 PRICE REAL)
```

在您的应用程序代码中，假设您希望针对一分钟窗口找到每个股票行情机的聚合（最小、最大）价格。您可以使用以下查询。

```
SELECT STREAM ROWTIME,
              Ticker_Symbol,
              MIN(Price) AS Price,
              MAX(Price) AS Price
FROM     "SOURCE_SQL_STREAM_001"
GROUP BY Ticker_Symbol,
         STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

上述示例是一个基于时间的窗口式查询。该查询根据 ROWTIME 值将记录分组。对于每分钟进行的报告，STEP 函数将 ROWTIME 值向下舍入到最接近的分钟。

> **Note**
>
> 您也可以使用 FLOOR 函数将记录分组为不同的窗口。但是，FLOOR 只能将时间值向下舍入到完整的时间单位（小时、分钟、秒等）。建议使用 STEP 以将记录分组为不同的滚动窗口，因为它可以将值向下舍入到任意间隔，例如，30 秒。

该查询是非重叠 (滚动) 窗口示例。GROUP BY 子句在一分钟窗口内对记录进行分组，每个记录属于一个特定窗口 (不重叠)。查询每分钟发送一个输出记录，在其中提供在特定分钟时记录的最小/最大股票行情机价格。在根据输入数据流生成周期性报告时，此类查询很有用。在本示例中，报告是每分钟生成的。

测试查询

1. 按照入门练习设置应用程序。
2. 使用先前的 SELECT 查询替换应用程序代码中的 SELECT 语句。下面显示得到的应用程序代码：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
                                    ticker_symbol VARCHAR(4),
                                    Min_Price     DOUBLE,
                                    Max_Price     DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
```

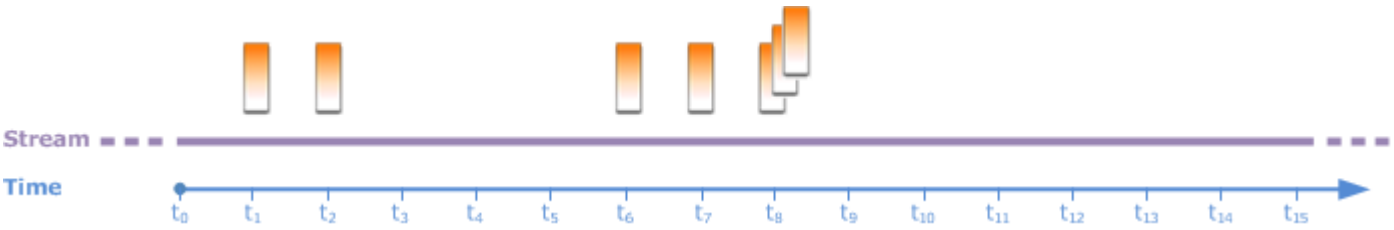
```
INSERT INTO "DESTINATION_SQL_STREAM"
  SELECT STREAM Ticker_Symbol,
                MIN(Price) AS Min_Price,
                MAX(Price) AS Max_Price
  FROM    "SOURCE_SQL_STREAM_001"
  GROUP BY Ticker_Symbol,
           STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

滑动窗口

您可以不使用 GROUP BY 对记录分组，而是定义基于时间或基于行的窗口。您应通过添加显式 WINDOW 子句执行此操作。

在这种情况下，当窗口随着时间滑动时，只要流中出现新记录，Amazon Kinesis Data Analytics 就会发送输出。Kinesis Data Analytics 将会通过在此窗口中处理行来发送此输出。窗口在这种类型的处理中可以重叠，一个记录可以属于多个窗口并且可随各个窗口一起处理。以下示例说明了滑动的窗口。

考虑创建一个简单的查询对流中的记录进行计数。此示例假定有一个 5 秒的窗口。在以下示例流中，新记录分别于时间 t_1、t_2、t_6 和 t_7 到达，有三个记录于时间 t_8 秒时到达。



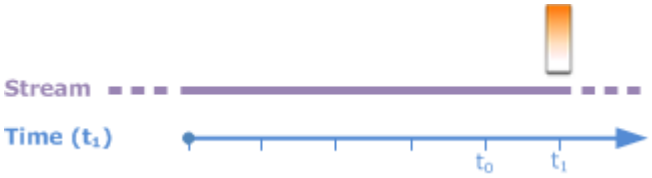
记住以下内容：

- 此示例假定有一个 5 秒的窗口。该 5 秒窗口持续随着时间滑动。
- 对于进入窗口的每一行，滑动窗口会发送输出行。应用程序启动后不久，您会看到查询针对出现在流中的每个新记录发送输出，即使尚未经过 5 秒窗口。例如，当记录出现在第一秒和第二秒时，查询会发送输出。稍后，查询会处理 5 秒窗口中的记录。
- 该窗口随着时间滑动。如果流中的旧记录落后于窗口，查询将不会发送任何输出，除非流中也有一个新记录落在该 5 秒窗口中。

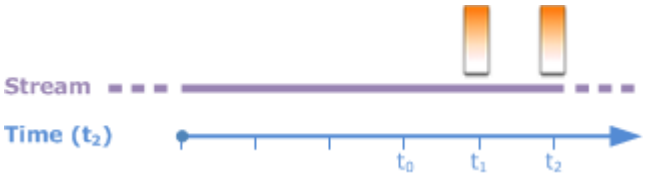
假设查询在 t_0 开始执行。那么，将出现以下情况：

1. 在时间 t_0，查询开始执行。查询不发送输出 (计数值)，因为此时没有记录。

2. 在时间 t_1 处，新记录出现在流中，并且查询发送计数值 1。



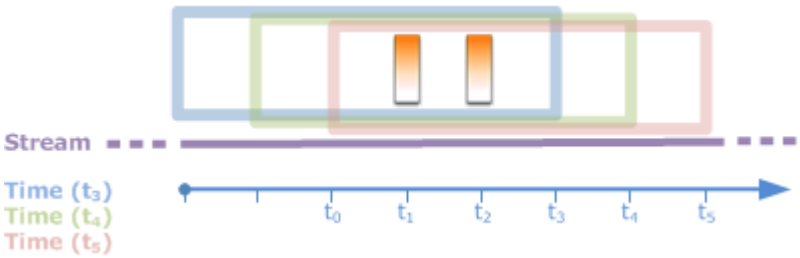
3. 在时间 t_2，出现另一个记录，并且查询发送计数 2。



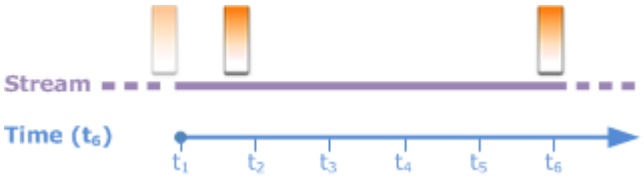
4. 此 5 秒窗口随着时间滑动：

   - 在 t_3 处，滑动窗口为 t_3 到 t_0
   - 在 t4 处 (滑动窗口为 t_4 到 t_0)
   - 在 t_5 处，滑动窗口为 t_5 到 t_0

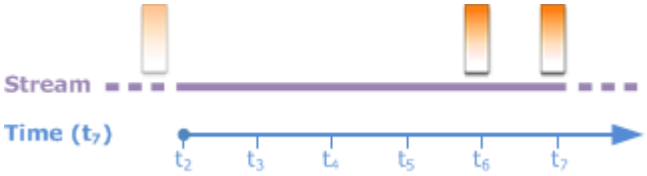
   在所有这些时间，5 秒窗口具有相同的记录——没有新记录。因此，查询不会发送任何输出。



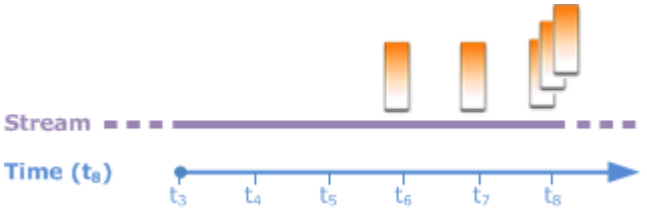
5. 在时间 t_6 处，此 5 秒窗口为 (t_6 到 t_1)。查询在 t_6 处检测到一个新记录，因此它发送了输出 2。t_1 处的记录不再位于窗口中，计数时不考虑。



6. 在时间 t_7 处，此 5 秒窗口为 (t_7 到 t_2)。查询在 t_7 处检测到一个新记录，因此它发送了输出 2。t_2 处的记录不再位于 5 秒窗口中，因此计数时不考虑。



7. 在时间 t_8 处，此 5 秒窗口为 (t_8 到 t_3)。查询检测到三个新记录，因此发送了记录计数 5。



总之，窗口是固定大小，并且随时间滑动。当出现新记录时，查询会发送输出。

> **Note**
>
> 我们建议使用滑动窗口的时间不要超过 1 小时。如果您使用时间更长的窗口，应用程序在常规系统维护之后需要更长的时间才能重新启动。这是因为必须再次从流中读取源数据。

以下示例查询使用 WINDOW 子句定义窗口和执行聚合。由于查询不指定 GROUP BY，因此查询使用滑动窗口方法处理流中的记录。

示例 1：使用一个 1 分钟滑动窗口处理流

在填充应用程序内部流 SOURCE_SQL_STREAM_001 时，请考虑“入门”练习中的演示流。下面是架构。

```
(TICKER_SYMBOL VARCHAR(4),
 SECTOR varchar(16),
 CHANGE REAL,
 PRICE REAL)
```

假设您希望应用程序使用 1 分钟滑动窗口计算聚合。也就是说，对于出现在流中的每个新记录，您希望应用程序通过对前面的 1 分钟窗口中的记录应用聚合来发送输出。

您可以使用以下基于时间的窗口式查询。查询使用 WINDOW 子句定义 1 分钟范围间隔。WINDOW 子句中的 PARTITION BY 按照滑动窗口中的股票行情机值对记录进行分组。

```
SELECT STREAM ticker_symbol,
```

```
              MIN(Price) OVER W1 AS Min_Price,
              MAX(Price) OVER W1 AS Max_Price,
              AVG(Price) OVER W1 AS Avg_Price
FROM   "SOURCE_SQL_STREAM_001"
WINDOW W1 AS (
   PARTITION BY ticker_symbol
   RANGE INTERVAL '1' MINUTE PRECEDING);
```

测试查询

1. 按照入门练习设置应用程序。
2. 使用先前的 SELECT 查询替换应用程序代码中的 SELECT 语句。生成的应用程序代码如下。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
                         ticker_symbol VARCHAR(10),
                         Min_Price     double,
                         Max_Price     double,
                         Avg_Price     double);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
     SELECT STREAM ticker_symbol,
                   MIN(Price) OVER W1 AS Min_Price,
                   MAX(Price) OVER W1 AS Max_Price,
                   AVG(Price) OVER W1 AS Avg_Price
     FROM   "SOURCE_SQL_STREAM_001"
     WINDOW W1 AS (
        PARTITION BY ticker_symbol
        RANGE INTERVAL '1' MINUTE PRECEDING);
```

示例 2：对滑动窗口应用聚合的查询

针对演示流的以下查询将返回一个 10 秒窗口中，每个股票行情机的价格的平均百分比变化。

```
SELECT STREAM Ticker_Symbol,
              AVG(Change / (Price - Change)) over W1 as Avg_Percent_Change
FROM "SOURCE_SQL_STREAM_001"
WINDOW W1 AS (
        PARTITION BY ticker_symbol
        RANGE INTERVAL '10' SECOND PRECEDING);
```

测试查询

1. 按照入门练习设置应用程序。
2. 使用先前的 SELECT 查询替换应用程序代码中的 SELECT 语句。生成的应用程序代码如下。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
                            ticker_symbol VARCHAR(10),
                            Avg_Percent_Change double);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM Ticker_Symbol,
                    AVG(Change / (Price - Change)) over W1 as Avg_Percent_Change
      FROM "SOURCE_SQL_STREAM_001"
      WINDOW W1 AS (
              PARTITION BY ticker_symbol
              RANGE INTERVAL '10' SECOND PRECEDING);
```

示例 3：从同一流的多个滑动窗口查询数据

您可以编写查询以发送输出，其中的每个列值都是使用同一流上定义的不同滑动窗口计算的。

在以下示例中，查询将发送输出股票行情机、价格、a2 和 a10。查询将发送股票代码的输出，这些代码的两行移动平均值超过了 10 行移动平均值。a2 和 a10 列值派生自 2 行和 10 行滑动窗口。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
                          ticker_symbol      VARCHAR(12),
                          price              double,
                          average_last2rows  double,
                          average_last10rows double);

CREATE OR REPLACE PUMP "myPump" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM ticker_symbol,
              price,
              avg(price) over last2rows,
              avg(price) over last10rows
FROM SOURCE_SQL_STREAM_001
WINDOW
    last2rows AS (PARTITION BY ticker_symbol ROWS 2 PRECEDING),
    last10rows AS (PARTITION BY ticker_symbol ROWS 10 PRECEDING);
```

要针对演示流测试此查询，请按照示例 1中介绍的测试过程操作。

流数据操作：流联接

您可以在应用程序中拥有多个应用程序内部流。您可以编写 JOIN 查询关联到达这些流的数据。例如，假设您拥有以下应用程序内部流：

- OrderStream— 接收正在下达的股票订单。

```
(orderId SqlType, ticker SqlType, amount SqlType, ROWTIME TimeStamp)
```

- TradeStream— 接收这些订单的股票交易结果。

```
(tradeId SqlType, orderId SqlType, ticker SqlType, amount SqlType, ticker SqlType,
 amount SqlType, ROWTIME TimeStamp)
```

以下 JOIN 查询示例与这些流上的数据相关联。

示例 1：报告所提交订单在 1 分钟内有成交记录的订单

在此示例中，查询联接了 OrderStream 和 TradeStream。但是，由于我们只需要在下订单后 1 分钟内产生的交易，因此查询针对 TradeStream 定义 1 分钟的窗口。有关窗口式查询的信息，请参阅滑动窗口。

```
SELECT STREAM
     ROWTIME,
     o.orderId, o.ticker, o.amount AS orderAmount,
     t.amount AS tradeAmount
FROM OrderStream AS o
JOIN TradeStream OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS t
ON   o.orderId = t.orderId;
```

您可以按如下所示使用 WINDOW 子句并编写前述查询来明确定义窗口：

```
SELECT STREAM
    ROWTIME,
    o.orderId, o.ticker, o.amount AS orderAmount,
    t.amount AS tradeAmount
FROM OrderStream AS o
```

```
JOIN TradeStream OVER t
ON o.orderId = t.orderId
WINDOW t AS
    (RANGE INTERVAL '1' MINUTE PRECEDING)
```

当您将此查询包含在您的应用程序代码中时，应用程序代码将连续运行。对于 OrderStream 上到达的各个记录，如果在下订单后的 1 分钟窗口内存在交易，则应用程序发送输出。

前述查询中的联接是内部联接，对于 TradeStream 中存在匹配记录的 OrderStream，该查询会在其中发出记录（反之亦然）。使用外部连接可以创建另一个有趣场景。假设您需要查询在提交股票订单的 1 分钟内没有交易的订单，以及在同一窗口内为其他一些订单报告交易。这是外部联接 示例。

```
SELECT STREAM
    ROWTIME,
    o.orderId, o.ticker, o.amount AS orderAmount,
    t.ticker, t.tradeId, t.amount AS tradeAmount,
FROM OrderStream AS o
LEFT OUTER JOIN TradeStream OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS t
ON    o.orderId = t.orderId;
```

迁移到适用于 Apache Flink Studio 的托管服务示例

以下示例演示如何将适用于 SQL 的 Kinesis Data Analytics 应用程序迁移到适用于 Apache Flink Studio 的托管服务。

在适用于 Apache Flink Studio 的托管服务中复制适用于 SQL 的 Kinesis Data Analytics 查询

> **Warning**
>
> 对于新项目，建议您使用适用于 Apache Flink Studio 的托管服务，而不是适用于 SQL 应用程序的 Kinesis Data Analytics。Managed Service for Apache Flink Studio 不仅操作简单，还具有高级分析功能，使您能够在几分钟内构建复杂的流处理应用程序。

本节提供了适用于常见用例的查询转换,以便将您的工作负载迁移到适用于 Apache Flink Studio 的托管服务或适用于 Apache Flink 的托管服务。

> **Note**
>
> 适用于 Apache Flink 的托管服务和适用于 Apache Flink Studio 的托管服务提供了基于 SQL 的 Kinesis Data Analytics 应用程序所不具备的高级数据流处理功能。其中包括恰好一次处理语义、事件时间窗口、通过用户定义的函数和自定义集成实现的可扩展性、命令式语言支持、应用程序持久状态、水平扩展、多数据源支持、可扩展的集成等。这些对确保数据流处理的准确性、完整性、一致性和可靠性至关重要。

在探索这些示例之前，建议您先查看 通过适用于 Apache Flink 的托管服务使用 Studio 笔记本。

主题

在适用于 Apache Flink Studio 的托管服务中重新创建适用于 SQL 的 Kinesis Data Analytics 查询

下表提供了基于 SQL 的 Kinesis Data Analytics 应用程序常见查询到适用于 Apache Flink Studio 的托管服务的转换。

多步应用程序

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "IN_APP_STREAM_001" (
   ingest_time TIMESTAMP,
   ticker_symbol VARCHAR(4),
   sector VARCHAR(16), price REAL, change REAL);
CREATE
OR REPLACE PUMP "STREAM_PUMP_001" AS
INSERT INTO
   "IN_APP_STREAM_001"
   SELECT
      STREAM APPROXIMATE_ARRIVAL_TIME,
      ticker_symbol,
      sector,
      price,
      change FROM "SOURCE_SQL_STREAM_001";
-- Second in-app stream and pump
CREATE
OR REPLACE STREAM "IN_APP_STREAM_02" (ingest_time TIMESTAMP,
   ticker_symbol VARCHAR(4),
   sector VARCHAR(16),
   price REAL,
   change REAL);
CREATE
OR REPLACE PUMP "STREAM_PUMP_02" AS
INSERT INTO
   "IN_APP_STREAM_02"
   SELECT
      STREAM ingest_time,
      ticker_symbol,
      sector,
      price,
      change FROM "IN_APP_STREAM_001";
-- Destination in-app stream and third pump
```

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ingest_time TIMESTAMP,
   ticker_symbol VARCHAR(4),
   sector VARCHAR(16),
   price REAL,
   change REAL);
CREATE
OR REPLACE PUMP "STREAM_PUMP_03" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM ingest_time,
      ticker_symbol,
      sector,
      price,
      change FROM "IN_APP_STREAM_02";
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001;

CREATE TABLE SOURCE_SQL_STREAM_001 (TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(16),
   PRICE DOUBLE,
   CHANGE DOUBLE,
   APPROXIMATE_ARRIVAL_TIME TIMESTAMP(3) METADATA

FROM
   'timestamp' VIRTUAL,
   WATERMARK FOR APPROXIMATE_ARRIVAL_TIME AS APPROXIMATE_ARRIVAL_TIME - INTERVAL '1'
 SECOND )
   PARTITIONED BY (TICKER_SYMBOL) WITH (
      'connector' = 'kinesis',
      'stream' = 'kinesis-analytics-demo-stream',
      'aws.region' = 'us-east-1',
      'scan.stream.initpos' = 'LATEST',
      'format' = 'json',
      'json.timestamp-format.standard' = 'ISO-8601');
DROP TABLE IF EXISTS IN_APP_STREAM_001;

CREATE TABLE IN_APP_STREAM_001 (
   INGEST_TIME TIMESTAMP,
   TICKER_SYMBOL VARCHAR(4),
```

```
   SECTOR VARCHAR(16),
   PRICE DOUBLE,
   CHANGE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
      'connector' = 'kinesis',
      'stream' = 'IN_APP_STREAM_001',
      'aws.region' = 'us-east-1',
      'scan.stream.initpos' = 'LATEST',
      'format' = 'json',
      'json.timestamp-format.standard' = 'ISO-8601');

DROP TABLE IF EXISTS IN_APP_STREAM_02;

CREATE TABLE IN_APP_STREAM_02 (
   INGEST_TIME TIMESTAMP,
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(16),
   PRICE DOUBLE,
   CHANGE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'IN_APP_STREAM_02',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;

CREATE TABLE DESTINATION_SQL_STREAM (
   INGEST_TIME TIMESTAMP, TICKER_SYMBOL VARCHAR(4), SECTOR VARCHAR(16),
   PRICE DOUBLE, CHANGE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'DESTINATION_SQL_STREAM',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');


Query 2 - % flink.ssql(type =
update
)
```

```
   INSERT INTO
      IN_APP_STREAM_001
      SELECT
         APPROXIMATE_ARRIVAL_TIME AS INGEST_TIME,
         TICKER_SYMBOL,
         SECTOR,
         PRICE,
         CHANGE
      FROM
         SOURCE_SQL_STREAM_001;


Query 3 - % flink.ssql(type =
update
)
   INSERT INTO
      IN_APP_STREAM_02
      SELECT
         INGEST_TIME,
         TICKER_SYMBOL,
         SECTOR,
         PRICE,
         CHANGE
      FROM
         IN_APP_STREAM_001;


Query 4 - % flink.ssql(type =
update
)
   INSERT INTO
      DESTINATION_SQL_STREAM
      SELECT
         INGEST_TIME,
         TICKER_SYMBOL,
         SECTOR,
         PRICE,
         CHANGE
      FROM
         IN_APP_STREAM_02;
```

转换 日期时间 值

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
   TICKER VARCHAR(4),
   event_time TIMESTAMP,
   five_minutes_before TIMESTAMP,
   event_unix_timestamp BIGINT,
   event_timestamp_as_char VARCHAR(50),
   event_second INTEGER);

CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM TICKER,
      EVENT_TIME,
      EVENT_TIME - INTERVAL '5' MINUTE,
      UNIX_TIMESTAMP(EVENT_TIME),
      TIMESTAMP_TO_CHAR('yyyy-MM-dd hh:mm:ss', EVENT_TIME),
      EXTRACT(SECOND
   FROM
      EVENT_TIME)
   FROM
      "SOURCE_SQL_STREAM_001"
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER VARCHAR(4),
   EVENT_TIME TIMESTAMP(3),
   FIVE_MINUTES_BEFORE TIMESTAMP(3),
   EVENT_UNIX_TIMESTAMP INT,
   EVENT_TIMESTAMP_AS_CHAR VARCHAR(50),
   EVENT_SECOND INT)

PARTITIONED BY (TICKER) WITH (
   'connector' = 'kinesis', 'stream' = 'kinesis-analytics-demo-stream',
```

```
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         TICKER,
         EVENT_TIME,
         EVENT_TIME - INTERVAL '5' MINUTE AS FIVE_MINUTES_BEFORE,
         UNIX_TIMESTAMP() AS EVENT_UNIX_TIMESTAMP,
         DATE_FORMAT(EVENT_TIME, 'yyyy-MM-dd hh:mm:ss') AS EVENT_TIMESTAMP_AS_CHAR,
         EXTRACT(SECOND
      FROM
         EVENT_TIME) AS EVENT_SECOND
      FROM
         DESTINATION_SQL_STREAM;
```

简单警报

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   ticker_symbol VARCHAR(4),
   sector VARCHAR(12),
   change DOUBLE,
   price DOUBLE);

CREATE
OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT
   STREAM ticker_symbol,
   sector,
   change,
   price
FROM
   "SOURCE_SQL_STREAM_001"
WHERE
   (
```

```
      ABS(Change / (Price - Change)) * 100
   )
   > 1
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;

CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(4),
   CHANGE DOUBLE,
   PRICE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

Query 2 - % flink.ssql(type =
update
)
   SELECT
      TICKER_SYMBOL,
      SECTOR,
      CHANGE,
      PRICE
   FROM
      DESTINATION_SQL_STREAM
   WHERE
      (
         ABS(CHANGE / (PRICE - CHANGE)) * 100
      )
      > 1;
```

受限警报

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "CHANGE_STREAM"(
   ticker_symbol VARCHAR(4),
   sector VARCHAR(12),
   change DOUBLE,
   price DOUBLE);

CREATE
OR REPLACE PUMP "change_pump" AS INSERT INTO "CHANGE_STREAM"
SELECT
   STREAM ticker_symbol,
   sector,
   change,
   price
FROM "SOURCE_SQL_STREAM_001"
WHERE
   (
      ABS(Change / (Price - Change)) * 100
   )
   > 1;
-- ** Trigger Count and Limit **
-- Counts "triggers" or those values that evaluated true against the previous where
 clause
-- Then provides its own limit on the number of triggers per hour per ticker symbol
 to what is specified in the WHERE clause

CREATE
OR REPLACE STREAM TRIGGER_COUNT_STREAM (
   ticker_symbol VARCHAR(4),
   change REAL,
   trigger_count INTEGER);

CREATE
OR REPLACE PUMP trigger_count_pump AS
INSERT INTO
   TRIGGER_COUNT_STREAMSELECT STREAM ticker_symbol,
   change,
   trigger_count
FROM
   (
```

```
      SELECT
         STREAM ticker_symbol,
         change,
         COUNT(*) OVER W1 as trigger_countFROM "CHANGE_STREAM" --window to perform
 aggregations over last minute to keep track of triggers
         WINDOW W1 AS
         (
            PARTITION BY ticker_symbol RANGE INTERVAL '1' MINUTE PRECEDING
         )
   )
WHERE
   trigger_count >= 1;
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;

CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(4),
   CHANGE DOUBLE, PRICE DOUBLE,
   EVENT_TIME AS PROCTIME())
PARTITIONED BY (TICKER_SYMBOL)
WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');
DROP TABLE IF EXISTS TRIGGER_COUNT_STREAM;
CREATE TABLE TRIGGER_COUNT_STREAM (
   TICKER_SYMBOL VARCHAR(4),
   CHANGE DOUBLE,
   TRIGGER_COUNT INT)
PARTITIONED BY (TICKER_SYMBOL);

Query 2 - % flink.ssql(type =
update
)
   SELECT
```

```
      TICKER_SYMBOL,
      SECTOR,
      CHANGE,
      PRICE
   FROM
      DESTINATION_SQL_STREAM
   WHERE
      (
         ABS(CHANGE / (PRICE - CHANGE)) * 100
      )
      > 1;

Query 3 - % flink.ssql(type =
update
)
   SELECT *
   FROM(
         SELECT
            TICKER_SYMBOL,
            CHANGE,
            COUNT(*) AS TRIGGER_COUNT
         FROM
            DESTINATION_SQL_STREAM
         GROUP BY
            TUMBLE(EVENT_TIME, INTERVAL '1' MINUTE),
            TICKER_SYMBOL,
            CHANGE
      )
   WHERE
      TRIGGER_COUNT > 1;
```

从查询中聚合部分结果

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "CALC_COUNT_SQL_STREAM"(
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP,
   TICKERCOUNT DOUBLE);

CREATE
```

```
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP,
   TICKERCOUNT DOUBLE);

CREATE PUMP "CALC_COUNT_SQL_PUMP_001" AS
INSERT INTO
   "CALC_COUNT_SQL_STREAM"(
   "TICKER",
   "TRADETIME",
   "TICKERCOUNT")
   SELECT
      STREAM "TICKER_SYMBOL",
      STEP("SOURCE_SQL_STREAM_001",
      "ROWTIME" BY INTERVAL '1' MINUTE) as "TradeTime",
      COUNT(*) AS "TickerCount "
   FROM
      "SOURCE_SQL_STREAM_001"
   GROUP BY
      STEP("SOURCE_SQL_STREAM_001". ROWTIME BY INTERVAL '1' MINUTE),
      STEP("SOURCE_SQL_STREAM_001"." APPROXIMATE_ARRIVAL_TIME" BY INTERVAL '1'
 MINUTE),
      TICKER_SYMBOL;
CREATE PUMP "AGGREGATED_SQL_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM" (
   "TICKER",
   "TRADETIME",
   "TICKERCOUNT")
   SELECT
      STREAM "TICKER",
      "TRADETIME",
      SUM("TICKERCOUNT") OVER W1 AS "TICKERCOUNT"
   FROM
      "CALC_COUNT_SQL_STREAM" WINDOW W1 AS
      (
         PARTITION BY "TRADETIME" RANGE INTERVAL '10' MINUTE PRECEDING
      )
;
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
```

```
update
) DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001;
CREATE TABLE SOURCE_SQL_STREAM_001 (
   TICKER_SYMBOL VARCHAR(4),
   TRADETIME AS PROCTIME(),
   APPROXIMATE_ARRIVAL_TIME TIMESTAMP(3) METADATA
FROM
   'timestamp' VIRTUAL,
   WATERMARK FOR APPROXIMATE_ARRIVAL_TIME AS APPROXIMATE_ARRIVAL_TIME - INTERVAL '1'
 SECOND)
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');
DROP TABLE IF EXISTS CALC_COUNT_SQL_STREAM;
CREATE TABLE CALC_COUNT_SQL_STREAM (
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP(3),
   WATERMARK FOR TRADETIME AS TRADETIME - INTERVAL '1' SECOND,
   TICKERCOUNT BIGINT NOT NULL ) PARTITIONED BY (TICKER) WITH (
      'connector' = 'kinesis',
      'stream' = 'CALC_COUNT_SQL_STREAM',
      'aws.region' = 'us-east-1',
      'scan.stream.initpos' = 'LATEST',
      'format' = 'csv');
DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;
CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP(3),
   WATERMARK FOR TRADETIME AS TRADETIME - INTERVAL '1' SECOND,
   TICKERCOUNT BIGINT NOT NULL )
   PARTITIONED BY (TICKER) WITH ('connector' = 'kinesis',
      'stream' = 'DESTINATION_SQL_STREAM',
      'aws.region' = 'us-east-1',
      'scan.stream.initpos' = 'LATEST',
      'format' = 'csv');

Query 2 - % flink.ssql(type =
update
)
   INSERT INTO
```

```
      CALC_COUNT_SQL_STREAM
      SELECT
         TICKER,
         TO_TIMESTAMP(TRADETIME, 'yyyy-MM-dd HH:mm:ss') AS TRADETIME,
         TICKERCOUNT
      FROM
         (
            SELECT
               TICKER_SYMBOL AS TICKER,
               DATE_FORMAT(TRADETIME, 'yyyy-MM-dd HH:mm:00') AS TRADETIME,
               COUNT(*) AS TICKERCOUNT
            FROM
               SOURCE_SQL_STREAM_001
            GROUP BY
               TUMBLE(TRADETIME, INTERVAL '1' MINUTE),
               DATE_FORMAT(TRADETIME, 'yyyy-MM-dd HH:mm:00'),
               DATE_FORMAT(APPROXIMATE_ARRIVAL_TIME, 'yyyy-MM-dd HH:mm:00'),
               TICKER_SYMBOL
         )
;

Query 3 - % flink.ssql(type =
update
)
   SELECT
      *
   FROM
      CALC_COUNT_SQL_STREAM;

Query 4 - % flink.ssql(type =
update
)
   INSERT INTO
      DESTINATION_SQL_STREAM
      SELECT
         TICKER,
         TRADETIME,
         SUM(TICKERCOUNT) OVER W1 AS TICKERCOUNT
      FROM
         CALC_COUNT_SQL_STREAM WINDOW W1 AS
         (
            PARTITION BY TICKER
         ORDER BY
            TRADETIME RANGE INTERVAL '10' MINUTE PRECEDING
```

```
         )
;

Query 5 - % flink.ssql(type =
update
)
   SELECT
      *
   FROM
      DESTINATION_SQL_STREAM;
```

转换字符串值

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM for cleaned up referrerCREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" ( "ingest_time" TIMESTAMP, "referrer"
 VARCHAR(32));
CREATE
OR REPLACE PUMP "myPUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT
   STREAM "APPROXIMATE_ARRIVAL_TIME",
   SUBSTRING("referrer", 12,
   (
      POSITION('.com' IN "referrer") - POSITION('www.' IN "referrer") - 4
   )
)
FROM
   "SOURCE_SQL_STREAM_001";
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   referrer VARCHAR(32),
   ingest_time AS PROCTIME() ) PARTITIONED BY (referrer)
WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
```

```
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         ingest_time,
         substring(referrer, 12, 6) as referrer
      FROM
         DESTINATION_SQL_STREAM;
```

使用正则表达式替换子字符串

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM for cleaned up referrerCREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" ( "ingest_time" TIMESTAMP, "referrer"
 VARCHAR(32));
CREATE
OR REPLACE PUMP "myPUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT
   STREAM "APPROXIMATE_ARRIVAL_TIME",
   REGEX_REPLACE("REFERRER", 'http://', 'https://', 1, 0)
FROM
   "SOURCE_SQL_STREAM_001";
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   referrer VARCHAR(32),
   ingest_time AS PROCTIME())
PARTITIONED BY (referrer) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
```

```
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         ingest_time,
         REGEXP_REPLACE(referrer, 'http', 'https') as referrer
      FROM
         DESTINATION_SQL_STREAM;
```

正则表达式模式日志解析

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   sector VARCHAR(24),
   match1 VARCHAR(24),
   match2 VARCHAR(24));
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM T.SECTOR,
      T.REC.COLUMN1,
      T.REC.COLUMN2
   FROM
      (
         SELECT
            STREAM SECTOR,
            REGEX_LOG_PARSE(SECTOR, '.*([E].).*([R].*)') AS REC
         FROM
            SOURCE_SQL_STREAM_001
      )
      AS T;
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
```

```
) CREATE TABLE DESTINATION_SQL_STREAM (
   CHANGE DOUBLE, PRICE DOUBLE,
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(16))
PARTITIONED BY (SECTOR) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
SELECT
   *
FROM
   (
      SELECT
         SECTOR,
         REGEXP_EXTRACT(SECTOR, '.([E].).([R].)', 1) AS MATCH1,
         REGEXP_EXTRACT(SECTOR, '.([E].).([R].)', 2) AS MATCH2
      FROM
         DESTINATION_SQL_STREAM
   )
WHERE
   MATCH1 IS NOT NULL
   AND MATCH2 IS NOT NULL;
```

转换 日期时间 值

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
   TICKER VARCHAR(4),
   event_time TIMESTAMP,
   five_minutes_before TIMESTAMP,
   event_unix_timestamp BIGINT,
   event_timestamp_as_char VARCHAR(50),
   event_second INTEGER);
```

```
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM TICKER,
      EVENT_TIME,
      EVENT_TIME - INTERVAL '5' MINUTE,
      UNIX_TIMESTAMP(EVENT_TIME),
      TIMESTAMP_TO_CHAR('yyyy-MM-dd hh:mm:ss', EVENT_TIME),
      EXTRACT(SECOND
   FROM
      EVENT_TIME)
   FROM
      "SOURCE_SQL_STREAM_001"
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER VARCHAR(4),
   EVENT_TIME TIMESTAMP(3),
   FIVE_MINUTES_BEFORE TIMESTAMP(3),
   EVENT_UNIX_TIMESTAMP INT,
   EVENT_TIMESTAMP_AS_CHAR VARCHAR(50),
   EVENT_SECOND INT) PARTITIONED BY (TICKER)
WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         TICKER,
         EVENT_TIME,
         EVENT_TIME - INTERVAL '5' MINUTE AS FIVE_MINUTES_BEFORE,
         UNIX_TIMESTAMP() AS EVENT_UNIX_TIMESTAMP,
```

```
         DATE_FORMAT(EVENT_TIME, 'yyyy-MM-dd hh:mm:ss') AS EVENT_TIMESTAMP_AS_CHAR,
         EXTRACT(SECOND
      FROM
         EVENT_TIME) AS EVENT_SECOND
      FROM
         DESTINATION_SQL_STREAM;
```

窗口和聚合

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
   event_time TIMESTAMP,
   ticker_symbol VARCHAR(4),
   ticker_count INTEGER);
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM EVENT_TIME,
      TICKER,
      COUNT(TICKER) AS ticker_count
   FROM
      "SOURCE_SQL_STREAM_001" WINDOWED BY STAGGER ( PARTITION BY
         TICKER,
         EVENT_TIME RANGE INTERVAL '1' MINUTE);
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   EVENT_TIME TIMESTAMP(3),
   WATERMARK FOR EVENT_TIME AS EVENT_TIME - INTERVAL '60' SECOND,
   TICKER VARCHAR(4),
   TICKER_COUNT INT) PARTITIONED BY (TICKER)
WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
```

```
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json'

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         EVENT_TIME,
         TICKER, COUNT(TICKER) AS ticker_count
      FROM
         DESTINATION_SQL_STREAM
      GROUP BY
         TUMBLE(EVENT_TIME,
         INTERVAL '60' second),
         EVENT_TIME, TICKER;
```

使用 ROWTIME 的滚动窗口

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   TICKER VARCHAR(4),
   MIN_PRICE REAL,
   MAX_PRICE REAL);
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM TICKER,
      MIN(PRICE),
      MAX(PRICE)
   FROM
      "SOURCE_SQL_STREAM_001"
   GROUP BY
      TICKER,
      STEP("SOURCE_SQL_STREAM_001".
            ROWTIME BY INTERVAL '60' SECOND);
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   ticker VARCHAR(4),
   price DOUBLE,
   event_time VARCHAR(32),
   processing_time AS PROCTIME())
PARTITIONED BY (ticker) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         ticker,
         min(price) AS MIN_PRICE,
         max(price) AS MAX_PRICE
      FROM
         DESTINATION_SQL_STREAM
      GROUP BY
         TUMBLE(processing_time, INTERVAL '60' second),
         ticker;
```

检索最常出现的值 (TOP_K_ITEMS_TUMBLING)

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "CALC_COUNT_SQL_STREAM"(TICKER VARCHAR(4),
   TRADETIME TIMESTAMP,
   TICKERCOUNT DOUBLE);
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP,
```

```
   TICKERCOUNT DOUBLE);
CREATE PUMP "CALC_COUNT_SQL_PUMP_001" AS INSERT INTO "CALC_COUNT_SQL_STREAM" (
   "TICKER",
   "TRADETIME",
   "TICKERCOUNT")
SELECT
   STREAM"TICKER_SYMBOL",
   STEP("SOURCE_SQL_STREAM_001"."ROWTIME" BY INTERVAL '1' MINUTE) as "TradeTime",
   COUNT(*) AS "TickerCount"
FROM
   "SOURCE_SQL_STREAM_001"
GROUP BY STEP("SOURCE_SQL_STREAM_001".
   ROWTIME BY INTERVAL '1' MINUTE),
   STEP("SOURCE_SQL_STREAM_001".
      "APPROXIMATE_ARRIVAL_TIME" BY INTERVAL '1' MINUTE),
   TICKER_SYMBOL;
CREATE PUMP "AGGREGATED_SQL_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM" (
   "TICKER",
   "TRADETIME",
   "TICKERCOUNT")
SELECT
   STREAM "TICKER",
   "TRADETIME",
   SUM("TICKERCOUNT") OVER W1 AS "TICKERCOUNT"
FROM
   "CALC_COUNT_SQL_STREAM" WINDOW W1 AS
   (
      PARTITION BY "TRADETIME" RANGE INTERVAL '10' MINUTE PRECEDING
   )
;
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;
CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER VARCHAR(4),
   EVENT_TIME TIMESTAMP(3),
   WATERMARK FOR EVENT_TIME AS EVENT_TIME - INTERVAL '1' SECONDS )
PARTITIONED BY (TICKER) WITH (
   'connector' = 'kinesis', 'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
```

```
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

Query 2 - % flink.ssql(type =
update
)
   SELECT
      *
   FROM
      (
         SELECT
            TICKER,
            COUNT(*) as MOST_FREQUENT_VALUES,
            ROW_NUMBER() OVER (PARTITION BY TICKER
         ORDER BY
            TICKER DESC) AS row_num
         FROM
            DESTINATION_SQL_STREAM
         GROUP BY
            TUMBLE(EVENT_TIME, INTERVAL '1' MINUTE),
            TICKER
      )
   WHERE
      row_num <= 5;
```

接近排名前 K 的项目

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ITEM VARCHAR(1024), ITEM_COUNT DOUBLE);
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM ITEM,
      ITEM_COUNT
   FROM
      TABLE(TOP_K_ITEMS_TUMBLING(CURSOR(
      SELECT
```

```
         STREAM *
      FROM
         "SOURCE_SQL_STREAM_001"), 'column1', -- name of column in single quotes10,
 -- number of top items60 -- tumbling window size in seconds));
```

Managed Service for Apache Flink Studio

```
%flinkssql
DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001
CREATE TABLE SOURCE_SQL_STREAM_001 ( TS TIMESTAMP(3), WATERMARK FOR TS as TS -
 INTERVAL '5' SECOND, ITEM VARCHAR(1024),
PRICE DOUBLE)
   WITH ( 'connector' = 'kinesis', 'stream' = 'SOURCE_SQL_STREAM_001',
'aws.region' = 'us-east-1', 'scan.stream.initpos' = 'LATEST', 'format' = 'json',
'json.timestamp-format.standard' = 'ISO-8601');


%flink.ssql(type=update)
SELECT
   *
FROM
   (
      SELECT
         *,
         ROW_NUMBER() OVER (PARTITION BY AGG_WINDOW
      ORDER BY
         ITEM_COUNT DESC) as rownum
      FROM
         (
            select
               AGG_WINDOW,
               ITEM,
               ITEM_COUNT
            from
               (
                  select
                     TUMBLE_ROWTIME(TS, INTERVAL '60' SECONDS) as AGG_WINDOW,
                     ITEM,
                     count(*) as ITEM_COUNT
                  FROM
                     SOURCE_SQL_STREAM_001
                  GROUP BY
```

```
                    TUMBLE(TS, INTERVAL '60' SECONDS),
                    ITEM
                )
        )
    )
where
    rownum <= 3
```

分析 Web 日志 (W3C_LOG_PARSE 函数)

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" ( column1 VARCHAR(16),
    column2 VARCHAR(16),
    column3 VARCHAR(16),
    column4 VARCHAR(16),
    column5 VARCHAR(16),
    column6 VARCHAR(16),
    column7 VARCHAR(16));
CREATE
OR REPLACE PUMP "myPUMP" ASINSERT INTO "DESTINATION_SQL_STREAM"
SELECT
    STREAM l.r.COLUMN1,
    l.r.COLUMN2,
    l.r.COLUMN3,
    l.r.COLUMN4,
    l.r.COLUMN5,
    l.r.COLUMN6,
    l.r.COLUMN7
FROM
    (
        SELECT
            STREAM W3C_LOG_PARSE("log", 'COMMON')
        FROM
            "SOURCE_SQL_STREAM_001"
    )
    AS l(r);
```

Managed Service for Apache Flink Studio

```
%flink.ssql(type=update)
```

```
DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001 CREATE TABLE SOURCE_SQL_STREAM_001 ( log
 VARCHAR(1024))
   WITH ( 'connector' = 'kinesis',
          'stream' = 'SOURCE_SQL_STREAM_001',
          'aws.region' = 'us-east-1',
          'scan.stream.initpos' = 'LATEST',
          'format' = 'json',
          'json.timestamp-format.standard' = 'ISO-8601');

% flink.ssql(type=update)
   select
      SPLIT_INDEX(log, ' ', 0),
      SPLIT_INDEX(log, ' ', 1),
      SPLIT_INDEX(log, ' ', 2),
      SPLIT_INDEX(log, ' ', 3),
      SPLIT_INDEX(log, ' ', 4),
      SPLIT_INDEX(log, ' ', 5),
      SPLIT_INDEX(log, ' ', 6)
   from
      SOURCE_SQL_STREAM_001;
```

将字符串拆分到多个字段 (VARIABLE_COLUMN_LOG_PARSE 函数)

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"( "column_A" VARCHAR(16),
   "column_B" VARCHAR(16),
   "column_C" VARCHAR(16),
   "COL_1" VARCHAR(16),
   "COL_2" VARCHAR(16),
   "COL_3" VARCHAR(16));
CREATE
OR REPLACE PUMP "SECOND_STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT
   STREAM t."Col_A",
   t."Col_B",
   t."Col_C",
   t.r."COL_1",
   t.r."COL_2",
   t.r."COL_3"
FROM
```

```
   (
      SELECT
         STREAM "Col_A",
         "Col_B",
         "Col_C",
         VARIABLE_COLUMN_LOG_PARSE ("Col_E_Unstructured",
         'COL_1 TYPE VARCHAR(16),
         COL_2 TYPE VARCHAR(16),
         COL_3 TYPE VARCHAR(16)', ',') AS r
      FROM
         "SOURCE_SQL_STREAM_001"
   )
   as t;
```

Managed Service for Apache Flink Studio

```
%flink.ssql(type=update)
DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001 CREATE TABLE SOURCE_SQL_STREAM_001 ( log
 VARCHAR(1024))
   WITH ( 'connector' = 'kinesis',
          'stream' = 'SOURCE_SQL_STREAM_001',
          'aws.region' = 'us-east-1',
          'scan.stream.initpos' = 'LATEST',
          'format' = 'json',
          'json.timestamp-format.standard' = 'ISO-8601');

% flink.ssql(type=update)
   select
      SPLIT_INDEX(log, ' ', 0),
      SPLIT_INDEX(log, ' ', 1),
      SPLIT_INDEX(log, ' ', 2),
      SPLIT_INDEX(log, ' ', 3),
      SPLIT_INDEX(log, ' ', 4),
      SPLIT_INDEX(log, ' ', 5)
)
from
   SOURCE_SQL_STREAM_001;
```

Joins

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
   ticker_symbol VARCHAR(4),
   "Company" varchar(20),
   sector VARCHAR(12),
   change DOUBLE,
   price DOUBLE);
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM ticker_symbol,
      "c"."Company",
      sector,
      change,
      priceFROM "SOURCE_SQL_STREAM_001"
      LEFT JOIN
         "CompanyName" as "c"
         ON "SOURCE_SQL_STREAM_001".ticker_symbol = "c"."Ticker";
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(12),
   CHANGE INT,
   PRICE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

Query 2 - CREATE TABLE CompanyName (
```

```
   Ticker VARCHAR(4),
   Company VARCHAR(4)) WITH (
      'connector' = 'filesystem',
      'path' = 's3://kda-demo-sample/TickerReference.csv',
      'format' = 'csv' );

Query 3 - % flink.ssql(type =
update
)
   SELECT
      TICKER_SYMBOL,
      c.Company,
      SECTOR,
      CHANGE,
      PRICE
   FROM
      DESTINATION_SQL_STREAM
      LEFT JOIN
         CompanyName as c
         ON DESTINATION_SQL_STREAM.TICKER_SYMBOL = c.Ticker;
```

错误

SQL-based Kinesis Data Analytics application

```
SELECT
   STREAM ticker_symbol,
   sector,
   change,
   (
      price / 0
   )
   as ProblemColumnFROM "SOURCE_SQL_STREAM_001"
WHERE
   sector SIMILAR TO '%TECH%';
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;
CREATE TABLE DESTINATION_SQL_STREAM (
```

```
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(16),
   CHANGE DOUBLE,
   PRICE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

Query 2 - % flink.pyflink @udf(input_types = [DataTypes.BIGINT()],
   result_type = DataTypes.BIGINT()) def DivideByZero(price): try: price / 0
except
: return - 1 st_env.register_function("DivideByZero",
   DivideByZero)

   Query 3 - % flink.ssql(type =
update
)
   SELECT
      CURRENT_TIMESTAMP AS ERROR_TIME,
      *
   FROM
      (
         SELECT
            TICKER_SYMBOL,
            SECTOR,
            CHANGE,
            DivideByZero(PRICE) as ErrorColumn
         FROM
            DESTINATION_SQL_STREAM
         WHERE
            SECTOR SIMILAR TO '%TECH%'
      )
      AS ERROR_STREAM;
```

迁移随机森林砍伐工作负载

如果你想将使用随机森林砍伐 (RCF) 的工作负载从适用于 SQL 的 Kinesis Analytics 迁移到适用于 Apache Flink 的托管服务，请阅读这篇AWS 博文，文中演示了如何使用适用于 Apache Flink 的托管服务运行用于异常检测的在线 RCF 算法。

用 Kinesis Data Streams 代替 Kinesis Data Firehose 源

有关完整教程，请参阅 Converting-KDASQL-KDAStudio/。

在以下练习中，您需要更改数据流，以便使用适用于 Apache Flink Studio 的亚马逊托管服务。即从 Amazon Kinesis Data Firehose 切换到Amazon Kinesis Data Streams。

首先，我们提供一个典型的 KDA-SQL 架构，然后演示如何使用适用于 Apache Flink Studio 的亚马逊托管服务和 Amazon Kinesis Data Streams 取而代之。或者，您可以在此处启动 AWS CloudFormation 模板：

Amazon Kinesis Data Analytics-SQL 和 Amazon Kinesis Data Firehose

以下是 Amazon Kinesis Data Analytics SQL 架构流程：

我们首先研究了传统 Amazon Kinesis Data Analytics-SQL 和 Amazon Kinesis Data Firehose 的设置。用例是一个交易市场，交易数据 (包括股票代码和价格) 从外部来源流向 Amazon Kinesis 系统。适用于 SQL 的 Amazon Kinesis Data Analytics 使用输入流执行窗口式查询，例如滚动窗口，从而确定每只股票在一分钟窗口内的交易量和 `min`、`max` 以及 `average` 交易价格。

描述配置的并行度 (映射到流式传输源的应用程序内流的数量)。处理后，Amazon Kinesis Data Analytics-SQL 将处理后的数据发送到另一个 Amazon Kinesis Data Firehose，然后由后者将输出保存在 Amazon S3 存储桶中。

在本例中，您将使用 Amazon Kinesis 数据生成器。使用 Amazon Kinesis 数据生成器，您可以将测试数据发送到 Amazon Kinesis Data Streams 或 Amazon Kinesis Data Firehose 传输流。请按照此处的说明进行操作。使用此处的 AWS CloudFormation 模板代替说明中提供的模板：

运行 AWS CloudFormation 模板后，输出部分将提供 Amazon Kinesis 数据生成器 url。使用您在此处设置的 Cognito 用户 ID 和密码登录门户。选择区域和目标流名称。当前状态请选择 Amazon Kinesis Data Firehose 传输流。对于新状态，请选择 Amazon Kinesis Data Firehose 流名称。您可以根据需要创建多个模板，然后使用测试模板按钮测试模板，然后再发送到目标流。

以下是使用 Amazon Kinesis 数据生成器的有效负载示例。数据生成器将输入的 Amazon Kinesis Firehose 流作为目标，持续流式处理数据。Amazon Kinesis 软件开发工具包客户端也可以发送来自其他创建器的数据。

```
2023-02-17 09:28:07.763,"AAPL",5032023-02-17 09:28:07.763,
"AMZN",3352023-02-17 09:28:07.763,
"GOOGL",1852023-02-17 09:28:07.763,
"AAPL",11162023-02-17 09:28:07.763,
"GOOGL",1582
```

以下 JSON 用于生成一系列随机交易时间和日期、股票代码和股票价格：

```
date.now(YYYY-MM-DD HH:mm:ss.SSS),
"random.arrayElement(["AAPL","AMZN","MSFT","META","GOOGL"])",
random.number(2000)
```

选择发送数据后，生成器将开始发送模拟数据。

外部系统将数据流式传输到 Amazon Kinesis Data Firehose。在适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 中，您可以使用标准 SQL 分析流数据。该服务用于根据流式传输源创建并运行 SQL 代码，以便执行时间序列分析，馈送实时控制面板和创建实时指标。适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 可以根据输入流上的 SQL 查询创建目标流，然后将目标流发送给另一个 Amazon Kinesis Data Firehose。目标 Amazon Kinesis Data Firehose 可以最终状态将分析数据发送到 Amazon S3。

Amazon Kinesis Data Analytics-SQL 传统代码基于 SQL 标准的扩展。

您可以在 Amazon Kinesis Data Analytics-SQL 中使用以下查询。首先为查询输出创建目标流。然后，您将使用 PUMP，一个 Amazon Kinesis Data Analytics 存储库对象 (SQL 标准的扩展)，其提供持续运行的 `INSERT INTO stream SELECT ... FROM` 查询功能，因此，查询结果可持续输入到指定流中。

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (EVENT_TIME TIMESTAMP,
```

```
INGEST_TIME TIMESTAMP,
TICKER VARCHAR(16),
VOLUME BIGINT,
AVG_PRICE DOUBLE,
MIN_PRICE DOUBLE,
MAX_PRICE DOUBLE);

CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM STEP("SOURCE_SQL_STREAM_001"."tradeTimestamp" BY INTERVAL '60' SECOND) AS
 EVENT_TIME,
      STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND) AS
 "STREAM_INGEST_TIME",
      "ticker",
       COUNT(*) AS VOLUME,
      AVG("tradePrice") AS AVG_PRICE,
      MIN("tradePrice") AS MIN_PRICE,
      MAX("tradePrice") AS MAX_PRICEFROM "SOURCE_SQL_STREAM_001"
   GROUP BY
      "ticker",
      STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND),
      STEP("SOURCE_SQL_STREAM_001"."tradeTimestamp" BY INTERVAL '60' SECOND);
```

以上 SQL 使用两个时间窗口，来自传入流有效负载的 `tradeTimestamp` 和 `ROWTIME`.`tradeTimestamp` (也称为 `Event Time` 或 `client-side time`)。经常需要在分析中使用此时间，因为它是事件发生时的时间。但是，许多事件源（例如手机和 Web 客户端）没有可靠的时钟，这可能会导致时间不准确。此外，连接问题可能会导致记录没有按照事件发生顺序出现在流中。

应用程序内部流也包含名为 `ROWTIME` 的特殊列。该列存储 Amazon Kinesis Data Analytics 在第一个应用程序内部流中插入行的时间戳。`ROWTIME` 反映了 Amazon Kinesis Data Analytics 从流式传输源中读取后将记录插入到第一个应用程序内部流的时间戳。之后，该 `ROWTIME` 值在您的整个应用程序中进行维护。

SQL 在 60 秒的时间间隔内确定股票的计数：`volume`、`min`、`max` 和 `average` 价格。

在基于时间的窗口式查询中使用这些时间有优点也有缺点。选择这些时间中的一个或多个，并根据您的使用案例场景选择一种策略来处理相关缺点。

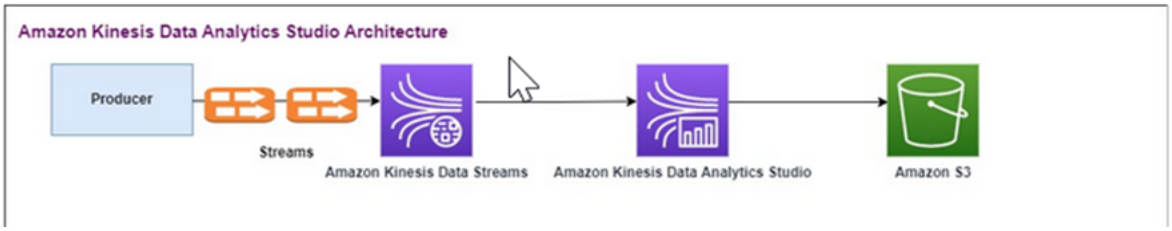
双窗口策略基于不同的时间，即 `ROWTIME` 和其他时间（接收时间或事件时间）中的一个。

- 使用 ROWTIME 作为第一个窗口，控制查询发送结果的频率，如以下示例所示。它不用作逻辑时间。
- 使用其他时间中您希望与分析关联的逻辑时间。该时间表示事件的发生时间。在以下示例中，分析目标是按股票行情机对记录分组并返回计数。

适用于 Apache Flink Studio 的亚马逊托管服务

在更新的架构中，将 Amazon Kinesis Data Firehose 替换为 Amazon Kinesis Data Streams。适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 已替换为 适用于 Apache Flink Studio 的亚马逊托管服务。Apache Flink 代码在 Apache Zeppelin 笔记本中以交互方式运行。Amazon Managed Service for Apache Flink Studio 将聚合的交易数据发送到 Amazon S3 桶中，以便存储。步骤如下：

以下是适用于 Apache Flink Studio 的亚马逊托管服务架构流程：



创建 Kinesis 数据流

使用控制台创建数据流

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航栏中，展开区域选择器并选择一个区域。
3. 选择创建数据流。
4. 在创建 Kinesis 流页面上，输入数据流的名称，然后接受默认的按需容量模式。

   在按需模式下，您可以选择创建 Kinesis 流来创建数据流。

   在 Kinesis stream (Kinesis 流)页面上，当流处于创建中时，流的 Status (状态) 为 Creating (正在创建)。当流可以使用时，Status (状态) 会更改为 Active (有效)。
5. 选择流的名称。Stream Details (流详细信息) 页面显示了流配置摘要以及监控信息。
6. 在 Amazon Kinesis 数据生成器中，将流/传输流更改为新的 Amazon Kinesis Data Streams：TRADE_SOURCE_STREAM。

JSON 和有效负载不变，即与您在 Amazon Kinesis Data Analytics-SQL 中使用的一致。使用 Amazon Kinesis 数据生成器生成一些交易有效负载示例数据，并将 TRADE_SOURCE_STREAM 数据流作为本次练习的目标：

```
{{date.now(YYYY-MM-DD HH:mm:ss.SSS)}},
"{{random.arrayElement(["AAPL","AMZN","MSFT","META","GOOGL"])}}",
{{random.number(2000)}}
```

7. 在 AWS Management Console 上，转至适用于 Apache Flink 的托管服务，然后选择创建应用程序。
8. 在左侧的导航窗格中，选择 工作室笔记本，然后选择 创建工作室笔记本。
9. 输入 Studio 笔记本的名称。
10. 在 AWSGlue 数据库下，提供一个已经存在的 AWS Glue 数据库，用于定义源和目标的元数据。如果您没有 AWS Glue 数据库，请选择创建，然后执行以下操作：

    a. 在 AWS Glue 控制台中，从左侧菜单中选择 Data catalog（数据目录）下的 Databases（数据库）。

    b. 选择 创建数据库。

    c. 在 创建数据库页面中，输入数据库的名称。在 位置 - 可选 部分中，选择 Browse Amazon S3（浏览 Amazon S3），然后选择 Amazon S3 桶。如果尚未设置 Amazon S3 桶，您可以跳过此步骤，稍后再返回。

    d. （可选）。输入数据库的描述。

    e. 选择 Create database（创建数据库）。
11. 选择 创建笔记本。
12. 创建笔记本后，选择运行。
13. 成功启动笔记本后，选择在 Apache Zeppelin 中打开，启动 Zeppelin 笔记本。
14. 在 Zeppelin 笔记本页面上，选择创建新笔记并将其命名为 MarketDataFeed。

Flink SQL 代码的说明如下，但首先请参阅 Zeppelin 笔记本屏幕外观。笔记本中的每个窗口都是一个单独的代码块，一次只能运行一个。

适用于 Apache Flink 的亚马逊托管服务代码

适用于 Apache Flink 的亚马逊托管服务使用 Zeppelin 笔记本来运行代码。在本示例中，已基于 Apache Flink 1.13 进行 ssql 代码映射。Zeppelin 笔记本中的代码如下所示，一次运行于一个数据块。

在 Zeppelin 笔记本中运行任何代码之前，必须先运行 Flink 配置命令。如果在运行代码（ssql、Python 或 Scala）后需要更改任何配置设置，则需要停止并重启笔记本。在此示例中，您需要设置检查点。必须设置检查点，以便您可以将数据流式传输到 Amazon S3 中的文件。从而实现向 Amazon S3 进行的数据流式传输刷新到文件。以下语句将间隔设置为 5000 毫秒。

```
%flink.conf
execution.checkpointing.interval 5000
```

`%flink.conf` 表示此数据块属于配置语句。有关 Flink 配置 (包括检查点) 的更多信息，请参阅 Apache Flink 检查点。

源 Amazon Kinesis Data Streams 的输入表使用下面的 Flink ssql 代码创建。请注意，`TRADE_TIME` 字段存储数据生成器创建的日期/时间。

```
%flink.ssql

DROP TABLE IF EXISTS TRADE_SOURCE_STREAM;
CREATE TABLE TRADE_SOURCE_STREAM (--`arrival_time` TIMESTAMP(3) METADATA FROM
 'timestamp' VIRTUAL,
TRADE_TIME TIMESTAMP(3),
WATERMARK FOR TRADE_TIME as TRADE_TIME - INTERVAL '5' SECOND,TICKER STRING,PRICE
 DOUBLE,
STATUS STRING)WITH ('connector' = 'kinesis','stream' = 'TRADE_SOURCE_STREAM',
'aws.region' = 'us-east-1','scan.stream.initpos' = 'LATEST','format' = 'csv');
```

您可以使用以下语句查看输入流：

```
%flink.ssql(type=update)-- testing the source stream

select * from TRADE_SOURCE_STREAM;
```

在将汇总数据发送到 Amazon S3 之前，您可以使用滚动窗口选择查询直接在适用于 Apache Flink 的亚马逊托管服务中查看。进行该操作后，交易数据会汇总在一分钟时间窗口内。请注意，%flink.ssql 语句必须进行 (type=update) 指定：

```
%flink.ssql(type=update)

select TUMBLE_ROWTIME(TRADE_TIME,
INTERVAL '1' MINUTE) as TRADE_WINDOW,
TICKER, COUNT(*) as VOLUME,
```

```
AVG(PRICE) as AVG_PRICE,
MIN(PRICE) as MIN_PRICE,
MAX(PRICE) as MAX_PRICE FROM TRADE_SOURCE_STREAMGROUP BY TUMBLE(TRADE_TIME, INTERVAL
 '1' MINUTE), TICKER;
```

然后，您才可以在 Amazon S3 中创建目标表。你需要使用水印。水印是一种进度指标，它表示您确信不会再出现延迟事件的时间点。使用水印的原因是为了考虑到达延迟。间隔‘5’ Second 允许交易延迟5秒输入 Amazon Kinesis 数据流，如果窗口内有时间戳，则仍包含在内。有关更多信息，请参阅生成水印。

```
%flink.ssql(type=update)

DROP TABLE IF EXISTS TRADE_DESTINATION_S3;
CREATE TABLE TRADE_DESTINATION_S3 (
TRADE_WINDOW_START TIMESTAMP(3),
WATERMARK FOR TRADE_WINDOW_START as TRADE_WINDOW_START - INTERVAL '5' SECOND,
TICKER STRING,
VOLUME BIGINT,
AVG_PRICE DOUBLE,
MIN_PRICE DOUBLE,
MAX_PRICE DOUBLE)
WITH ('connector' = 'filesystem','path' = 's3://trade-destination/','format' = 'csv');
```

此语句将数据插入 TRADE_DESTINATION_S3。TUMPLE_ROWTIME 是滚动窗口上限 (含) 的时间戳。

```
%flink.ssql(type=update)

insert into TRADE_DESTINATION_S3
select TUMBLE_ROWTIME(TRADE_TIME,
INTERVAL '1' MINUTE),
TICKER, COUNT(*) as VOLUME,
AVG(PRICE) as AVG_PRICE,
MIN(PRICE) as MIN_PRICE,
MAX(PRICE) as MAX_PRICE FROM TRADE_SOURCE_STREAM
GROUP BY TUMBLE(TRADE_TIME, INTERVAL '1' MINUTE), TICKER;
```

运行语句 10 到 20 分钟，以便在 Amazon S3 中积累一些数据。然后中止语句。

从而关闭 Amazon S3 中的文件以便进行查看。

以下是内容：

part-2d9eca09-1d57-450f-b583-11b33b089518-0-3

```
"2023-03-01 17:23:59.999",AMZN,16,1014.75,224.0,1855.0
"2023-03-01 17:23:59.999",AAPL,20,935.45,123.0,1972.0
"2023-03-01 17:23:59.999",MSFT,20,847.55,15.0,1963.0
"2023-03-01 17:23:59.999",META,23,968.5217391304348,125.0,1995.0
"2023-03-01 17:23:59.999",GOOGL,21,949.0,43.0,1996.0
"2023-03-01 17:26:59.999",GOOGL,19,957.578947368421,180.0,1968.0
"2023-03-01 17:26:59.999",AAPL,25,969.72,24.0,1939.0
"2023-03-01 17:26:59.999",AMZN,24,1021.875,101.0,1995.0
"2023-03-01 17:26:59.999",META,14,715.1428571428571,16.0,1504.0
"2023-03-01 17:26:59.999",MSFT,18,1050.4444444444443,123.0,1977.0
"2023-03-01 17:30:59.999",AAPL,20,860.3,94.0,1776.0
"2023-03-01 17:30:59.999",GOOGL,19,1078.4736842105262,9.0,1732.0
"2023-03-01 17:30:59.999",MSFT,20,1180.35,181.0,1981.0
"2023-03-01 17:30:59.999",AMZN,20,974.3,50.0,1791.0
"2023-03-01 17:30:59.999",META,21,902.2857142857143,6.0,1896.0
"2023-03-01 17:33:59.999",AAPL,21,1029.8095238095239,121.0,1657.0
"2023-03-01 17:33:59.999",GOOGL,21,1008.6190476190476,62.0,1979.0
"2023-03-01 17:33:59.999",META,21,1119.142857142857,126.0,1916.0
"2023-03-01 17:33:59.999",AMZN,27,1073.2592592592594,179.0,1849.0
"2023-03-01 17:33:59.999",MSFT,10,1353.4,584.0,1996.0
"2023-03-01 17:36:59.999",GOOGL,18,1262.5,15.0,1997.0
"2023-03-01 17:36:59.999",MSFT,19,780.8947368421053,30.0,1558.0
"2023-03-01 17:36:59.999",AAPL,15,1263.1333333333334,185.0,1928.0
"2023-03-01 17:36:59.999",AMZN,20,882.85,153.0,1764.0
"2023-03-01 17:36:59.999",META,28,858.5,14.0,1917.0
"2023-03-01 17:40:59.999",AMZN,22,976.9545454545455,98.0,1878.0
"2023-03-01 17:40:59.999",GOOGL,25,830.32,39.0,1991.0
"2023-03-01 17:40:59.999",MSFT,18,1325.0,28.0,1966.0
"2023-03-01 17:40:59.999",META,19,855.578947368421,96.0,1725.0
"2023-03-01 17:40:59.999",AAPL,16,1134.25,1.0,1939.0
```

您可以使用AWS CloudFormation 模板来创建基础设施。

AWS CloudFormation 会在您的 AWS 账户中创建以下资源：

- Amazon Kinesis Data Streams
- 适用于 Apache Flink Studio 的亚马逊托管服务
- Amazon Glue 数据库
- Amazon S3 存储桶
- 适用于 Apache Flink Studio 的亚马逊托管服务访问相应资源的 IAM 角色和策略

导入笔记本并将 Amazon S3 存储桶名称更改为 AWS CloudFormation 创建的新 Amazon S3 存储桶。

```
%flink.ssql(type=update)

DROP TABLE IF EXISTS  TRADE_DESTINATION_S3;
CREATE TABLE TRADE_DESTINATION_S3 (
TRADE_WINDOW_START TIMESTAMP(3),
WATERMARK FOR TRADE_WINDOW_START as TRADE_WINDOW_START - INTERVAL '5' SECOND,
TICKER STRING,
VOLUME BIGINT,
AVG_PRICE DOUBLE,
MIN_PRICE DOUBLE,
MAX_PRICE DOUBLE)
WITH ('connector' = 'filesystem','path' = 's3://kda-studio-test-stack-markettradinganalyticsc[redacted]/','format' = 'csv');
```

查看更多

以下是一些其他资源，你可以用来详细了解如何使用适用于 Apache Flink Studio 的托管服务：

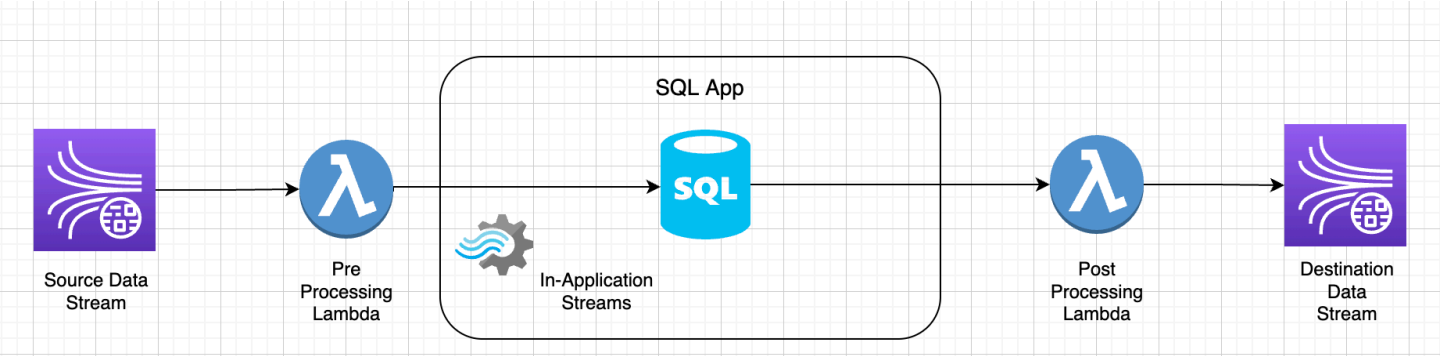
- 适用于 Apache Flink Studio 的托管服务开发人员指南
- Apache Flink 1.13 文档
- 适用于 Apache Flink Studio Workshop 的托管服务
- Apache Flink 窗口化
- Amazon Kinesis Data Analytics 开发人员指南——从 Kinesis Data Analytics 流写入 S3 存储桶

利用用户定义的函数 (UDF)

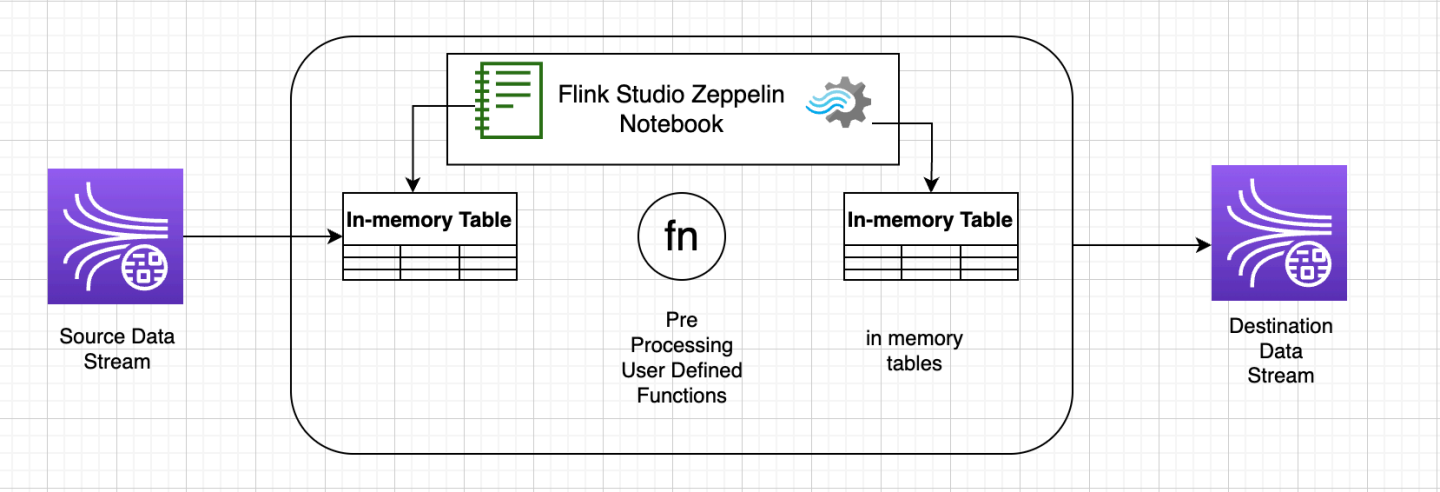
该模式用于演示如何利用 Kinesis Data Analytics-Studio Zeppelin 笔记本中的 UDF 来处理 Kinesis 流中的数据。适用于 Apache Flink Studio 的托管服务使用 Apache Flink 提供高级分析功能，其中包括恰好一次处理语义、事件时间窗口、通过用户定义的函数和客户集成实现的可扩展性、命令式语言支持、应用程序持久状态、水平扩展、多数据源支持、可扩展的集成等。这些对确保数据流处理的准确性、完整性、一致性和可靠性至关重要，也是适用于 SQL 的 Amazon Kinesis Data Analytics 所不具备的功能。

在此示例应用程序中，我们将演示如何利用 KDA-Studio Zeppelin 笔记本中的 UDF 来处理 Kinesis 流中的数据。通过使用适用于 Kinesis Data Analytics 的 Studio 笔记本，您可以实时以交互式方式查询数据流，并使用标准 SQL、Python 和 Scala 轻松构建和运行流处理应用程序。只需在 AWS Management Console 中进行几次单击操作，即可启动无服务器笔记本来查询数据流，并在几秒钟内获得结果。有关更多信息，请参阅将 使用适用于 Apache Flink 的 Kinesis Data Analytics Studio 笔记本。

在 KDA-SQL 应用程序中用于对数据进行预处理/后处理的 Lambda 函数：



用户定义的函数，用于使用 KDA-Studio Zeppelin 笔记本对数据进行预处理/后处理



用户定义的函数（UDF）

引用用户定义的函数来转换数据流可将常见的业务逻辑重新用于运算符。这可以在适用于 Apache Flink Studio 的托管服务笔记本内完成，也可以作为外部引用的应用程序 jar 文件完成。使用用户定义的函数可以简化您可能对流数据执行的转换或数据扩充。

在你的笔记本中，你需要引用一个简单的 Java 应用程序 jar，它具有匿名化个人电话号码的功能。你也可以编写 Python 或 Scala UDF，以便在笔记本中使用。我们选择了一个 Java 应用程序 jar 来突出将应用程序 jar 导入 Pyflink 笔记本的功能。

环境设置

为遵循本指南以及与您的流数据进行交互，您需要使用 AWS CloudFormation 脚本启动以下资源：

- 源和目标 Kinesis Data Streams
- Glue 数据库
- IAM 角色
- 适用于 Apache Flink Studio 的托管服务应用程序
- Lambda 函数 (启动适用于 Apache Flink Studio 的托管服务应用程序)
- 执行上述 Lambda 函数的 Lambda 角色
- 用于调用 Lambda 函数的自定义资源

在这里下载 AWS CloudFormation 模板。

创建 AWS CloudFormation 堆栈。

1. 转至 AWS Management Console 然后在服务列表下选择 CloudFormation。
2. 在 CloudFormation 页面，选择 堆栈，然后选择以新资源创建堆栈（标准）。
3. 在创建堆栈页面上，选择上传模板文件，然后选择您之前下载的 `kda-flink-udf.yml`。上传文件，然后选择 下一步。
4. 为模板指定一个名称 (比如 `kinesis-UDF`)，以便记忆。如果您想更改名称，请更新输入参数，例如输入流。选择 Next（下一步）。
5. 在配置堆栈选项页面上，根据需要添加标签，然后选择下一步。
6. 在查看页面上，选中允许创建 IAM 资源的复选框，然后选择提交。

AWS CloudFormation 堆栈可能需要 10 到 15 分钟才能启动，具体取决于您要启动的区域。看到整个堆栈处于 `CREATE_COMPLETE` 状态后，继续操作。

使用适用于 Apache Flink Studio 的托管服务笔记本

通过使用适用于 Kinesis Data Analytics 的 Studio 笔记本，您可以实时以交互式方式查询数据流，并使用标准 SQL、Python 和 Scala 轻松构建和运行流处理应用程序。只需在 AWS Management Console 中进行几次单击操作，即可启动无服务器笔记本来查询数据流，并在几秒钟内获得结果。

笔记本是一个基于 Web 的开发环境。通过使用笔记本，您可以获得简单的交互式开发体验以及 Apache Flink 提供的高级数据流处理功能。Studio 笔记本使用由 Apache Zeppelin 提供支持的笔记本，并使用 Apache Flink 作为流处理引擎。Studio 笔记本将这些技术无缝结合，使所有技能组开发人员都可以对数据流进行高级分析。

Apache Zeppelin 为您的 Studio 笔记本提供了一整套分析工具，包括：

- 数据可视化
- 将数据导出到文件
- 控制输出格式以便分析

使用笔记本

1. 转至 AWS Management Console 然后在服务列表下选择 Amazon Kinesis。
2. 在左侧导航页面上，选择 Analytics 应用程序，然后选择 Studio 笔记本。
3. 验证 KinesisDataAnalyticsStudio 笔记本是否正在运行。

4. 选择笔记本，然后选择在 Apache Zeppelin 中打开。
5. 下载 数据创建器 Zeppelin 笔记本文件，您需要使用该文件读取数据并将数据加载到 Kinesis 流中。
6. 导入 `Data Producer` Zeppelin 笔记本。请务必修改笔记本代码中的输入 `STREAM_NAME` 和 `REGION`。输入流名称可以在AWS CloudFormation堆栈输出中找到。
7. 选择运行此段落按钮，将示例数据插入输入的 Kinesis 数据流，执行数据创建器笔记本。
8. 示例数据加载过程中，下载 MaskPhoneNumber 交互式笔记本，它将读取输入数据，匿名化输入流中的电话号码，并将匿名化数据存储到输出流中。
9. 导入 `MaskPhoneNumber-interactive` Zeppelin 笔记本。
10. 执行笔记本中的每个段落。

    a. 在第 1 段中，通过导入用户定义函数来匿名化电话号码。

    ```
    %flink(parallelism=1)
    import com.mycompany.app.MaskPhoneNumber
    stenv.registerFunction("MaskPhoneNumber", new MaskPhoneNumber())
    ```

    b. 在下一段中，通过创建一个内存表来读取输入流数据。确保流名称和 AWS 区域正确无误。

    ```
    %flink.ssql(type=update)

    DROP TABLE IF EXISTS customer_reviews;

    CREATE TABLE customer_reviews (
    customer_id VARCHAR,
    product VARCHAR,
    review VARCHAR,
    phone VARCHAR
    )
    WITH (
    'connector' = 'kinesis',
    'stream' = 'KinesisUDFSampleInputStream',
    'aws.region' = 'us-east-1',
    'scan.stream.initpos' = 'LATEST',
    'format' = 'json');
    ```

    c. 检查数据是否已加载到内存表中。

    ```
    %flink.ssql(type=update)
    ```

```
select * from customer_reviews
```

d. 调用用户定义的函数对电话号码进行匿名化处理。

```
%flink.ssql(type=update)
select customer_id, product, review, MaskPhoneNumber('mask_phone', phone) as
 phoneNumber from customer_reviews
```

e. 电话号码屏蔽后，创建一个屏蔽号码的视图。

```
%flink.ssql(type=update)

DROP VIEW IF EXISTS sentiments_view;

CREATE VIEW
    sentiments_view
AS
  select customer_id, product, review, MaskPhoneNumber('mask_phone', phone) as
 phoneNumber from customer_reviews
```

f. 验证数据。

```
%flink.ssql(type=update)
select * from sentiments_view
```

g. 为输出的 Kinesis 流创建内存表。确保流名称和 AWS 区域正确无误。

```
%flink.ssql(type=update)

DROP TABLE IF EXISTS customer_reviews_stream_table;

CREATE TABLE customer_reviews_stream_table (
customer_id VARCHAR,
product VARCHAR,
review VARCHAR,
phoneNumber varchar
)
WITH (
'connector' = 'kinesis',
'stream' = 'KinesisUDFSampleOutputStream',
'aws.region' = 'us-east-1',
'scan.stream.initpos' = 'TRIM_HORIZON',
```

```
'format' = 'json');
```

h. 在目标 Kinesis 流中插入更新的记录。

```
%flink.ssql(type=update)
INSERT INTO customer_reviews_stream_table
SELECT customer_id, product, review, phoneNumber
FROM sentiments_view
```

i. 查看和验证来自目标 Kinesis 流的数据。

```
%flink.ssql(type=update)
select * from customer_reviews_stream_table
```

将笔记本发布为应用程序

现在，您已经以交互方式对笔记本代码进行了测试，接下来您需要把代码部署为具有持久状态的流应用程序。您需要先修改应用程序配置，以便在 Amazon S3 中为您的代码指定一个位置。

1. 在 AWS Management Console，选择您的笔记本，然后在部署为应用程序配置 (可选) 中，选择编辑。
2. 在 Amazon S3 中代码目标项下，选择通过AWS CloudFormation 脚本创建的 Amazon S3 存储桶。此过程可能耗时数分钟。
3. 您将无法按原样发布笔记。如果尝试按原样发布，您将遇到不支持 `Select` 语句的报错。要避免出现此问题，请下载 MaskPhoneNumber 流式处理Zeppelin 笔记本。
4. 导入 `MaskPhoneNumber-streaming` Zeppelin 笔记本。
5. 打开笔记，然后选择 KinesisDataAnalyticsStudio 操作。
6. 选择构建 maskPhoneNumber 流式处理并导出到 S3。确保重命名应用程序名称，请勿使用任何特殊字符。
7. 选择构建并导出。设置流应用程序将花费数分钟。
8. 构建完成后，选择使用 AWS 控制台部署。
9. 在下一页，查看设置并确保选择正确的 IAM 角色。接下来，选择创建流应用程序。
10. 几分钟后，您将看到一条消息，提示流应用程序已成功创建。

部署具有持久状态和限制的应用程序相关更多信息，请参阅部署为具有持久状态的应用程序。

清除

或者，您现在也可以卸载 AWS CloudFormation 堆栈。该操作将删除您之前设置的所有服务。

适用于 SQL 的 Kinesis Data Analytics 示例

此部分提供在 Amazon Kinesis Data Analytics 中创建和使用应用程序的示例。它们包括示例代码和分步说明，以帮助您创建 Kinesis Data Analytics 应用程序和测试结果。

在开始探索这些示例之前，建议您先查看适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理和适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 入门。

主题

示例：转换数据

有时，在 Amazon Kinesis Data Analytics 中执行任何分析之前，应用程序代码必须先预处理传入记录。这种情况是由多种原因导致的，例如，不符合支持的记录格式的记录会导致应用程序内部输入流中出现非规范化的列。

此部分提供了有关如何使用可用的字符串函数来规范化数据、如何从字符串列中提取所需信息等操作的示例，还指明了您可能发现很有用的日期时间函数。

使用 Lambda 预处理流

有关使用预处理流的信息 AWS Lambda，请参见使用 Lambda 函数预处理数据。

主题

示例：转换字符串值

对于流式传输源中的记录，Amazon Kinesis Data Analytics 支持 JSON 和 CSV 等格式。有关详细信息，请参阅 RecordFormat。然后，这些记录根据输入配置映射到应用程序内部流中的行。有关详细信息，请参阅 配置应用程序输入。输入配置指定流式传输源中的记录字段如何映射到应用程序内部流中的列。

当流式传输源中的记录遵循支持的格式时，此映射有效，这会导致应用程序内部流具有规范化数据。但是，如果流式传输源中的数据不符合支持的标准怎么办？ 例如，如果流式传输源包含点击流数据、IoT 传感器和应用程序日志等数据时该怎么办呢？

请考虑以下示例：

- 流式源包含应用程序日志 - 应用程序日志遵循标准 Apache 日志格式，并使用 JSON 格式写入到流。

  ```
  {
     "Log":"192.168.254.30 - John [24/May/2004:22:01:02 -0700] "GET /icons/
  apache_pb.gif HTTP/1.1" 304 0"
  }
  ```

  有关标准 Apache 日志格式的更多信息，请参阅 Apache 网站上的日志文件。

- 流式源包含半结构化数据 - 以下示例显示了两种记录。Col_E_Unstructured 字段值是一系列逗号分隔的值。这里总共有五列：前四列包含字符串类型的值，最后一列包含逗号分隔的值。

  ```
  { "Col_A" : "string",
    "Col_B" : "string",
    "Col_C" : "string",
    "Col_D" : "string",
    "Col_E_Unstructured" : "value,value,value,value"}

  { "Col_A" : "string",
    "Col_B" : "string",
    "Col_C" : "string",
    "Col_D" : "string",
    "Col_E_Unstructured" : "value,value,value,value"}
  ```

- 流式传输源中的记录包含 URL，需要部分 URL 域名才能进行分析。

```
{ "referrer" : "http://www.amazon.com"}
{ "referrer" : "http://www.stackoverflow.com" }
```

此种情况下，以下包含两个步骤的过程通常适用于创建包含规范化数据的应用程序内部流：

1. 配置应用程序输入以便将非结构化字段映射到创建的应用程序内部输入流中的 VARCHAR(N) 类型的列。
2. 在应用程序代码中，使用字符串功能将这一个列拆分为多个列，并将行保存到其他应用程序内部流。应用程序代码创建的该应用程序内部流将包含规范化数据。然后，您可以对该应用程序内部流进行分析。

Amazon Kinesis Data Analytics 提供以下字符串操作、标准 SQL 函数和 SQL 标准扩展以处理字符串列：

- 字符串运算符 - 在比较字符串时，LIKE 和 SIMILAR 等运算符是非常有用的。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的字符串运算符。
- SQL 函数 - 在处理各种字符串时，以下函数是非常有用的。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的字符串和搜索函数。
  - CHAR_LENGTH – 提供字符串的长度。
  - INITCAP – 返回输入字符串的转换后版本，在这类字符串中，每个以空格分隔的单词的第一个字符都是大写的，所有其他字符都是小写的。
  - LOWER/UPPER – 将字符串转换为小写或大写。
  - OVERLAY – 使用第二个字符串参数 (替代字符串) 替换第一个字符串参数的一部分 (原始字符串)。
  - POSITION - 在某个字符串中搜索其他字符串。
  - REGEX_REPLACE – 将子字符串替换为备用子字符串。
  - SUBSTRING - 从特定位置开始提取部分源字符串。
  - TRIM – 从源字符串的开头或结尾删除指定字符的实例。
- SQL 扩展 – 这对使用日志和 URI 等非结构化字符串非常有用。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的日志解析函数。
  - FAST_REGEX_LOG_PARSER – 工作原理类似于正则表达式分析程序，但方法更简便，生成结果的速度更快。例如，较快的正则表达式解析程序会在找到第一个匹配项时停止（称为延迟语义）。
  - FIXED_COLUMN_LOG_PARSE – 分析固定宽度的字段，自动将其转换为指定的 SQL 类型。

- REGEX_LOG_PARSE – 根据默认的 Java 正则表达式模式分析字符串。
- SYS_LOG_PARSE – 分析 UNIX/Linux 系统日志中的常见条目。
- VARIABLE_COLUMN_LOG_PARSE – 将输入字符串拆分为多个由分隔符或分隔符字符串分隔的字段。
- W3C_LOG_PARSE – 可用于快速格式化 Apache 日志。

有关使用这些函数的示例，请参阅以下主题：

主题

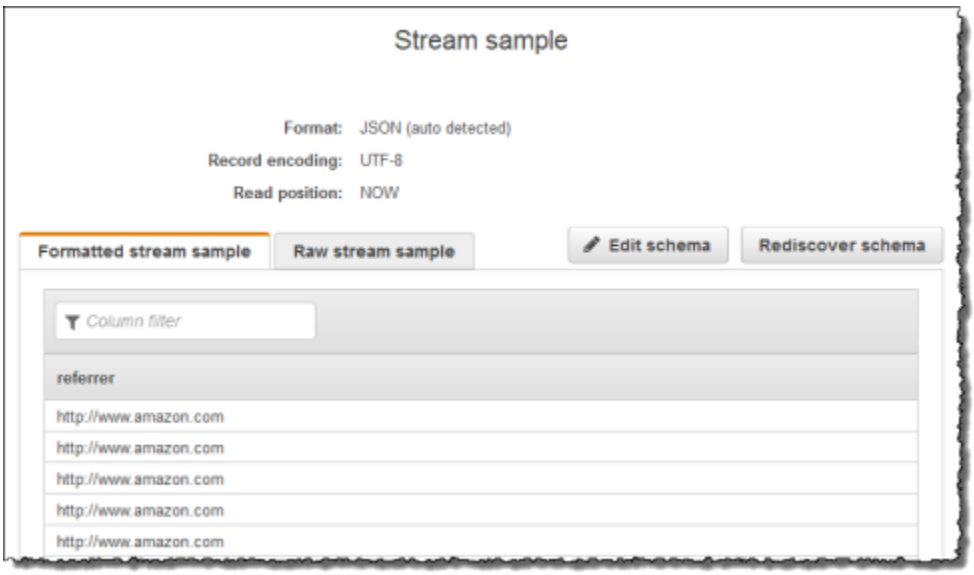
示例：提取部分字符串 (SUBSTRING 函数)

此示例使用 SUBSTRING 函数在 Amazon Kinesis Data Analytics 中转换字符串。SUBSTRING 函数从特定位置开始提取部分源字符串。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的 子字符串。

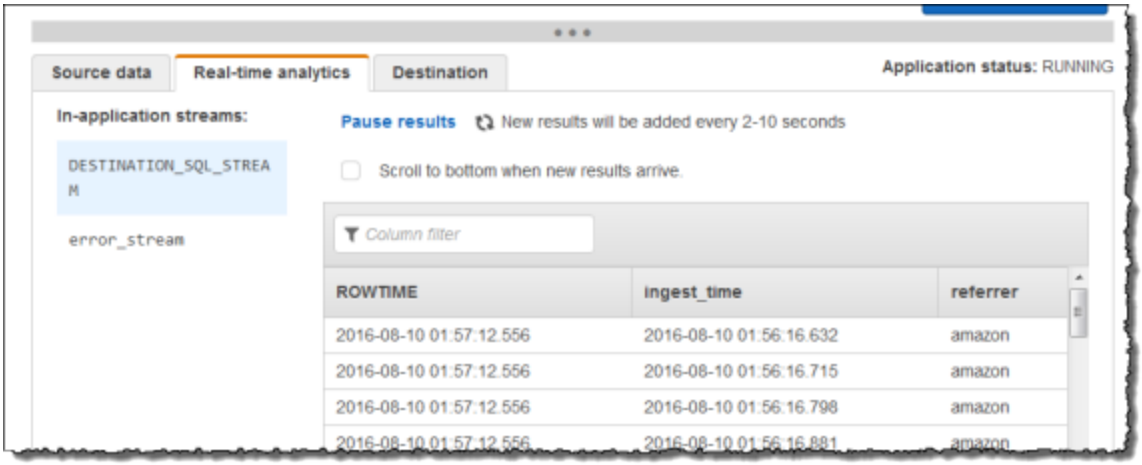
在本示例中，您将以下记录写入到 Amazon Kinesis 数据流中。

```
{ "REFERRER" : "http://www.amazon.com" }
{ "REFERRER" : "http://www.amazon.com"}
{ "REFERRER" : "http://www.amazon.com"}
...
```

然后，您在控制台上创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 数据流作为流式传输源。发现过程读取有关流式传输源的示例记录，并推断出具有一列 (REFERRER) 的应用程序内部架构，如下所示。



然后，您可以将应用程序代码与 SUBSTRING 函数结合使用，来解析 URL 字符串以检索公司名称。随后将结果数据插入另一个应用程序内部流，如下所示：



主题

- 步骤 1：创建 Kinesis 数据流
- 步骤 2：创建 Kinesis Data Analytics 应用程序

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis 数据流并填充日志记录，如下所示：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。

2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流)，然后创建带有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
4. 运行以下 Python 代码以便填充示例日志记录。这段简单代码不断地将同一日志记录写入到流中。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {"REFERRER": "http://www.amazon.com"}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Kinesis Data Analytics 应用程序

接下来，创建一个 Kinesis Data Analytics 应用程序，如下所示：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择 创建应用程序，键入应用程序名称，然后选择 创建应用程序。
3. 在应用程序详细信息页面上，选择 连接流数据。
4. 在 连接到源 页面上，执行以下操作：

   a. 选择在上一部分中创建的流。

b. 选择创建 IAM 角色的选项。

c. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。推断的架构仅包含一列。

d. 选择 保存并继续。

5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择是，启动应用程序。

6. 在 SQL 编辑器中编写应用程序代码并确认结果如下所示：

a. 复制下面的应用程序代码并将其粘贴到编辑器中。

```
-- CREATE OR REPLACE STREAM for cleaned up referrer
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "ingest_time" TIMESTAMP,
    "referrer" VARCHAR(32));

CREATE OR REPLACE PUMP "myPUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM
         "APPROXIMATE_ARRIVAL_TIME",
         SUBSTRING("referrer", 12, (POSITION('.com' IN "referrer") -
 POSITION('www.' IN "referrer") - 4))
      FROM "SOURCE_SQL_STREAM_001";
```

b. 选择 保存并运行 SQL。在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

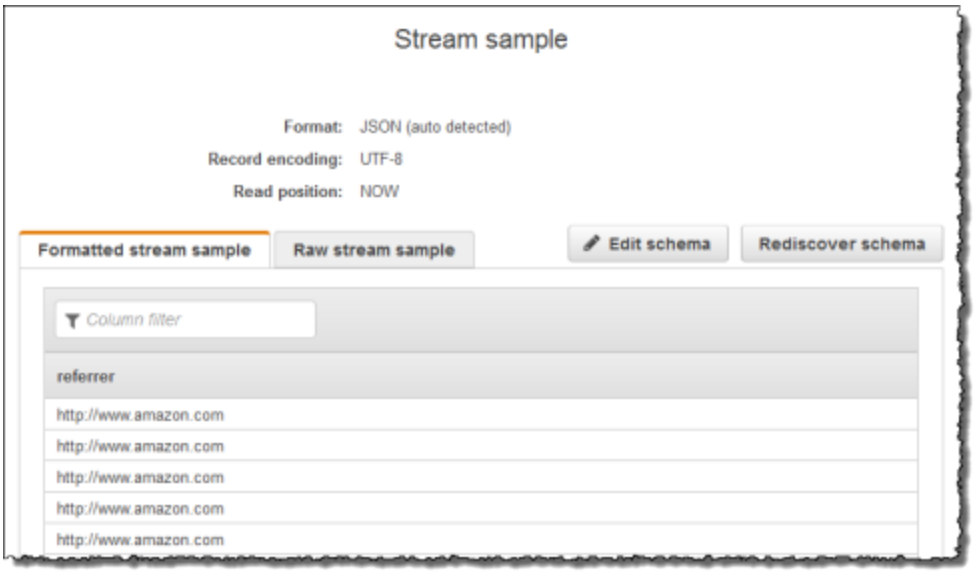
示例：使用正则表达式替换子字符串 (REGEX_REPLACE 函数)

此示例使用 REGEX_REPLACE 函数在 Amazon Kinesis Data Analytics 中转换字符串。REGEX_REPLACE 将子字符串替换为备用子字符串。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的 REGEX_REPLACE。

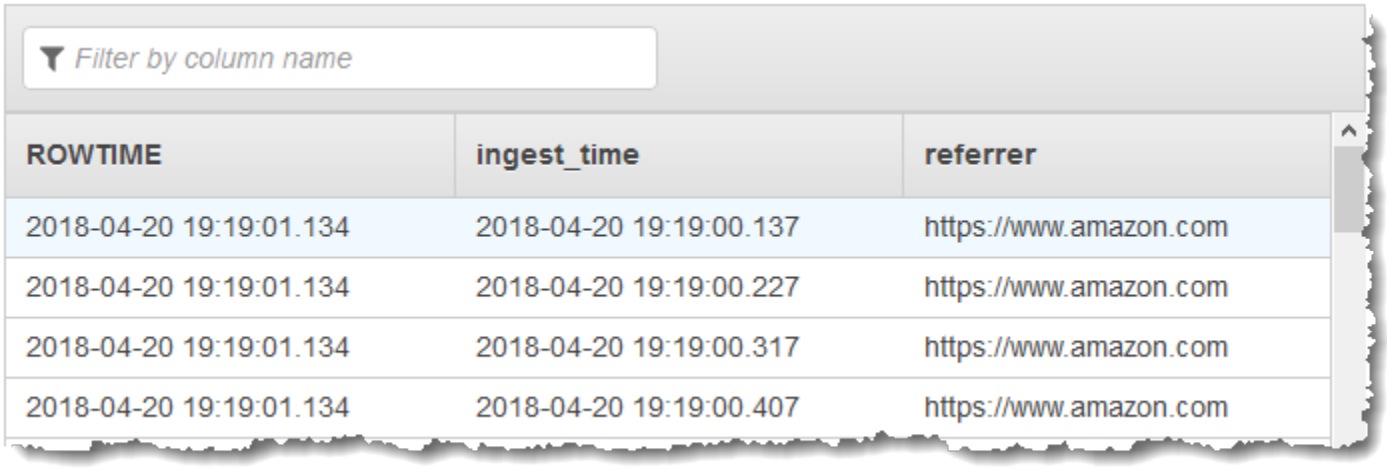
在本示例中，您将以下记录写入到 Amazon Kinesis 数据流中：

```
{ "REFERRER" : "http://www.amazon.com" }
{ "REFERRER" : "http://www.amazon.com"}
{ "REFERRER" : "http://www.amazon.com"}
...
```

然后，您在控制台上创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 数据流作为流式传输源。发现过程读取有关流式传输源的示例记录，并推断出具有一列 (REFERRER) 的应用程序内部架构，如下所示。



然后，通过 REGEX_REPLACE 函数使用应用程序代码将 URL 转换为使用 `https://` 而不是 `http://`。随后将结果数据插入另一个应用程序内部流，如下所示：



主题

- 步骤 1：创建 Kinesis 数据流
- 步骤 2：创建 Kinesis Data Analytics 应用程序

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis 数据流并填充日志记录，如下所示：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流，然后创建带有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
4. 运行以下 Python 代码以便填充示例日志记录。这段简单代码不断地将同一日志记录写入到流中。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {"REFERRER": "http://www.amazon.com"}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Kinesis Data Analytics 应用程序

接下来，创建一个 Kinesis Data Analytics 应用程序，如下所示：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择 创建应用程序，键入应用程序名称，然后选择 创建应用程序。
3. 在应用程序详细信息页面上，选择 连接流数据。
4. 在 连接到源 页面上，执行以下操作：

   a. 选择在上一部分中创建的流。

   b. 选择创建 IAM 角色的选项。

   c. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。推断的架构仅包含一列。

   d. 选择 保存并继续。

5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择是，启动应用程序。
6. 在 SQL 编辑器中编写应用程序代码并确认结果，如下所示：

   a. 复制下面的应用程序代码并将其粘贴到编辑器中：

```
-- CREATE OR REPLACE STREAM for cleaned up referrer
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "ingest_time" TIMESTAMP,
    "referrer" VARCHAR(32));

CREATE OR REPLACE PUMP "myPUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM
         "APPROXIMATE_ARRIVAL_TIME",
         REGEX_REPLACE("REFERRER", 'http://', 'https://', 1, 0)
      FROM "SOURCE_SQL_STREAM_001";
```

   b. 选择 保存并运行 SQL。在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

示例：根据正则表达式分析日志字符串 (REGEX_LOG_PARSE 函数)

此示例使用 REGEX_LOG_PARSE 函数在 Amazon Kinesis Data Analytics 中转换字符串。REGEX_LOG_PARSE 根据默认 Java 正则表达式模式分析字符串。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的 REGEX_LOG_PARSE。

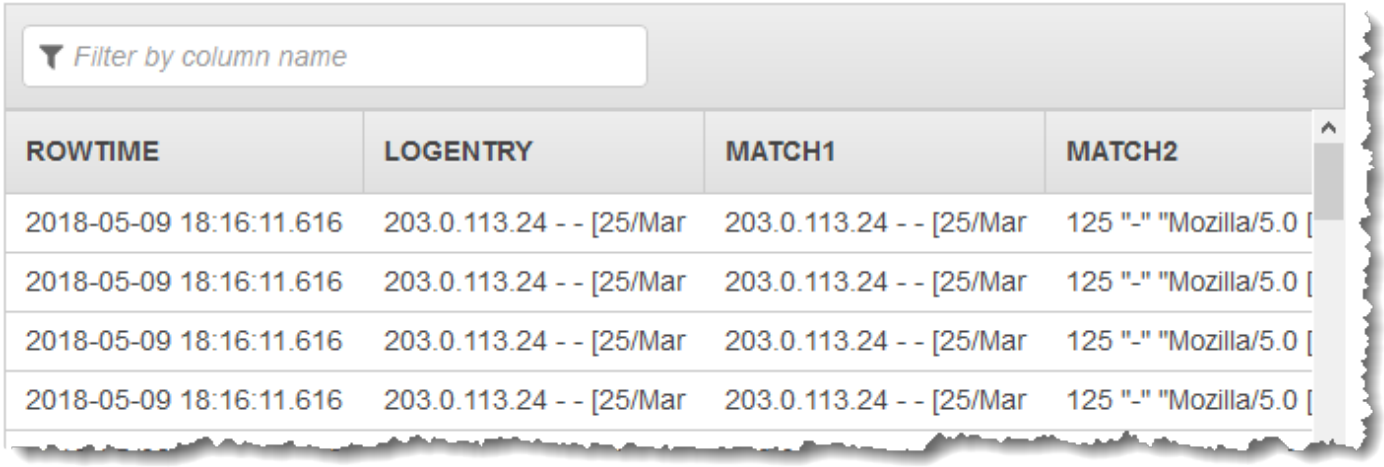
在本示例中，您将以下记录写入到 Amazon Kinesis 流中。

```
{"LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] \"GET /index.php HTTP/1.1\"
 200 125 \"-\" \"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0\""}
{"LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] \"GET /index.php HTTP/1.1\"
 200 125 \"-\" \"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0\""}
{"LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] \"GET /index.php HTTP/1.1\"
 200 125 \"-\" \"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0\""}
...
```

然后，您在控制台上创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 数据流作为流式传输源。发现过程读取有关流式传输源的示例记录，并推断出具有一列 (LOGENTRY) 的应用程序内部架构，如下所示。

ROWTIME TIMESTAMP	**LOGENTRY** VARCHAR(256)
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"

然后，在 REGEX_LOG_PARSE 函数中使用应用程序代码分析日志字符串从而检索数据元素。随后将结果数据插入另一个应用程序内部流，如下面的屏幕截图所示：



主题

- 步骤 1：创建 Kinesis 数据流
- 步骤 2：创建 Kinesis Data Analytics 应用程序

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis 数据流并填充日志记录，如下所示：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流，然后创建带有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
4. 运行以下 Python 代码以便填充示例日志记录。这段简单代码不断地将同一日志记录写入到流中。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "
        '"GET /index.php HTTP/1.1" 200 125 "-" '
```

```
        '"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0"'
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Kinesis Data Analytics 应用程序

接下来，创建一个 Kinesis Data Analytics 应用程序，如下所示：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择 Create application，并指定应用程序名称。
3. 在应用程序详细信息页面上，选择 连接流数据。
4. 在 连接到源 页面上，执行以下操作：

   a. 选择在上一部分中创建的流。

   b. 选择创建 IAM 角色的选项。

   c. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。推断的架构仅包含一列。

   d. 选择 保存并继续。

5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择是，启动应用程序。
6. 在 SQL 编辑器中编写应用程序代码并确认结果如下所示：

   a. 复制下面的应用程序代码并将其粘贴到编辑器中。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (logentry VARCHAR(24), match1
 VARCHAR(24), match2 VARCHAR(24));

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM T.LOGENTRY, T.REC.COLUMN1, T.REC.COLUMN2
    FROM
         (SELECT STREAM LOGENTRY,
             REGEX_LOG_PARSE(LOGENTRY, '(\w.+) (\d.+) (\w.+) (\w.+)') AS REC
             FROM SOURCE_SQL_STREAM_001) AS T;
```

b. 选择 保存并运行 SQL。在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

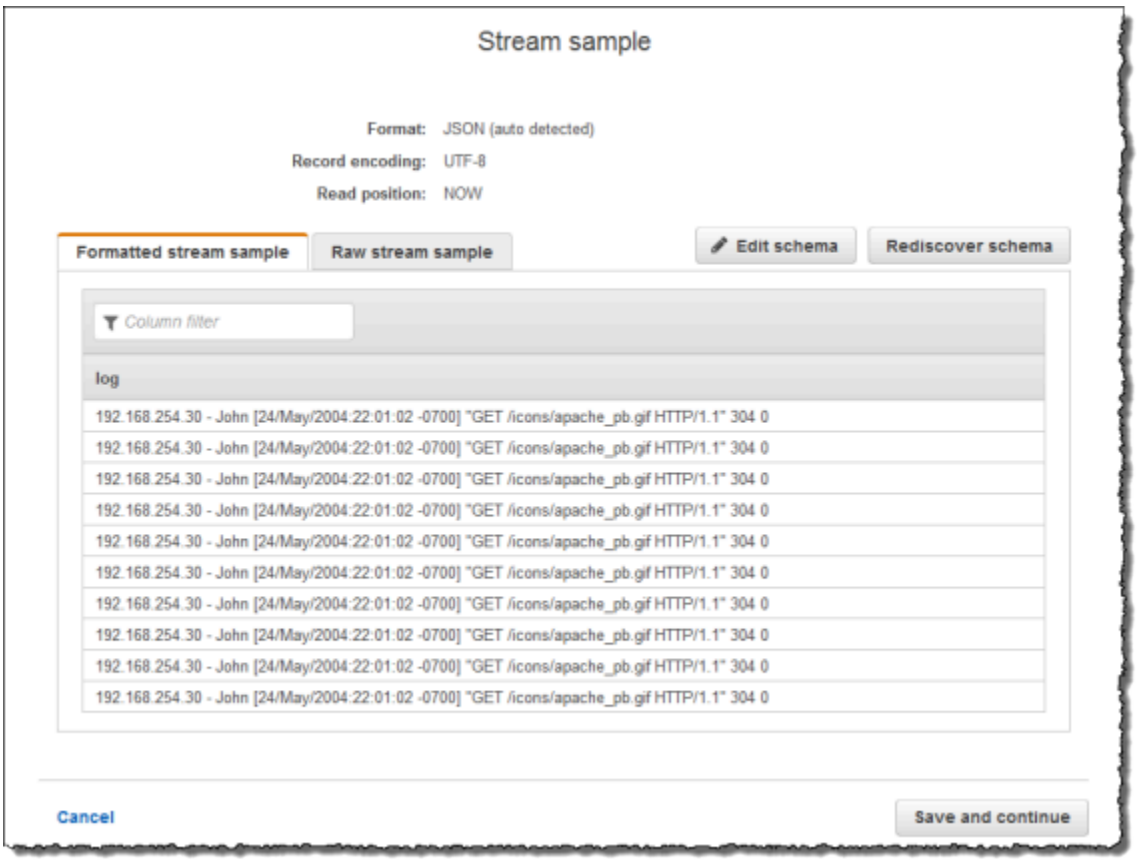
示例：分析 Web 日志 (W3C_LOG_PARSE 函数)

此示例使用 W3C_LOG_PARSE 函数在 Amazon Kinesis Data Analytics 中转换字符串。您可以使用 W3C_LOG_PARSE 快速格式化 Apache 日志。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的 W3C_LOG_PARSE。

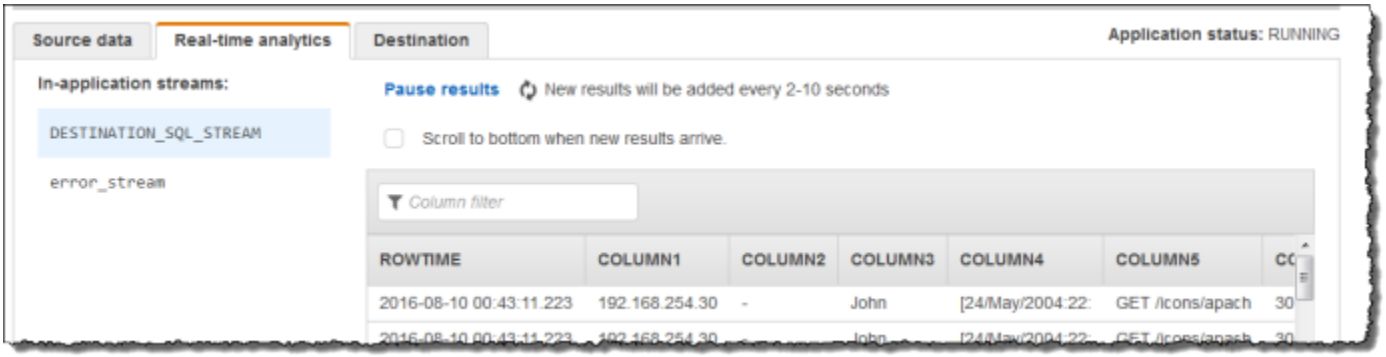
在本示例中，您将日志记录写入到 Amazon Kinesis 数据流中。示例日志如下所示：

```
{"Log":"192.168.254.30 - John [24/May/2004:22:01:02 -0700] "GET /icons/apache_pba.gif
 HTTP/1.1" 304 0"}
{"Log":"192.168.254.30 - John [24/May/2004:22:01:03 -0700] "GET /icons/apache_pbb.gif
 HTTP/1.1" 304 0"}
{"Log":"192.168.254.30 - John [24/May/2004:22:01:04 -0700] "GET /icons/apache_pbc.gif
 HTTP/1.1" 304 0"}
...
```

然后，您在控制台上创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 数据流作为流式传输源。发现过程读取流式传输源上的示例记录，并推断出具有一个列（日志）的应用程序内部架构，如下所示：



然后，您将使用应用程序代码和 W3C_LOG_PARSE 函数解析日志，创建另一应用程序内部流（将不同日志字段放置在单独的列中），如下所示：



主题

- 步骤 1：创建 Kinesis 数据流
- 步骤 2：创建 Kinesis Data Analytics 应用程序

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis Data Stream 并填充日志记录，如下所示：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流，然后创建带有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
4. 运行以下 Python 代码以便填充示例日志记录。这段简单代码不断地将同一日志记录写入到流中。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "log": "192.168.254.30 - John [24/May/2004:22:01:02 -0700] "
        '"GET /icons/apache_pb.gif HTTP/1.1" 304 0'
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Kinesis Data Analytics 应用程序

创建一个 Kinesis Data Analytics 应用程序，如下所示：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。

2. 选择 创建应用程序，键入应用程序名称，然后选择 创建应用程序。
3. 在应用程序详细信息页面上，选择 连接流数据。
4. 在 连接到源 页面上，执行以下操作：
   a. 选择在上一部分中创建的流。
   b. 选择创建 IAM 角色的选项。
   c. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。推断的架构仅包含一列。
   d. 选择 保存并继续。
5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择是，启动应用程序。
6. 在 SQL 编辑器中编写应用程序代码并确认结果如下所示：
   a. 复制下面的应用程序代码并将其粘贴到编辑器中。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
column1 VARCHAR(16),
column2 VARCHAR(16),
column3 VARCHAR(16),
column4 VARCHAR(16),
column5 VARCHAR(16),
column6 VARCHAR(16),
column7 VARCHAR(16));

CREATE OR REPLACE PUMP "myPUMP" AS
INSERT INTO "DESTINATION_SQL_STREAM"
        SELECT STREAM
            l.r.COLUMN1,
            l.r.COLUMN2,
            l.r.COLUMN3,
            l.r.COLUMN4,
            l.r.COLUMN5,
            l.r.COLUMN6,
            l.r.COLUMN7
        FROM (SELECT STREAM W3C_LOG_PARSE("log", 'COMMON')
              FROM "SOURCE_SQL_STREAM_001") AS l(r);
```

b. 选择 保存并运行 SQL。在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

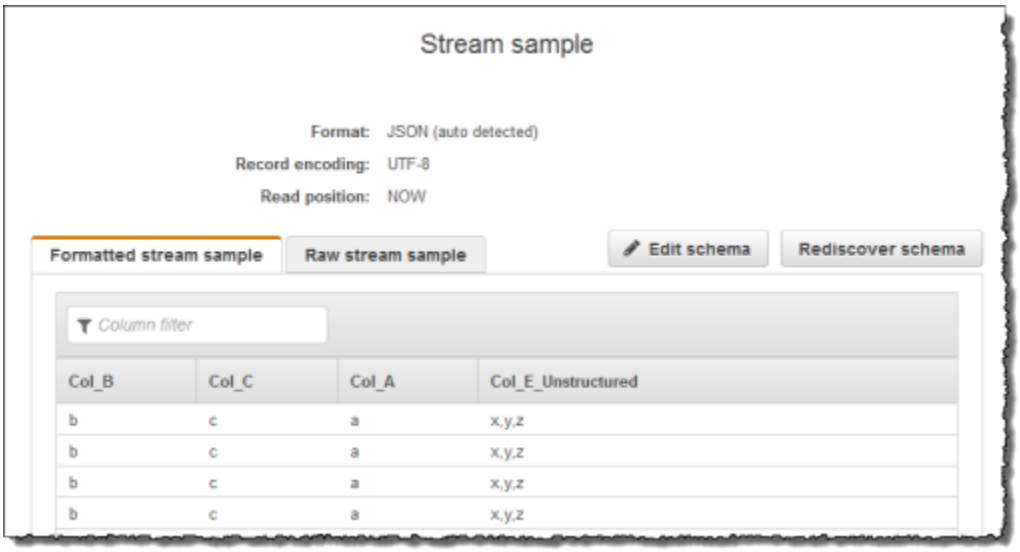
示例：将字符串拆分到多个字段 (VARIABLE_COLUMN_LOG_PARSE 函数)

此示例使用 VARIABLE_COLUMN_LOG_PARSE 函数在 Kinesis Data Analytics 中处理字符串列。VARIABLE_COLUMN_LOG_PARSE 将输入字符串拆分为多个字段，它们由分隔符或分隔符字符串分隔。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的 VARIABLE_COLUMN_LOG_PARSE。

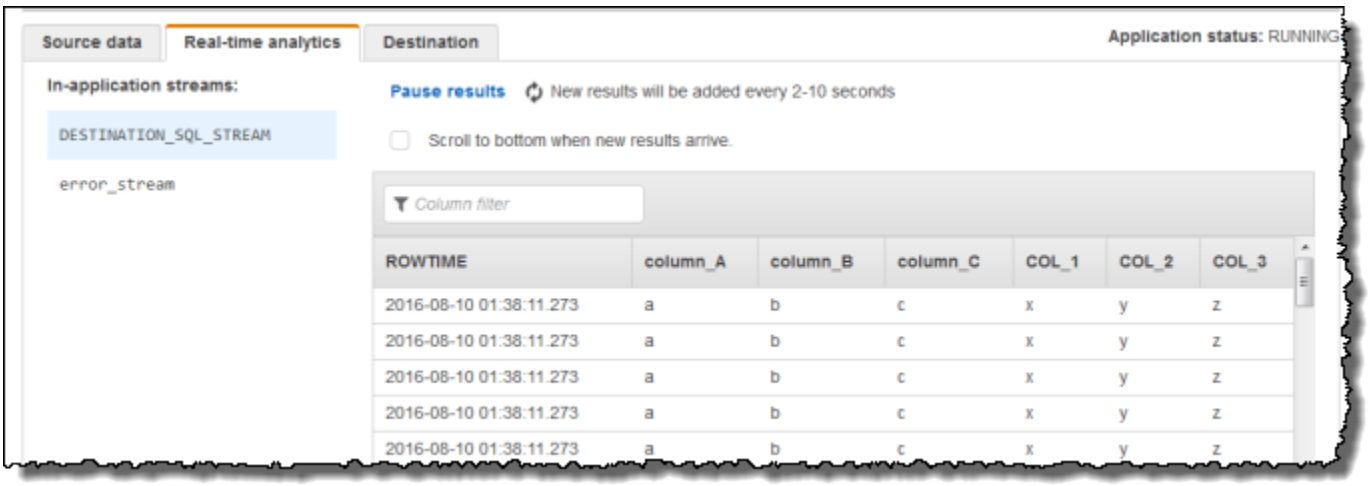
在本示例中，您将半结构化记录写入到 Amazon Kinesis Data Stream 中。示例记录如下所示：

```
{ "Col_A" : "string",
  "Col_B" : "string",
  "Col_C" : "string",
  "Col_D_Unstructured" : "value,value,value,value"}
{ "Col_A" : "string",
  "Col_B" : "string",
  "Col_C" : "string",
  "Col_D_Unstructured" : "value,value,value,value"}
```

然后，您在控制台上创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 流作为流式传输源。发现过程读取流式传输源上的示例记录，并推断出具有四个列的应用程序内部架构，如下所示：



然后，您将使用应用程序代码和 VARIABLE_COLUMN_LOG_PARSE 函数解析逗号分隔的值，将规范化的行插入到其他应用程序内部流，如下所示：



主题

- 步骤 1：创建 Kinesis 数据流
- 步骤 2：创建 Kinesis Data Analytics 应用程序

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis 数据流并填充日志记录，如下所示：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流，然后创建带有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
4. 运行以下 Python 代码以便填充示例日志记录。这段简单代码不断地将同一日志记录写入到流中。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {"Col_A": "a", "Col_B": "b", "Col_C": "c", "Col_E_Unstructured":
 "x,y,z"}
```

```
def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Kinesis Data Analytics 应用程序

创建一个 Kinesis Data Analytics 应用程序，如下所示：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择 创建应用程序，键入应用程序名称，然后选择 创建应用程序。
3. 在应用程序详细信息页面上，选择 连接流数据。
4. 在 连接到源 页面上，执行以下操作：
   a. 选择在上一部分中创建的流。
   b. 选择创建 IAM 角色的选项。
   c. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。请注意推断的架构仅包含一列。
   d. 选择 保存并继续。
5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择是，启动应用程序。
6. 在 SQL 编辑器中，编写应用程序代码并确认结果：
   a. 复制下面的应用程序代码并将其粘贴到编辑器中：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
```

```
            "column_A" VARCHAR(16),
            "column_B" VARCHAR(16),
            "column_C" VARCHAR(16),
            "COL_1" VARCHAR(16),
            "COL_2" VARCHAR(16),
            "COL_3" VARCHAR(16));

CREATE OR REPLACE PUMP "SECOND_STREAM_PUMP" AS
INSERT INTO "DESTINATION_SQL_STREAM"
   SELECT STREAM  t."Col_A", t."Col_B", t."Col_C",
                  t.r."COL_1", t.r."COL_2", t.r."COL_3"
   FROM (SELECT STREAM
           "Col_A", "Col_B", "Col_C",
           VARIABLE_COLUMN_LOG_PARSE ("Col_E_Unstructured",
                                      'COL_1 TYPE VARCHAR(16), COL_2 TYPE
 VARCHAR(16), COL_3 TYPE VARCHAR(16)',
                                      ',') AS r
         FROM "SOURCE_SQL_STREAM_001") as t;
```

b. 选择 保存并运行 SQL。在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

示例：转换 DateTime 值

Amazon Kinesis Data Analytics 支持将列转换为时间戳。例如，除了 ROWTIME 列以外，建议您将自己的时间戳用作 GROUP BY 子句中另一个基于时间的窗口。Kinesis Data Analytics 提供用于处理日期和时间字段的操作和 SQL 函数。

- 日期和时间运算符 - 您可以对日期、时间和间隔数据类型执行算术运算。有关更多信息，请参阅 Amazon Managed Service for Apache Flink SQL 参考中的日期、时间戳和间隔运算符。

- SQL 函数 - 其中包括以下各项。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的日期和时间函数。
  - EXTRACT( ) - 从日期、时间、时间戳或间隔表达式中提取一个字段。
  - CURRENT_TIME - 返回执行查询的时间 (UTC)。
  - CURRENT_DATE - 返回执行查询的日期 (UTC)。
  - CURRENT_TIMESTAMP - 返回执行查询的时间戳 (UTC)。
  - LOCALTIME - 返回 Kinesis Data Analytics 运行环境所定义执行查询的当前时间 (UTC)。

- LOCALTIMESTAMP - 返回 Kinesis Data Analytics 运行环境定义的当前时间戳 (UTC)。

- SQL 扩展 - 其中包括以下各项。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的日期和时间函数以及日期时间转换函数。
  - CURRENT_ROW_TIMESTAMP - 为流中的每一行返回一个新的时间戳。
  - TSDIFF - 返回两个时间戳的差异 (以毫秒为单位)。
  - CHAR_TO_DATE - 将字符串转换为日期。
  - CHAR_TO_TIME - 将字符串转换为时间。
  - CHAR_TO_TIMESTAMP - 将字符串转换为时间戳。
  - DATE_TO_CHAR - 将日期转换为字符串。
  - TIME_TO_CHAR - 将时间转换为字符串。
  - TIMESTAMP_TO_CHAR - 将时间戳转换为字符串。

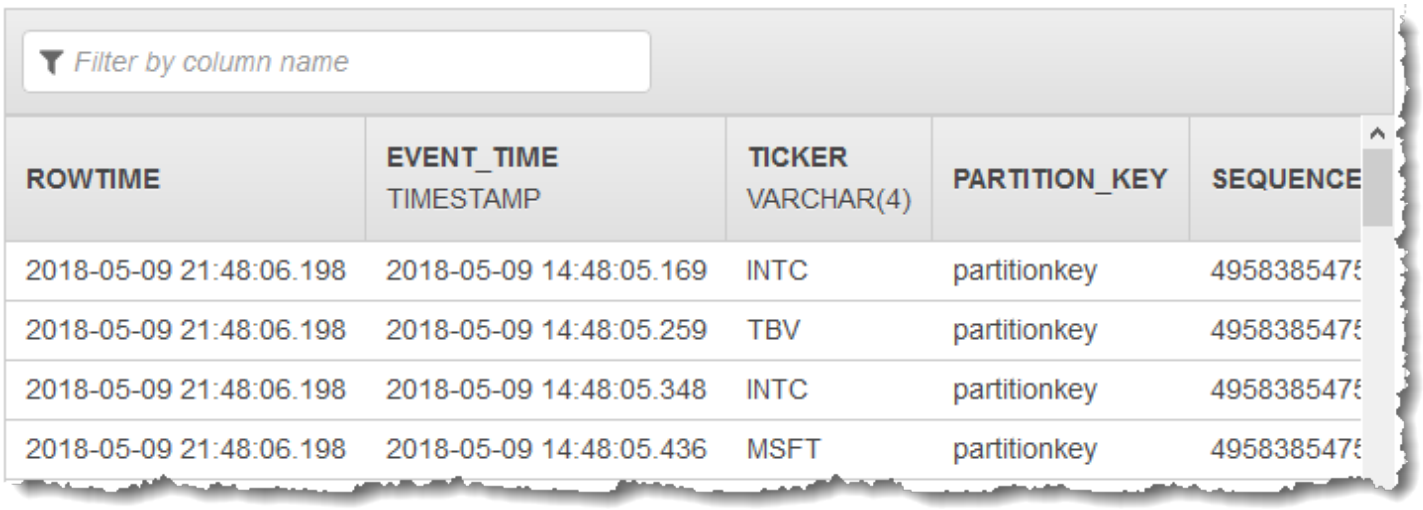
上面大多数 SQL 函数都采用一种格式来转换列。格式是灵活多变的。例如，您可以指定采用格式 yyyy-MM-dd hh:mm:ss 将输入字符串 2009-09-16 03:15:24 转换为时间戳。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的字符到时间戳 (Sys)。

示例：转换日期

在本示例中，您将以下记录写入到 Amazon Kinesis 数据流中。

```
{"EVENT_TIME": "2018-05-09T12:50:41.337510", "TICKER": "AAPL"}
{"EVENT_TIME": "2018-05-09T12:50:41.427227", "TICKER": "MSFT"}
{"EVENT_TIME": "2018-05-09T12:50:41.520549", "TICKER": "INTC"}
{"EVENT_TIME": "2018-05-09T12:50:41.610145", "TICKER": "MSFT"}
{"EVENT_TIME": "2018-05-09T12:50:41.704395", "TICKER": "AAPL"}
...
```

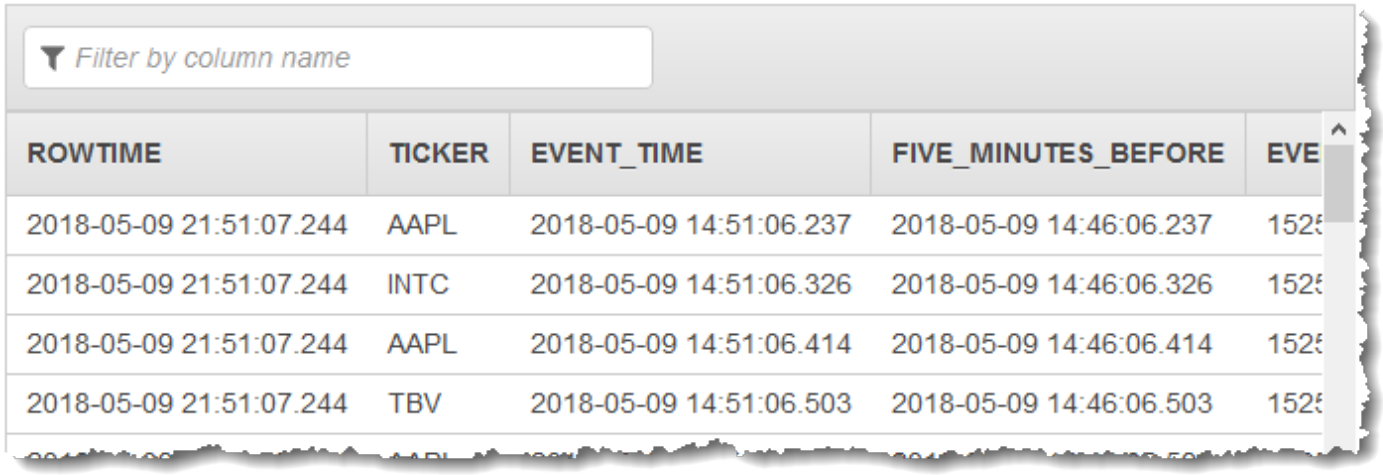
然后，您在控制台上创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 流作为流式传输源。发现过程读取流式传输源中的示例记录，并推断出具有两个列 (EVENT_TIME 和 TICKER) 的如下应用程序内部架构。



Filter by column name

ROWTIME	EVENT_TIME TIMESTAMP	TICKER VARCHAR(4)	PARTITION_KEY	SEQUENCE
2018-05-09 21:48:06.198	2018-05-09 14:48:05.169	INTC	partitionkey	4958385475
2018-05-09 21:48:06.198	2018-05-09 14:48:05.259	TBV	partitionkey	4958385475
2018-05-09 21:48:06.198	2018-05-09 14:48:05.348	INTC	partitionkey	4958385475
2018-05-09 21:48:06.198	2018-05-09 14:48:05.436	MSFT	partitionkey	4958385475

然后，将该应用程序代码与 SQL 函数结合使用，以多种方式转换 EVENT_TIME 时间戳字段。随后将结果数据插入另一个应用程序内部流，如下面的屏幕截图所示：



Filter by column name

ROWTIME	TICKER	EVENT_TIME	FIVE_MINUTES_BEFORE	EVE
2018-05-09 21:51:07.244	AAPL	2018-05-09 14:51:06.237	2018-05-09 14:46:06.237	1525
2018-05-09 21:51:07.244	INTC	2018-05-09 14:51:06.326	2018-05-09 14:46:06.326	1525
2018-05-09 21:51:07.244	AAPL	2018-05-09 14:51:06.414	2018-05-09 14:46:06.414	1525
2018-05-09 21:51:07.244	TBV	2018-05-09 14:51:06.503	2018-05-09 14:46:06.503	1525

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis Data Stream 并填充事件时间和股票代码记录，如下所示：

1. 登录 AWS Management Console 并打开 Kinesis 控制台，网址为 https://console.aws.amazon.com/kinesis。
2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流，然后创建带有一个分片的流。
4. 运行以下 Python 代码以便用示例数据填充流。此简单代码会不断将具有随机股票代号和当前时间戳的记录写入流中。

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Amazon Kinesis Data Analytics 应用程序

按如下方式创建应用程序：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择 创建应用程序，键入应用程序名称，然后选择 创建应用程序。
3. 在应用程序详细信息页面上，选择 连接流数据，以连接到源。
4. 在 连接到源 页面上，执行以下操作：

a. 选择在上一部分中创建的流。

b. 选择以创建 IAM 角色。

c. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。推断的架构有两列。

d. 选择 编辑架构。将 EVENT_TIME 列的 列类型 更改为 TIMESTAMP。

e. 选择 保存架构并更新流示例。在控制台保存架构后，选择 退出。

f. 选择 保存并继续。

5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择 是，启动应用程序。

6. 在 SQL 编辑器中编写应用程序代码并确认结果，如下所示：

a. 复制下面的应用程序代码并将其粘贴到编辑器中。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    TICKER VARCHAR(4),
    event_time TIMESTAMP,
    five_minutes_before TIMESTAMP,
    event_unix_timestamp BIGINT,
    event_timestamp_as_char VARCHAR(50),
    event_second INTEGER);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM
    TICKER,
    EVENT_TIME,
    EVENT_TIME - INTERVAL '5' MINUTE,
    UNIX_TIMESTAMP(EVENT_TIME),
    TIMESTAMP_TO_CHAR('yyyy-MM-dd hh:mm:ss', EVENT_TIME),
    EXTRACT(SECOND FROM EVENT_TIME)
FROM "SOURCE_SQL_STREAM_001"
```

b. 选择 保存并运行 SQL。在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

示例：转换多个数据类型

提取、转换和加载 (ETL) 应用程序的共同要求是处理流式传输源中的多种记录。您可以通过创建一个 Kinesis Data Analytics 应用程序来处理此类流式传输源。流程如下：

1. 首先，您将流式传输源映射到应用程序内部输入流，与所有其他 Kinesis Data Analytics 应用程序类似。
2. 然后，在应用程序代码中编写 SQL 语句，从应用程序内部输入流中检索特定类型的行。然后，将这些行插入单独的应用程序内部流。(您可以在应用程序代码中创建其他应用程序内部流)。

在本练习中，您具有一个可接收两种类型 (`Order` 和 `Trade`) 记录的流式传输源。它们分别表示库存订单和相应交易。每批订单可以有零笔或多笔交易。下面显示了每个类型的示例记录：

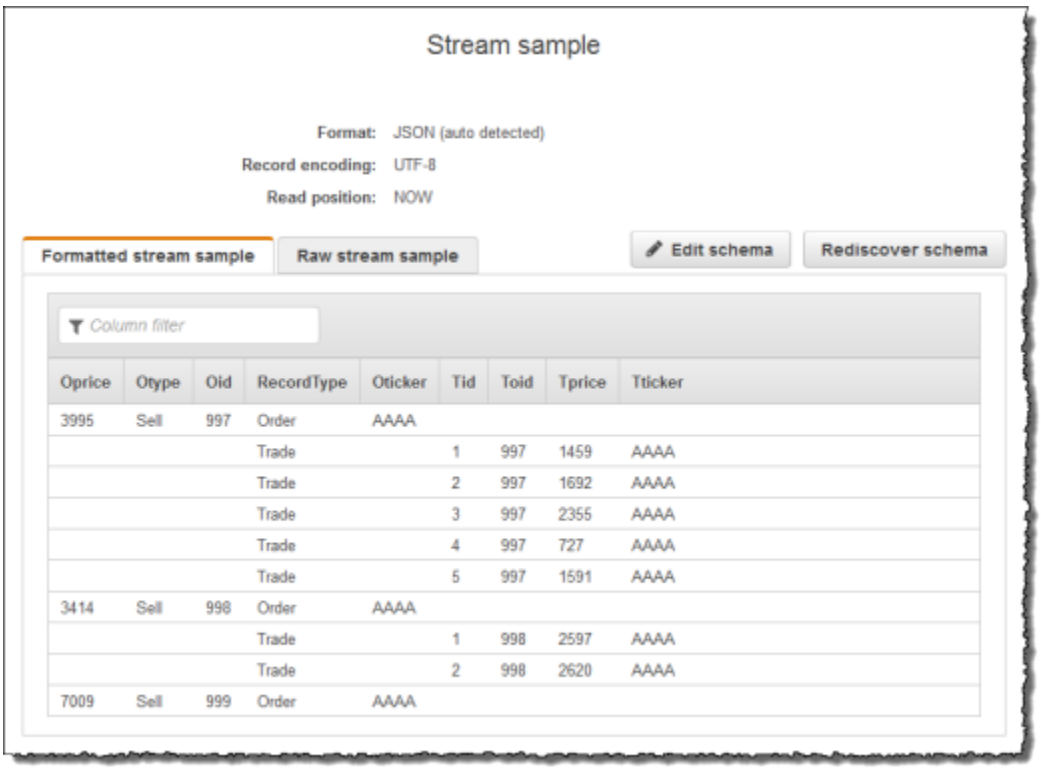
Order record

```
{"RecordType": "Order", "Oprice": 9047, "Otype": "Sell", "Oid": 3811, "Oticker":
 "AAAA"}
```

Trade record

```
{"RecordType": "Trade", "Tid": 1, "Toid": 3812, "Tprice": 2089, "Tticker": "BBBB"}
```

使用创建应用程序时 AWS Management Console，控制台会显示所创建的应用程序内输入流的以下推断架构。默认情况下，控制台将该应用程序内部流命名为 `SOURCE_SQL_STREAM_001`。



Oprice	Otype	Oid	RecordType	Oticker	Tid	Toid	Tprice	Tticker
3995	Sell	997	Order	AAAA				
			Trade		1	997	1459	AAAA
			Trade		2	997	1692	AAAA
			Trade		3	997	2355	AAAA
			Trade		4	997	727	AAAA
			Trade		5	997	1591	AAAA
3414	Sell	998	Order	AAAA				
			Trade		1	998	2597	AAAA
			Trade		2	998	2620	AAAA
7009	Sell	999	Order	AAAA				

在保存配置时，Amazon Kinesis Data Analytics 持续从流式传输源中读取数据，并在应用程序内部流中插入行。现在，您可以对应用程序内部流中的数据进行分析。

在本示例的应用程序代码中，首先创建两个额外的应用程序内部流：Order_Stream 和 Trade_Stream。然后，根据记录类型筛选 SOURCE_SQL_STREAM_001 流中的行，使用数据泵将这些行插入到新创建的流。有关此编码模式的信息，请参阅应用程序代码。

1. 将订单和交易行筛选到单独的应用程序内部流:

   a. 筛选 SOURCE_SQL_STREAM_001 中的订单记录，并将订单保存到 Order_Stream。

```
--Create Order_Stream.
CREATE OR REPLACE STREAM "Order_Stream"
           (
            order_id     integer,
            order_type   varchar(10),
            ticker       varchar(4),
            order_price  DOUBLE,
            record_type  varchar(10)
            );

CREATE OR REPLACE PUMP "Order_Pump" AS
```

```
   INSERT INTO "Order_Stream"
      SELECT STREAM oid, otype,oticker, oprice, recordtype
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  recordtype = 'Order';
```

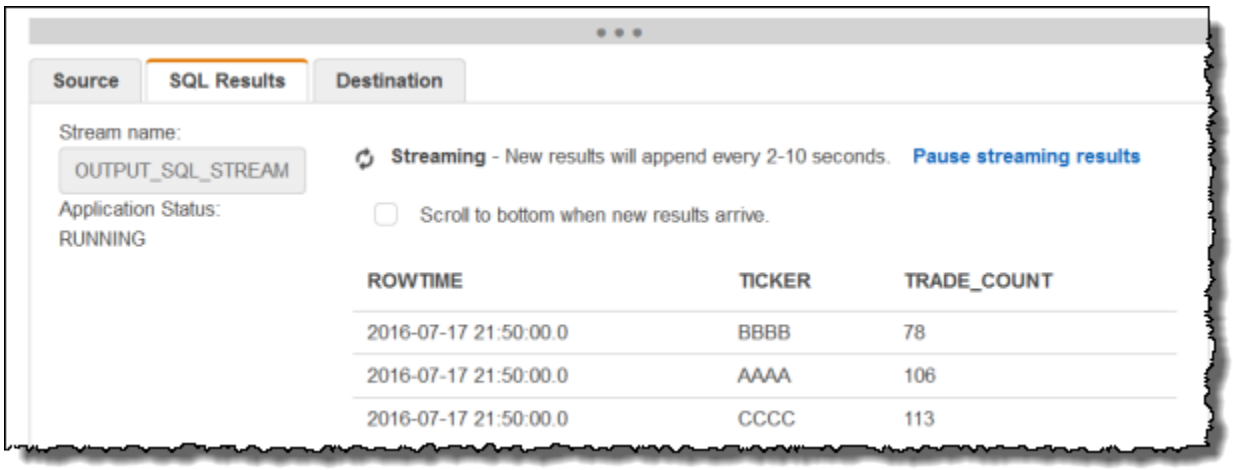
b. 筛选 SOURCE_SQL_STREAM_001 中的交易记录，并将订单保存到 Trade_Stream。

```
--Create Trade_Stream.
CREATE OR REPLACE STREAM "Trade_Stream"
           (trade_id     integer,
            order_id     integer,
            trade_price  DOUBLE,
            ticker       varchar(4),
            record_type  varchar(10)
            );

CREATE OR REPLACE PUMP "Trade_Pump" AS
   INSERT INTO "Trade_Stream"
      SELECT STREAM tid, toid, tprice, tticker, recordtype
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  recordtype = 'Trade';
```

2. 现在，您可以对这些流进行其他分析。在本示例中，您将按股票在时长一分钟的滚动窗口中计算交易数，并将结果保存到另一个流 DESTINATION_SQL_STREAM。

```
--do some analytics on the Trade_Stream and Order_Stream.
-- To see results in console you must write to OPUT_SQL_STREAM.

CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
            ticker  varchar(4),
            trade_count   integer
            );

CREATE OR REPLACE PUMP "Output_Pump" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM ticker, count(*) as trade_count
      FROM   "Trade_Stream"
      GROUP BY ticker,
               FLOOR("Trade_Stream".ROWTIME TO MINUTE);
```

此时将显示如下结果：



主题

- 步骤 1：准备数据
- 步骤 2：创建应用程序

下一个步骤

步骤 1：准备数据

步骤 1：准备数据

在此部分中，您创建一个 Kinesis 数据流，然后在该流上填充订单和交易记录。此式源将用于下一步创建的应用程序。

主题

- 步骤 1.1：创建流式传输源
- 步骤 1.2：填充流式传输源

步骤 1.1：创建流式传输源

可以使用控制台或 AWS CLI创建一个 Kinesis 数据流。本示例采用 `OrdersAndTradesStream` 作为流名称。

- 使用控制台 — 登录 AWS Management Console 并打开 Kinesis 控制台，网址为 https://console.aws.amazon.com/kinesis。选择 Data Streams，然后创建带有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。

- 使用 AWS CLI— 使用以下 Kinesis `create-stream` AWS CLI 命令创建直播：

```
$ aws kinesis create-stream \
--stream-name OrdersAndTradesStream \
--shard-count 1 \
--region us-east-1 \
--profile adminuser
```

步骤 1.2：填充流式传输源

运行以下 Python 脚本以便在 `OrdersAndTradesStream` 中填充示例记录。如果您使用其他名称创建了流，请相应更新 Python 代码。

1. 安装 Python 和 `pip`。

   有关安装 Python 的信息，请访问 Python 网站。

   您可以使用 pip 安装依赖项。有关安装 pip 的信息，请参阅 pip 网站上的安装。

2. 运行以下 Python 代码。代码中的 `put-record` 命令将 JSON 记录写入到流。

```
import json
import random
import boto3

STREAM_NAME = "OrdersAndTradesStream"
PARTITION_KEY = "partition_key"


def get_order(order_id, ticker):
    return {
        "RecordType": "Order",
        "Oid": order_id,
        "Oticker": ticker,
        "Oprice": random.randint(500, 10000),
        "Otype": "Sell",
    }


def get_trade(order_id, trade_id, ticker):
    return {
```

```
        "RecordType": "Trade",
        "Tid": trade_id,
        "Toid": order_id,
        "Tticker": ticker,
        "Tprice": random.randint(0, 3000),
    }


def generate(stream_name, kinesis_client):
    order_id = 1
    while True:
        ticker = random.choice(["AAAA", "BBBB", "CCCC"])
        order = get_order(order_id, ticker)
        print(order)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(order),
 PartitionKey=PARTITION_KEY
        )
        for trade_id in range(1, random.randint(0, 6)):
            trade = get_trade(order_id, trade_id, ticker)
            print(trade)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(trade),
                PartitionKey=PARTITION_KEY,
            )
        order_id += 1


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

下一个步骤

步骤 2：创建应用程序

步骤 2：创建应用程序

在此部分中，您创建一个 Kinesis Data Analytics 应用程序。然后，通过添加将您在前一部分中创建的流式传输源映射到应用程序内部输入流的输入配置，您可以更新应用程序。

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择创建应用程序。此示例使用应用程序名称 **ProcessMultipleRecordTypes**。
3. 在应用程序详细信息页面上，选择 连接流数据，以连接到源。
4. 在 连接到源 页面上，执行以下操作：
   a. 选择在步骤 1：准备数据中创建的流。
   b. 选择以创建 IAM 角色。
   c. 等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。
   d. 选择 保存并继续。
5. 在应用程序中心上，选择 Go to SQL editor。要启动应用程序，请在显示的对话框中选择 是，启动应用程序。
6. 在 SQL 编辑器中编写应用程序代码并确认结果：
   a. 复制下面的应用程序代码并将其粘贴到编辑器中。

```
--Create Order_Stream.
CREATE OR REPLACE STREAM "Order_Stream"
           (
            "order_id"     integer,
            "order_type"   varchar(10),
            "ticker"       varchar(4),
            "order_price"  DOUBLE,
            "record_type"  varchar(10)
            );

CREATE OR REPLACE PUMP "Order_Pump" AS
   INSERT INTO "Order_Stream"
      SELECT STREAM "Oid", "Otype","Oticker", "Oprice", "RecordType"
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  "RecordType" = 'Order';
--*****************************************
--Create Trade_Stream.
CREATE OR REPLACE STREAM "Trade_Stream"
           ("trade_id"     integer,
            "order_id"     integer,
            "trade_price"  DOUBLE,
            "ticker"       varchar(4),
            "record_type"  varchar(10)
            );
```

```
CREATE OR REPLACE PUMP "Trade_Pump" AS
   INSERT INTO "Trade_Stream"
      SELECT STREAM "Tid", "Toid", "Tprice", "Tticker", "RecordType"
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  "RecordType" = 'Trade';
--************************************************************
--do some analytics on the Trade_Stream and Order_Stream.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
            "ticker"  varchar(4),
            "trade_count"   integer
            );

CREATE OR REPLACE PUMP "Output_Pump" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM "ticker", count(*) as trade_count
      FROM   "Trade_Stream"
      GROUP BY "ticker",
                FLOOR("Trade_Stream".ROWTIME TO MINUTE);
```

b. 选择 保存并运行 SQL。选择 Real-time analytics (实时分析) 选项卡可查看应用程序已创建的所有应用程序内部流并验证数据。

下一个步骤

您可以将应用程序输出配置为将结果保存到外部目标，例如另一个 Kinesis 流或 Firehose 数据传输流。

示例：窗口和聚合

本部分提供使用窗口式查询和聚合查询的 Amazon Kinesis Data Analytics 应用程序示例。（有关更多信息，请参阅 窗口式查询。）每个示例提供分步说明和示例代码以设置 Kinesis Data Analytics 应用程序。

主题

- 示例：交错窗口
- 示例：使用 ROWTIME 的滚动窗口
- 示例：使用事件时间戳的滚动窗口

- 示例：检索最常出现的值 (TOP_K_ITEMS_TUMBLING)
- 示例：从查询中聚合部分结果

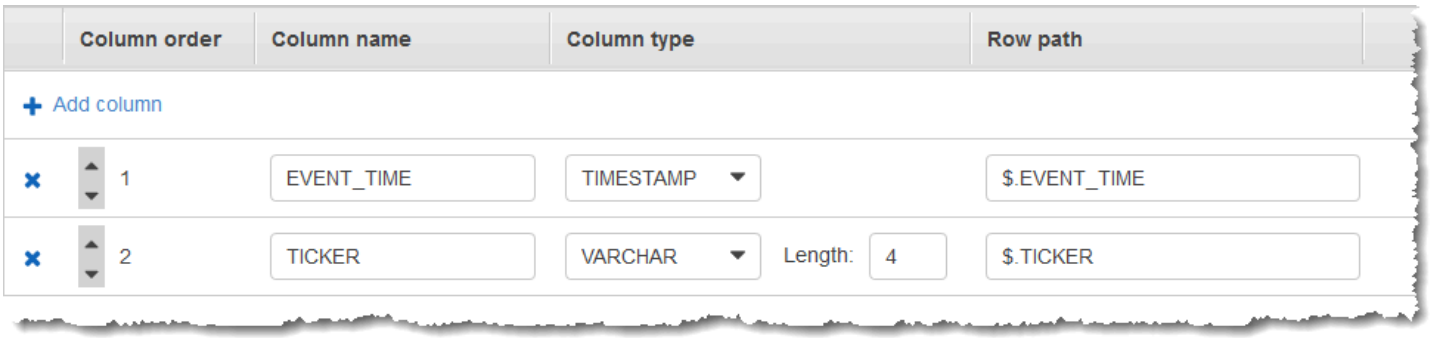
示例：交错窗口

当窗口化查询处理每个唯一分区键的单独窗口时，从具有匹配键的数据到达时开始，该窗口被称为交错窗口。有关详细信息，请参阅 交错窗口。此 Amazon Kinesis Data Analytics 示例使用 EVENT_TIME 和 TICKER 列创建交错窗口。源流包含具有相同 EVENT_TIME 和 TICKER 值的六个记录组成的组，这些值在一分钟时间内到达，但不一定具有相同的分钟值（例如 18:41:xx）。

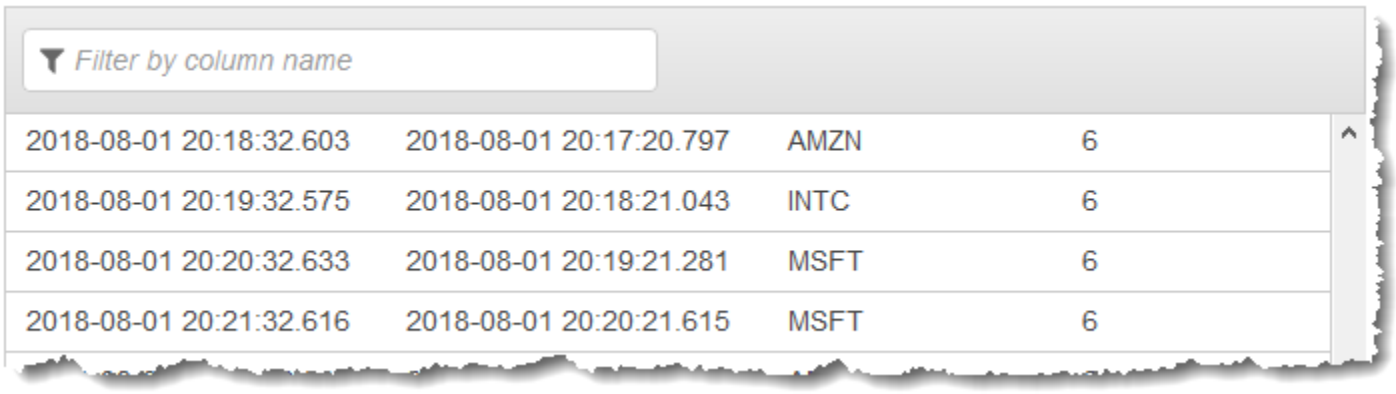
在本示例中，您在以下时间将以下记录写入到 Kinesis 数据流中。该脚本不会将时间写入流，但应用程序接收记录的时间将写入 ROWTIME 字段：

```
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:17:30
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:17:40
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:17:50
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:18:00
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:18:10
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:18:21
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:18:31
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:18:41
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:18:51
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:19:01
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:19:11
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:19:21
...
```

然后，您在 AWS Management Console 内创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 数据流作为流式传输源。发现过程读取流式传输源中的示例记录，并推断出具有两个列（EVENT_TIME 和 TICKER）的如下所示的应用程序内部架构。



您使用应用程序代码以及 COUNT 函数以创建数据的窗口式聚合。然后，将结果数据插入另一个应用程序内部流，如下面的屏幕截图所示：



在以下过程中，您创建一个 Kinesis Data Analytics 应用程序，它在基于 EVENT_TIME 和 TICKER 的交错窗口中聚合输入流中的值。

主题

- 步骤 1：创建 Kinesis 数据流
- 步骤 2：创建 Kinesis Data Analytics 应用程序

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis Data Stream 并填充记录，如下所示：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流，然后创建具有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
4. 要在生产环境中将记录写入到 Kinesis 数据流，我们建议您使用 Kinesis 创建器库或 Kinesis 数据流 API。为简单起见，此示例使用以下 Python 脚本以便生成记录。运行此代码以填充示例股票代码记录。这段简单代码在一分钟时间中连续地将一组六个记录与相同的随机 EVENT_TIME 和股票代码符号一起写入流中。让脚本保持运行，以便可以在后面的步骤中生成应用程序架构。

```
import datetime
import json
import random
```

```
import time
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    event_time = datetime.datetime.utcnow() - datetime.timedelta(seconds=10)
    return {
        "EVENT_TIME": event_time.isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        # Send six records, ten seconds apart, with the same event time and ticker
        for _ in range(6):
            print(data)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(data),
                PartitionKey="partitionkey",
            )
            time.sleep(10)


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Kinesis Data Analytics 应用程序

创建一个 Kinesis Data Analytics 应用程序，如下所示：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择 创建应用程序，键入应用程序名称，然后选择 创建应用程序。
3. 在应用程序详细信息页面上，选择 连接流数据，以连接到源。
4. 在 连接到源 页面上，执行以下操作：

a. 选择在上一部分中创建的流。

b. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。推断的架构有两列。

c. 选择 编辑架构。将 EVENT_TIME 列的 列类型 更改为 TIMESTAMP。

d. 选择 保存架构并更新流示例。在控制台保存架构后，选择 退出。

e. 选择 保存并继续。

5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择 是，启动应用程序。

6. 在 SQL 编辑器中编写应用程序代码并确认结果如下所示：

a. 复制下面的应用程序代码并将其粘贴到编辑器中。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    event_time TIMESTAMP,
    ticker_symbol    VARCHAR(4),
    ticker_count     INTEGER);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        EVENT_TIME,
        TICKER,
        COUNT(TICKER) AS ticker_count
    FROM "SOURCE_SQL_STREAM_001"
    WINDOWED BY STAGGER (
            PARTITION BY TICKER, EVENT_TIME RANGE INTERVAL '1' MINUTE);
```

b. 选择 保存并运行 SQL。

在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

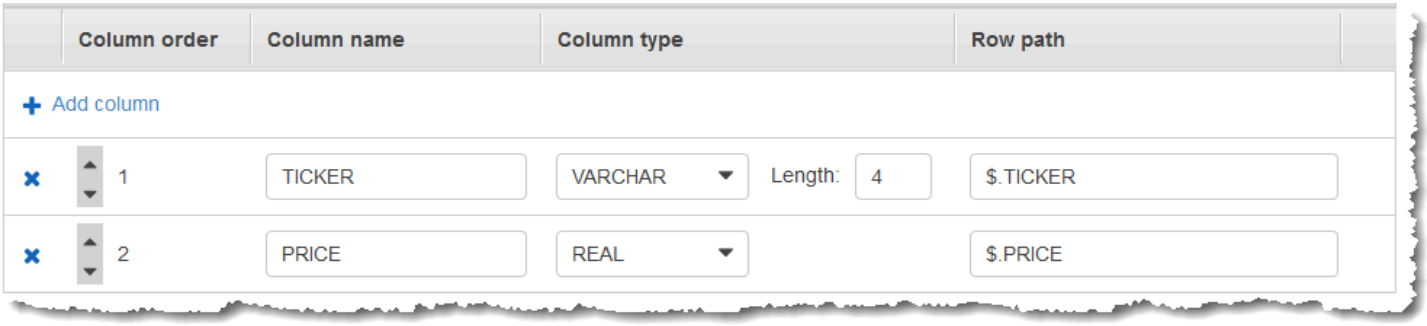
示例：使用 ROWTIME 的滚动窗口

当一个窗口式查询以非重叠方式处理每个窗口时，这样的窗口称为滚动窗口。有关详细信息，请参阅滚动窗口（使用 GROUP BY 组的聚合）。此 Amazon Kinesis Data Analytics 示例使用 ROWTIME 列创建滚动窗口。ROWTIME 列表示应用程序读取该记录的时间。

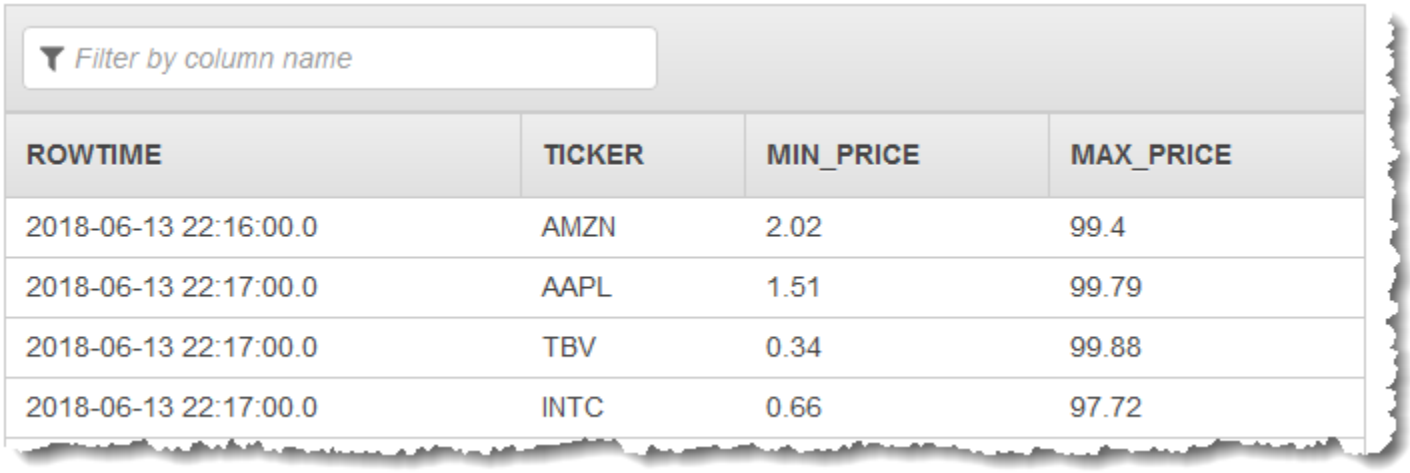
在本示例中，您将以下记录写入到 Kinesis 数据流中。

```
{"TICKER": "TBV", "PRICE": 33.11}
{"TICKER": "INTC", "PRICE": 62.04}
{"TICKER": "MSFT", "PRICE": 40.97}
{"TICKER": "AMZN", "PRICE": 27.9}
...
```

然后，您在 AWS Management Console 内创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 数据流作为流式传输源。发现过程读取流式传输源中的示例记录，并推断出具有两个列（TICKER 和 PRICE）的如下所示的应用程序内部架构。



您使用应用程序代码以及 MIN 和 MAX 函数以创建数据的窗口式聚合。然后，将结果数据插入另一个应用程序内部流，如下面的屏幕截图所示：



在以下过程中，您创建一个 Kinesis Data Analytics 应用程序，它在基于 ROWTIME 的滚动窗口中聚合输入流中的值。

主题

- 步骤 1：创建 Kinesis 数据流

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis Data Stream 并填充记录，如下所示：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流，然后创建具有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
4. 要在生产环境中将记录写入到 Kinesis 数据流，我们建议您使用 Kinesis 客户端库或 Kinesis 数据流 API。为简单起见，此示例使用以下 Python 脚本以便生成记录。运行此代码以填充示例股票代码记录。这段简单代码不断地将随机的股票代码记录写入到流中。让脚本保持运行，以便可以在后面的步骤中生成应用程序架构。

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
```

```
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Kinesis Data Analytics 应用程序

创建一个 Kinesis Data Analytics 应用程序，如下所示：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择 Create application (创建应用程序)，输入应用程序名称，然后选择 Create application (创建应用程序)。
3. 在应用程序详细信息页面上，选择 连接流数据，以连接到源。
4. 在 连接到源 页面上，执行以下操作：

   a. 选择在上一部分中创建的流。

   b. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。推断的架构有两列。

   c. 选择 保存架构并更新流示例。在控制台保存架构后，选择 退出。

   d. 选择 保存并继续。
5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择是，启动应用程序。
6. 在 SQL 编辑器中编写应用程序代码并确认结果如下所示：

   a. 复制下面的应用程序代码并将其粘贴到编辑器中。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (TICKER VARCHAR(4), MIN_PRICE
 REAL, MAX_PRICE REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM TICKER, MIN(PRICE), MAX(PRICE)
        FROM "SOURCE_SQL_STREAM_001"
        GROUP BY TICKER,
            STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

b. 选择 保存并运行 SQL。

在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

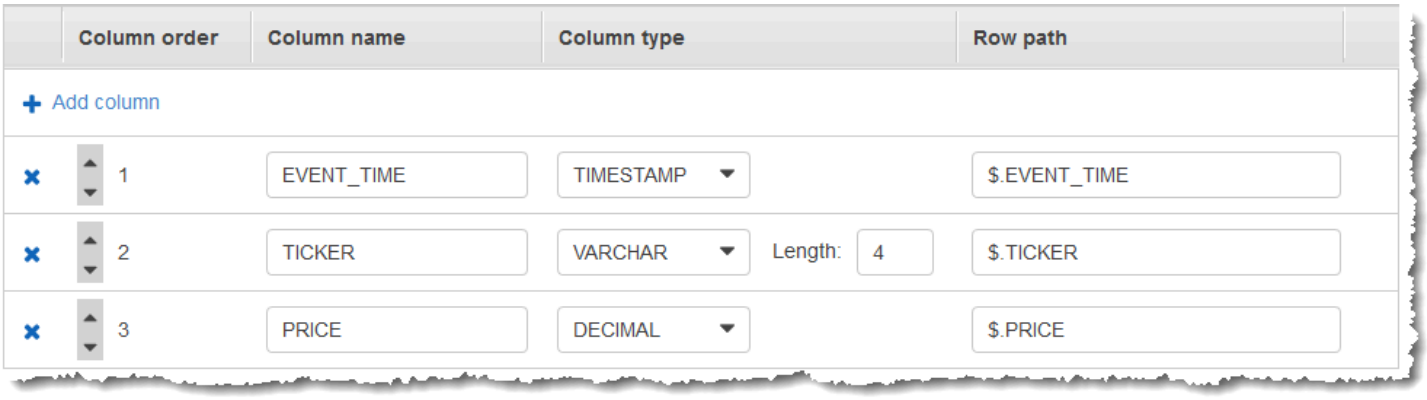
示例：使用事件时间戳的滚动窗口

当一个窗口式查询以非重叠方式处理每个窗口时，这样的窗口称为滚动窗口。有关详细信息，请参阅滚动窗口（使用 GROUP BY 组的聚合）。此 Amazon Kinesis Data Analytics 示例说明了一个使用事件时间戳的滚动窗口，这是一个用户创建的时间戳，它包含在流数据中。它使用这种方法而不是只使用 ROWTIME，这是 Kinesis Data Analytics 在应用程序收到记录时创建的时间戳。如果您想根据事件发生的时间而非应用程序收到此事件的时间创建一个聚合，则可以在流数据中使用事件时间戳。在本例中，ROWTIME 值每分钟触发一次此聚合，ROWTIME 和所包含事件时间都会聚合记录。

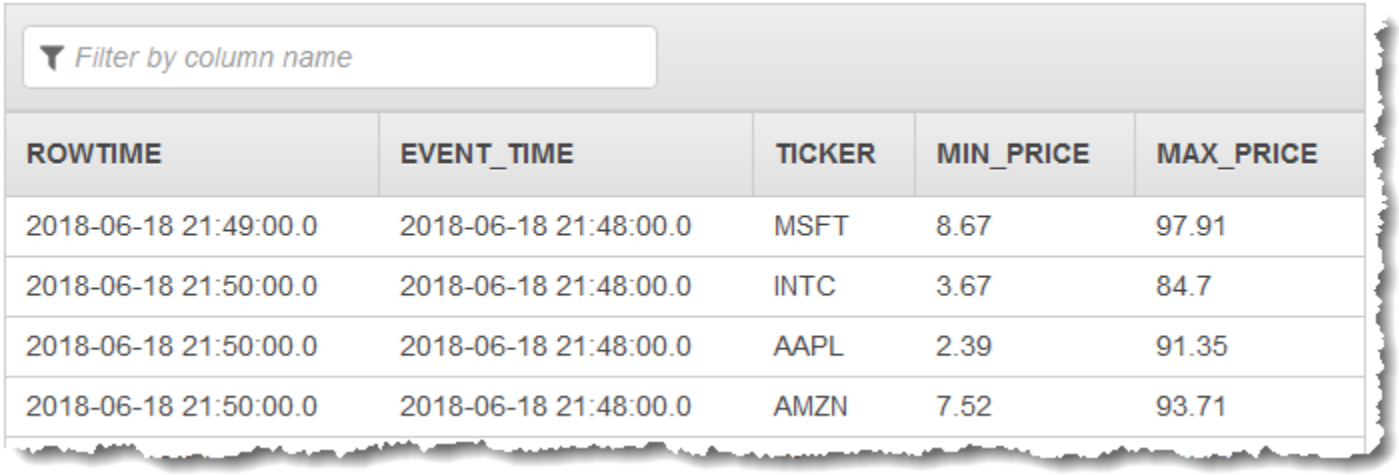
在本示例中，您将以下记录写入到 Amazon Kinesis 流中。EVENT_TIME 值在过去设置为 5 秒以模拟处理和传输滞后，而滞后可能导致在发生事件和将记录提取到 Kinesis Data Analytics 之间出现延迟。

```
{"EVENT_TIME": "2018-06-13T14:11:05.766191", "TICKER": "TBV", "PRICE": 43.65}
{"EVENT_TIME": "2018-06-13T14:11:05.848967", "TICKER": "AMZN", "PRICE": 35.61}
{"EVENT_TIME": "2018-06-13T14:11:05.931871", "TICKER": "MSFT", "PRICE": 73.48}
{"EVENT_TIME": "2018-06-13T14:11:06.014845", "TICKER": "AMZN", "PRICE": 18.64}
...
```

然后，您在 AWS Management Console 内创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 数据流作为流式传输源。发现过程读取流式传输源中的示例记录，并推断出具有三个列（EVENT_TIME、TICKER 和 PRICE）的如下所示的应用程序内部架构。



您使用应用程序代码以及 MIN 和 MAX 函数以创建数据的窗口式聚合。然后，将结果数据插入另一个应用程序内部流，如下面的屏幕截图所示：



Filter by column name

ROWTIME	EVENT_TIME	TICKER	MIN_PRICE	MAX_PRICE
2018-06-18 21:49:00.0	2018-06-18 21:48:00.0	MSFT	8.67	97.91
2018-06-18 21:50:00.0	2018-06-18 21:48:00.0	INTC	3.67	84.7
2018-06-18 21:50:00.0	2018-06-18 21:48:00.0	AAPL	2.39	91.35
2018-06-18 21:50:00.0	2018-06-18 21:48:00.0	AMZN	7.52	93.71

在以下过程中，您创建一个 Kinesis Data Analytics 应用程序，它在基于事件时间的滚动窗口中聚合输入流中的值。

主题

- 步骤 1：创建 Kinesis 数据流
- 步骤 2：创建 Kinesis Data Analytics 应用程序

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis Data Stream 并填充记录，如下所示：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流，然后创建具有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
4. 要在生产环境中将记录写入到 Kinesis 数据流，我们建议您使用 Kinesis 客户端库或 Kinesis 数据流 API。为简单起见，此示例使用以下 Python 脚本以便生成记录。运行此代码以填充示例股票代码记录。这段简单代码不断地将随机的股票代码记录写入到流中。让脚本保持运行，以便可以在后面的步骤中生成应用程序架构。

```
import datetime
import json
import random
import boto3
```

```
STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Kinesis Data Analytics 应用程序

创建一个 Kinesis Data Analytics 应用程序，如下所示：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择 创建应用程序，输入应用程序名称，然后选择 创建应用程序。
3. 在应用程序详细信息页面上，选择 连接流数据，以连接到源。
4. 在 连接到源 页面上，执行以下操作：

    a. 选择在上一部分中创建的流。

    b. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。推断的架构有三列。

    c. 选择 编辑架构。将 EVENT_TIME 列的 列类型 更改为 TIMESTAMP。

d. 选择 保存架构并更新流示例。在控制台保存架构后，选择 退出。

e. 选择 保存并继续。

5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择 是，启动应用程序。

6. 在 SQL 编辑器中编写应用程序代码并确认结果如下所示：

a. 复制下面的应用程序代码并将其粘贴到编辑器中。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (EVENT_TIME timestamp, TICKER
 VARCHAR(4), min_price REAL, max_price REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM STEP("SOURCE_SQL_STREAM_001".EVENT_TIME BY INTERVAL '60'
 SECOND),
        TICKER,
         MIN(PRICE) AS MIN_PRICE,
         MAX(PRICE) AS MAX_PRICE
    FROM    "SOURCE_SQL_STREAM_001"
    GROUP BY TICKER,
             STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND),
             STEP("SOURCE_SQL_STREAM_001".EVENT_TIME BY INTERVAL '60' SECOND);
```

b. 选择 保存并运行 SQL。

在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

示例：检索最常出现的值 (TOP_K_ITEMS_TUMBLING)

此 Amazon Kinesis Data Analytics 示例说明了如何使用 TOP_K_ITEMS_TUMBLING 函数在滚动窗口中检索最常出现的值。有关更多信息，请参阅Amazon Managed Service for Apache Flink SQL 参考中的TOP_K_ITEMS_TUMBLING 函数。

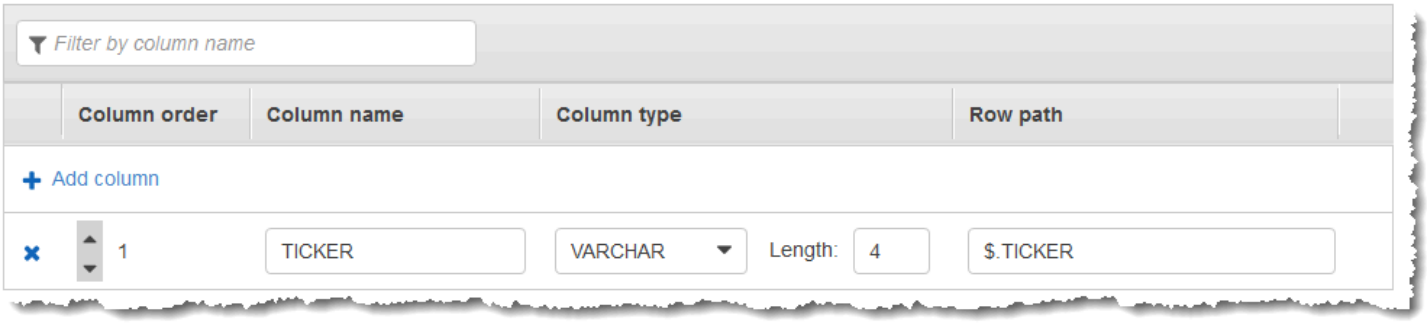
在对数万或数十万个密钥进行聚合时，如果您希望降低资源占用，则 TOP_K_ITEMS_TUMBLING 函数很有用。该函数生成的结果与使用 GROUP BY 和 ORDER BY 子句进行聚合一样。

在本示例中，您将以下记录写入到 Amazon Kinesis 数据流中：

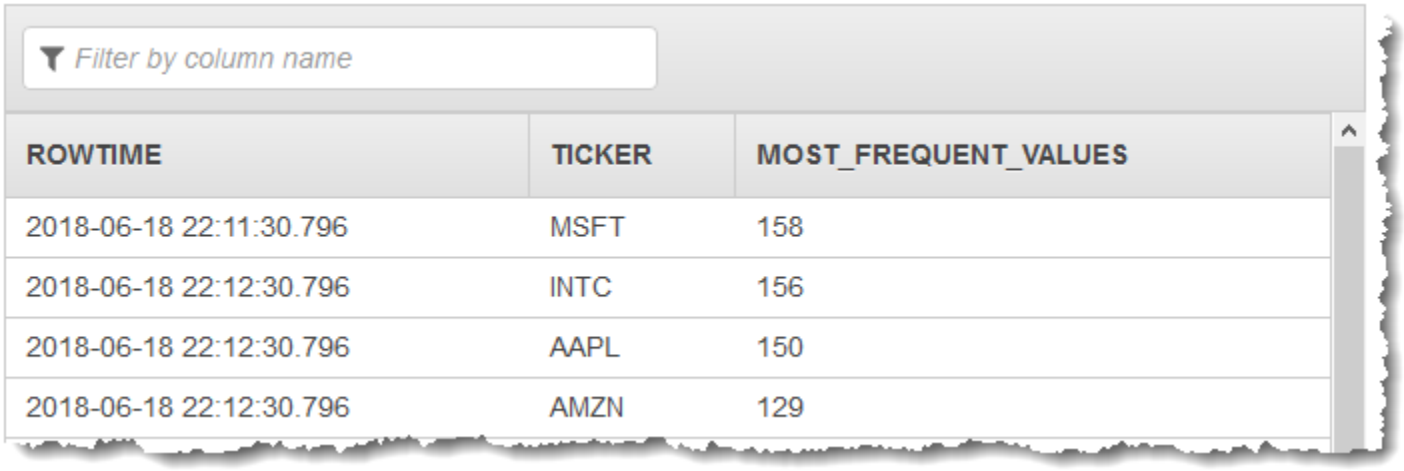
```
{"TICKER": "TBV"}
{"TICKER": "INTC"}
```

```
{"TICKER": "MSFT"}
{"TICKER": "AMZN"}
...
```

然后，您在 AWS Management Console 内创建一个 Kinesis Data Analytics 应用程序，并将 Kinesis 数据流作为流式传输源。发现过程读取流式传输源上的示例记录，并推断出具有一个列 (TICKER) 的应用程序内部架构，如下所示：



您使用应用程序代码以及 TOP_K_VALUES_TUMBLING 函数以创建数据的窗口式聚合。然后，将结果数据插入另一个应用程序内部流，如下面的屏幕截图所示：



在以下过程中，您创建一个 Kinesis Data Analytics 应用程序，它在输入流中检索最常出现的值。

主题

- 步骤 1：创建 Kinesis 数据流
- 步骤 2：创建 Kinesis Data Analytics 应用程序

步骤 1：创建 Kinesis 数据流

创建一个 Amazon Kinesis Data Stream 并填充记录，如下所示：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 数据流。
3. 选择 创建 Kinesis 流，然后创建具有一个分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
4. 要在生产环境中将记录写入到 Kinesis 数据流，我们建议您使用 Kinesis 客户端库或 Kinesis 数据流 API。为简单起见，此示例使用以下 Python 脚本以便生成记录。运行此代码以填充示例股票代码记录。这段简单代码不断地将随机的股票代码记录写入到流中。让脚本保持运行，以便可以在后面的步骤中生成应用程序架构。

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )
```

```
if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

步骤 2：创建 Kinesis Data Analytics 应用程序

创建一个 Kinesis Data Analytics 应用程序，如下所示：

1. 打开适用于 Apache Flink 的托管服务控制台，网址为 https://console.aws.amazon.com/kinesisanalytics。
2. 选择 创建应用程序，键入应用程序名称，然后选择 创建应用程序。
3. 在应用程序详细信息页面上，选择 连接流数据，以连接到源。
4. 在 连接到源 页面上，执行以下操作：

   a. 选择在上一部分中创建的流。

   b. 选择 发现架构。等待控制台显示推断的架构和为创建的应用程序内部流推断架构所使用的示例记录。推断的架构包含一列。

   c. 选择 保存架构并更新流示例。在控制台保存架构后，选择 退出。

   d. 选择 保存并继续。
5. 在应用程序详细信息页面上，选择 转到 SQL编辑器。要启动应用程序，请在显示的对话框中选择 是，启动应用程序。
6. 在 SQL 编辑器中编写应用程序代码并确认结果如下所示：

   a. 复制下面的应用程序代码并将其粘贴到编辑器中：

```
CREATE OR REPLACE STREAM DESTINATION_SQL_STREAM (
  "TICKER" VARCHAR(4),
  "MOST_FREQUENT_VALUES" BIGINT
);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM *
        FROM TABLE (TOP_K_ITEMS_TUMBLING(
            CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"),
            'TICKER',         -- name of column in single quotes
            5,                      -- number of the most frequently occurring
 values
```

```
            60                          -- tumbling window size in seconds
            )
        );
```

b. 选择 保存并运行 SQL。

在 实时分析 选项卡上，可以查看应用程序已创建的所有应用程序内部流并验证数据。

示例：从查询中聚合部分结果

如果 Amazon Kinesis 数据流包含的记录具有与提取时间不完全匹配的事件时间，在滚动窗口中选择的结果将包含在窗口中到达但未必发生的记录。在这种情况下，滚动窗口只包含您需要的部分结果集。您可以通过多种方法来纠正这一问题：

- 仅使用滚动窗口，并使用 upsert 通过数据库或数据仓库在后处理中聚合部分结果。这种方法在处理应用程序时很有效。它为聚合运算符（sum、min、max 等）无限期地处理后期数据。这种方法的缺点是，您必须在数据库层开发和维护额外的应用程序逻辑。
- 使用滚动和滑动窗口，这会在早期生成部分结果，还会继续在滑动窗口期间生成完整的结果。此方法使用 overwrite 操作而非 upsert 操作来处理新近数据，这样就不需要在数据库层添加任何其他应用程序逻辑。这种方法的缺点是，它使用更多 Kinesis 处理单元 (KPU)，并且仍生成两个结果，这可能不适用于某些使用案例。

有关滚动和滑动窗口的更多信息，请参阅窗口式查询。

在以下过程中，滚动窗口聚合会生成两个部分结果（发送到 CALC_COUNT_SQL_STREAM 应用程序内部流），它们必须合并以生成最终结果。然后，应用程序生成第二个聚合（发送到 DESTINATION_SQL_STREAM 应用程序内部流），它合并这两个部分结果。

创建使用事件时间聚合部分结果的应用程序

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 Data Analytics (数据分析)。按照 适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 入门 教程中所述，创建一个 Kinesis Data Analytics 应用程序。
3. 在 SQL 编辑器中，将应用程序代码替换为以下内容：

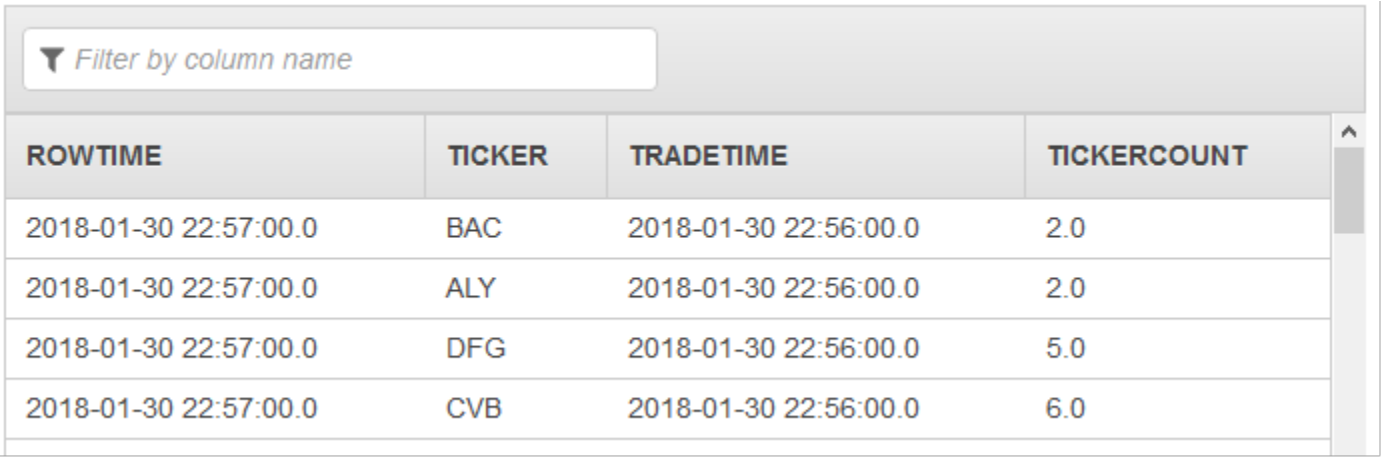
```
CREATE OR REPLACE STREAM "CALC_COUNT_SQL_STREAM"
    (TICKER      VARCHAR(4),
```

```
    TRADETIME   TIMESTAMP,
    TICKERCOUNT       DOUBLE);

CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
    (TICKER      VARCHAR(4),
    TRADETIME   TIMESTAMP,
    TICKERCOUNT       DOUBLE);

CREATE PUMP "CALC_COUNT_SQL_PUMP_001" AS
    INSERT INTO "CALC_COUNT_SQL_STREAM" ("TICKER","TRADETIME", "TICKERCOUNT")
    SELECT STREAM
        "TICKER_SYMBOL",
        STEP("SOURCE_SQL_STREAM_001"."ROWTIME" BY INTERVAL '1' MINUTE) as
 "TradeTime",
        COUNT(*) AS "TickerCount"
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY
        STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE),
        STEP("SOURCE_SQL_STREAM_001"."APPROXIMATE_ARRIVAL_TIME" BY INTERVAL '1'
 MINUTE),
        TICKER_SYMBOL;

CREATE PUMP "AGGREGATED_SQL_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM" ("TICKER","TRADETIME", "TICKERCOUNT")
    SELECT STREAM
        "TICKER",
        "TRADETIME",
        SUM("TICKERCOUNT") OVER W1 AS "TICKERCOUNT"
    FROM "CALC_COUNT_SQL_STREAM"
    WINDOW W1 AS (PARTITION BY "TRADETIME" RANGE INTERVAL '10' MINUTE PRECEDING);
```

应用程序代码中的 SELECT 语句将在 SOURCE_SQL_STREAM_001 中筛选出显示股票价格更改大于 1% 的行，并使用数据泵将这些行插入另一个应用程序内部流 CHANGE_STREAM。

4. 选择 保存并运行 SQL。

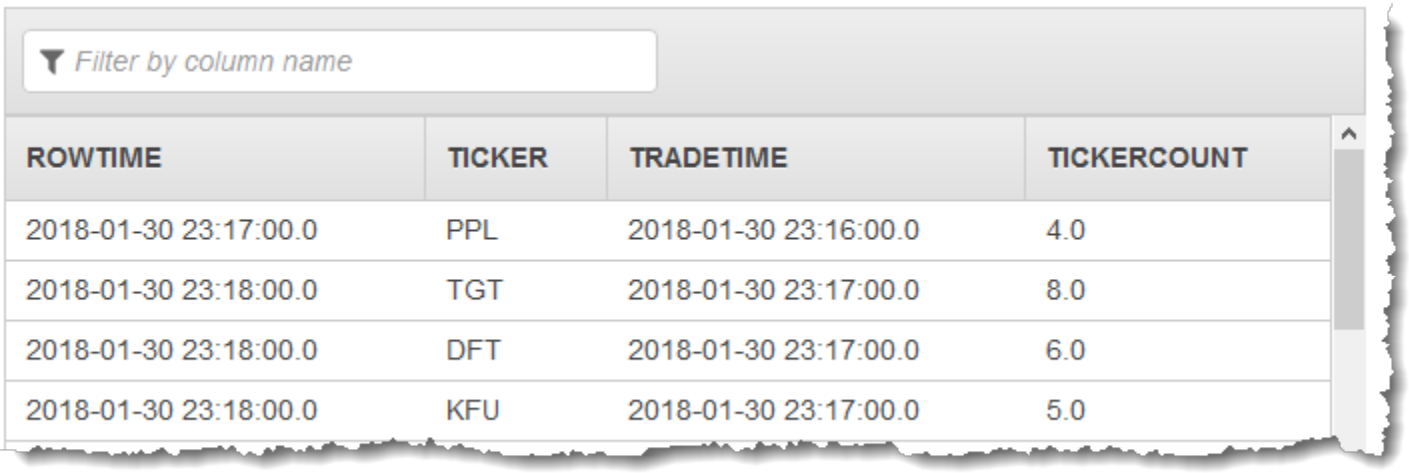
第一个数据泵将流输出到与以下内容类似的 CALC_COUNT_SQL_STREAM。请注意，结果集不完整：



Filter by column name

ROWTIME	TICKER	TRADETIME	TICKERCOUNT
2018-01-30 22:57:00.0	BAC	2018-01-30 22:56:00.0	2.0
2018-01-30 22:57:00.0	ALY	2018-01-30 22:56:00.0	2.0
2018-01-30 22:57:00.0	DFG	2018-01-30 22:56:00.0	5.0
2018-01-30 22:57:00.0	CVB	2018-01-30 22:56:00.0	6.0

然后，第二个泵将流输出到 DESTINATION_SQL_STREAM，其中包含完整的结果集：



Filter by column name

ROWTIME	TICKER	TRADETIME	TICKERCOUNT
2018-01-30 23:17:00.0	PPL	2018-01-30 23:16:00.0	4.0
2018-01-30 23:18:00.0	TGT	2018-01-30 23:17:00.0	8.0
2018-01-30 23:18:00.0	DFT	2018-01-30 23:17:00.0	6.0
2018-01-30 23:18:00.0	KFU	2018-01-30 23:17:00.0	5.0

示例：联接

此部分提供了使用联接查询的 数据分析应用程序示例。每个示例提供分步说明以设置和测试 数据分析应用程序。

主题

- 示例：在 应用程序中添加引用数据

示例：在 应用程序中添加引用数据

在本练习中，您在现有 数据分析应用程序中添加引用数据。有关引用数据的信息，请参阅以下主题：

- 适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理

- 配置应用程序输入

在本练习中，您将引用数据添加到在 Kinesis Data Analytics 入门练习中创建的应用程序。引用数据提供每个股票代号对应的公司名称；例如：

```
Ticker, Company
AMZN,Amazon
ASD, SomeCompanyA
MMB, SomeCompanyB
WAS,  SomeCompanyC
```

首先，请完成入门练习中的步骤，以创建一个启动应用程序。然后，执行以下步骤进行设置，并将引用数据添加到应用程序：

1. 准备数据
   - 将上述参考数据作为对象存储在 Amazon Simple Storage Service (Amazon S3) 中
   - 创建 IAM 角色，Kinesis Data Analytics 可以担任该角色来代表您读取 Amazon S3 对象。
2. 将引用数据源添加到您的应用程序。

   读取 对象，并创建应用程序内部引用表，您可以在应用程序代码中查询该表。
3. 测试代码。

   在应用程序代码中，将编写一个联接查询来联接应用程序内部流和应用程序内部引用表，从而获取每个股票代码的对应公司名称。

主题

- 步骤 1：准备
- 步骤 2：将引用数据源添加到应用程序配置中
- 步骤 3：测试：查询应用程序内部引用表

步骤 1：准备

在此部分中，您将示例引用数据作为对象存储在 存储桶中。您还将创建 IAM 角色， 可担任该角色来代表您读取对象。

将引用数据存储为 Amazon S3 对象。

在这一步中，将示例引用数据存储为 Amazon S3 对象。

1. 打开文本编辑器，添加以下数据，然后将文件另存为 `TickerReference.csv`。

```
Ticker, Company
AMZN,Amazon
ASD, SomeCompanyA
MMB, SomeCompanyB
WAS,  SomeCompanyC
```

2. 将 `TickerReference.csv` 文件上传到 S3 存储桶。有关说明，请参阅 Amazon Simple Storage Service 用户指南中的上传对象。

创建 IAM 角色

接下来，创建一个 Kinesis Data Analytics 可以代入的 IAM 角色并读取 Amazon S3 对象。

1. 在 AWS Identity and Access Management (IAM) 中，创建名为 **`KinesisAnalytics-ReadS3Object`** 的 IAM 角色。要创建该角色，请按照 IAM 用户指南中的为 Amazon Web Service 创建角色（AWS Management Console）中的说明操作。

   在 &IAM; 控制台上，指定以下项：

   - 在选择角色类型中，选择 AWS Lambda。在创建角色后，您将更改信任策略以允许（而非）代入该角色。
   - 不要在 Attach Policy 页面上附加任何策略。
2. 更新 &IAM; 角色策略：

   a. 在 &IAM; 控制台上，选择您创建的角色。

   b. 在信任关系选项卡上，更新信任策略以便为 Kinesis Data Analytics 授予权限以代入该角色。下面显示了信任策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "kinesisanalytics.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

c. 在 权限 选项卡上，附加名为 AmazonS3ReadOnlyAccess 的 Amazon 托管策略。这会为该角色授予权限以读取 对象。下面显示了此策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

步骤 2：将引用数据源添加到应用程序配置中

在该步骤中，将引用数据源添加到应用程序配置中。要开始操作，需要以下信息：

- S3 存储桶名称和对象键名称
- &IAM; 角色的 Amazon 资源名称 (ARN)

1. 在应用程序主页面中，选择 Connect reference data (连接引用数据)。
2. 在连接引用数据来源页面上，选择包含引用数据对象的 Amazon S3 存储桶，并输入对象的键名称。
3. 在应用程序内部引用表名称处输入 **`CompanyName`**。

4. 在 Access to chosen resources (访问所选的资源) 部分中，选择 Choose from IAM roles that Kinesis Analytics can assume (从 Kinesis Analytics 可代入的 IAM 角色中选择)，然后选择您在上一部分中创建的 KinesisAnalytics-ReadS3Object IAM 角色。
5. 选择 发现架构。控制台检测到引用数据中的两列。
6. 选择 Save and close。

步骤 3：测试：查询应用程序内部引用表

现在，您可以查询应用程序内部引用表 CompanyName。您可以联接股票价格数据和引用表以使用引用信息扩充应用程序。结果显示公司名称。

1. 用以下内容替换您的应用程序代码。查询会联接应用程序内部输入流和应用程序内部引用表。应用程序代码将结果写入到另一应用程序内部流 DESTINATION_SQL_STREAM。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 "Company" varchar(20), sector VARCHAR(12), change DOUBLE, price DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
   SELECT STREAM ticker_symbol, "c"."Company", sector, change, price
   FROM "SOURCE_SQL_STREAM_001" LEFT JOIN "CompanyName" as "c"
   ON "SOURCE_SQL_STREAM_001".ticker_symbol = "c"."Ticker";
```

2. 验证 SQLResults 选项卡中是否会显示应用程序输出。确保某些行显示公司名称 (示例引用数据不包含所有公司名称)。

示例：机器学习

此部分提供了使用机器学习查询的 Amazon Kinesis Data Analytics 应用程序示例。机器学习查询对数据执行复杂的分析，同时依靠流中数据的历史记录以查找异常模式。这些示例提供了分步说明以设置和测试 Kinesis Data Analytics 应用程序。

主题

- 示例：在流中检测数据异常情况 (RANDOM_CUT_FOREST 函数)
- 示例：检测数据异常和获取说明 (RANDOM_CUT_FOREST_WITH_EXPLANATION 函数)
- 示例：检测流上的热点 (HOTSPOTS 函数)

示例：在流中检测数据异常情况 (RANDOM_CUT_FOREST 函数)

Amazon Kinesis Data Analytics 提供了一个函数 (RANDOM_CUT_FOREST)，它可以根据数值列中的值将异常分数分配给每个记录。有关更多信息，Amazon Managed Service for Apache Flink SQL 参考中的RANDOM_CUT_FOREST 函数。

在本练习中，您将编写应用程序代码以将异常分数分配给应用程序的流式传输源中的记录。要设置应用程序，请执行以下操作：

1. 设置流式源 - 您设置 Kinesis 数据流并编写示例 heartRate 数据，如下所示：

   ```
   {"heartRate": 60, "rateType":"NORMAL"}
   ...
   {"heartRate": 180, "rateType":"HIGH"}
   ```

   此过程提供用于填充流的 Python 脚本。heartRate 值将随机生成，99% 的记录具有的 heartRate 值介于 60 和 100 之间，仅 1% 的记录具有的 heartRate 值介于 150 和 200 之间。因此，heartRate 值介于 150 和 200 之间的记录是异常情况。

2. 配置输入 - 通过使用控制台，您创建 Kinesis Data Analytics 应用程序，并通过将流式源映射到应用程序内部流 (SOURCE_SQL_STREAM_001) 来配置应用程序输入。在应用程序启动时，Kinesis Data Analytics 持续读取流式传输源，并将记录插入到应用程序内部流中。

3. 指定应用程序代码 - 示例使用以下应用程序代码：

   ```
   --Creates a temporary stream.
   CREATE OR REPLACE STREAM "TEMP_STREAM" (
            "heartRate"        INTEGER,
            "rateType"         varchar(20),
            "ANOMALY_SCORE"    DOUBLE);

   --Creates another stream for application output.
   CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
            "heartRate"        INTEGER,
            "rateType"         varchar(20),
            "ANOMALY_SCORE"    DOUBLE);

   -- Compute an anomaly score for each record in the input stream
   -- using Random Cut Forest
   CREATE OR REPLACE PUMP "STREAM_PUMP" AS
      INSERT INTO "TEMP_STREAM"
         SELECT STREAM "heartRate", "rateType", ANOMALY_SCORE
   ```

```
      FROM TABLE(RANDOM_CUT_FOREST(
              CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001")));

-- Sort records by descending anomaly score, insert into output stream
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM * FROM "TEMP_STREAM"
      ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), ANOMALY_SCORE DESC;
```

此代码读取 SOURCE_SQL_STREAM_001 中的行，分配异常分数，并将结果行写入另一个应用程序内部流 (TEMP_STREAM)。随后，应用程序代码将对 TEMP_STREAM 中的记录进行排序，并将结果保存到另一个应用程序内部流 (DESTINATION_SQL_STREAM)。您使用数据泵将流插入到应用程序内部流。有关更多信息，请参阅 应用程序内部流和数据泵。

4. 配置输出 - 您配置应用程序输出以将 DESTINATION_SQL_STREAM 中的数据保存到外部目标 (另一个 Kinesis 数据流)。查看分配给每条记录的异常分数并确定哪个分数指示应用程序外部的异常情况 (您需要收到这些异常情况的警报)。您可以使用 AWS Lambda 函数处理这些异常分数并配置警报。

此练习使用美国东部 (弗吉尼亚州北部) (us-east-1) 来创建这些流和您的应用程序。如果您使用任何其他区域，则必须相应地更新代码。

主题

- 步骤 1：准备
- 步骤 2：创建应用程序
- 步骤 3：配置应用程序输出
- 步骤 4：验证输出

下一个步骤

步骤 1：准备

步骤 1：准备

在为本练习创建 Amazon Kinesis Data Analytics 应用程序之前，您必须创建两个 Kinesis 数据流。将一个流配置为应用程序的流式传输源，并将另一个流配置为目标（Kinesis Data Analytics 在其中永久保存应用程序输出）。

主题

步骤 1.1：创建输入和输出数据流

在此部分中，您创建两个 Kinesis 流：ExampleInputStream 和 ExampleOutputStream。您可以使用 AWS Management Console或 AWS CLI 创建这些流。

- 要使用 控制台

  1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
  2. 选择创建数据流。创建带有一个名为 ExampleInputStream 的分片的流。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的创建流。
  3. 重复上一步骤以创建带有一个名为 ExampleOutputStream 的分片的流。

- 使用 AWS CLI

  1. 使用下面的 Kinesis create-stream AWS CLI 命令创建第一个流 (ExampleInputStream)。

     ```
     $ aws kinesis create-stream \
     --stream-name ExampleInputStream \
     --shard-count 1 \
     --region us-east-1 \
     --profile adminuser
     ```

  2. 运行同一命令，同时将流名称更改为 ExampleOutputStream。此命令创建应用程序用来写入输出的第二个流。

步骤 1.2：将示例记录写入输入流

在此步骤中，您运行 Python 代码以持续生成示例记录，并将这些记录写入 ExampleInputStream 流。

```
{"heartRate": 60, "rateType":"NORMAL"}
...
{"heartRate": 180, "rateType":"HIGH"}
```

1. 安装 Python 和 pip。

有关安装 Python 的信息，请访问 Python 网站。

您可以使用 pip 安装依赖项。有关安装 pip 的信息，请参阅 pip 网站上的安装。

2. 运行以下 Python 代码。代码中的 `put-record` 命令将 JSON 记录写入到流。

```
from enum import Enum
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


class RateType(Enum):
    normal = "NORMAL"
    high = "HIGH"


def get_heart_rate(rate_type):
    if rate_type == RateType.normal:
        rate = random.randint(60, 100)
    elif rate_type == RateType.high:
        rate = random.randint(150, 200)
    else:
        raise TypeError
    return {"heartRate": rate, "rateType": rate_type.value}


def generate(stream_name, kinesis_client, output=True):
    while True:
        rnd = random.random()
        rate_type = RateType.high if rnd < 0.01 else RateType.normal
        heart_rate = get_heart_rate(rate_type)
        if output:
            print(heart_rate)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(heart_rate),
            PartitionKey="partitionkey",
        )
```

```
if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

下一个步骤

步骤 2：创建应用程序

步骤 2：创建应用程序

在此部分中，您创建一个 Amazon Kinesis Data Analytics 应用程序，如下所示：

- 配置应用程序输入以将在 the section called “步骤 1：准备” 中创建的 Kinesis 数据流作为流式传输源。
- 在控制台上使用 Anomaly Detection (异常检测) 模板。

创建应用程序

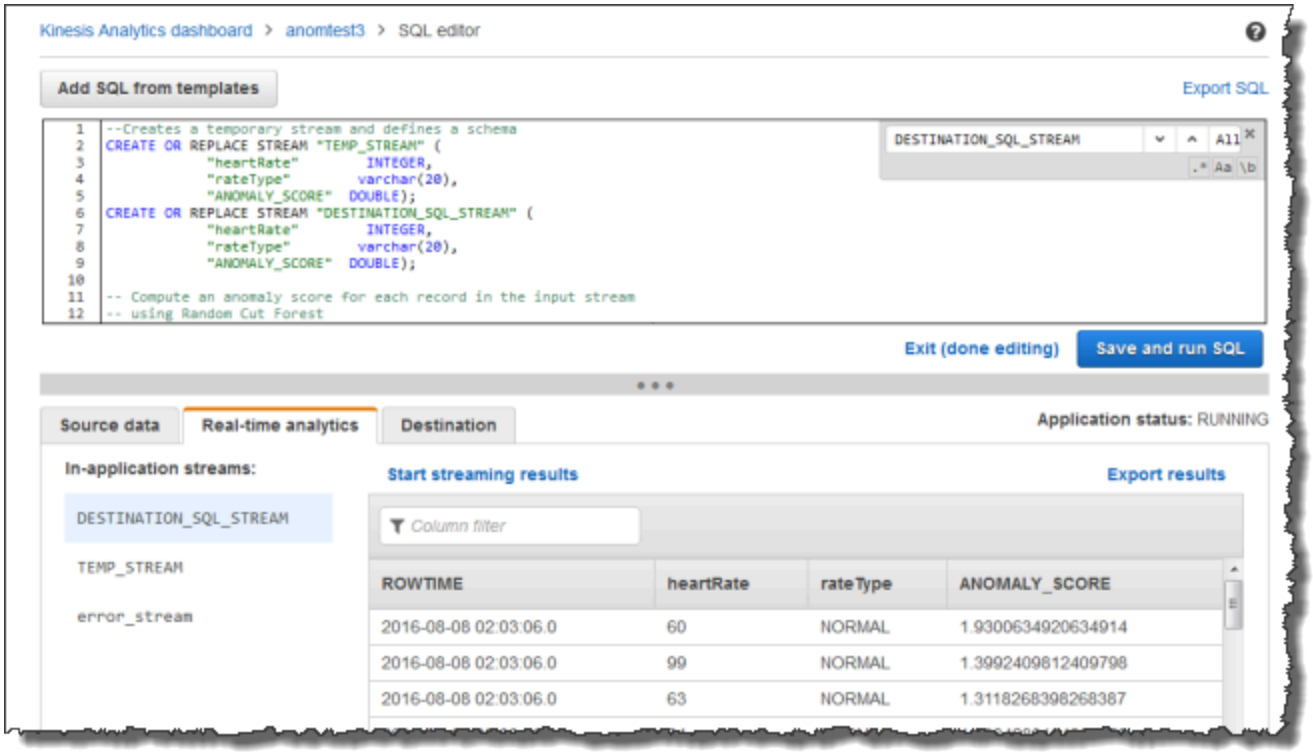
1. 按照 Kinesis Data Analytics 入门练习中的步骤 1、2 和 3（请参阅 步骤 3.1：创建应用程序）。
   - 在源配置中，执行以下操作：
     - 指定您在上一部分中创建的流式传输源。
     - 在控制台推断架构后，编辑架构并将 `heartRate` 列类型设置为 `INTEGER`。

       大多数心率值是正常的，发现过程最有可能将 `TINYINT` 类型分配给此列。但有小部分值显示了高心率。如果这些较高的值不适合 `TINYINT` 类型，则 Kinesis Data Analytics 会将这些行发送到错误流。将数据类型更新为 `INTEGER`，以便能适合所有生成的心率数据。
   - 在控制台上使用 Anomaly Detection (异常检测) 模板。随后，您更新模板代码以提供适当的列名称。
2. 通过提供列名称来更新应用程序代码。下面显示了生成的应用程序代码 (将此代码粘贴到 SQL 编辑器中)：

```
--Creates a temporary stream.
CREATE OR REPLACE STREAM "TEMP_STREAM" (
         "heartRate"        INTEGER,
         "rateType"         varchar(20),
         "ANOMALY_SCORE"    DOUBLE);
```

```
--Creates another stream for application output.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
         "heartRate"        INTEGER,
         "rateType"         varchar(20),
         "ANOMALY_SCORE"    DOUBLE);

-- Compute an anomaly score for each record in the input stream
-- using Random Cut Forest
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "TEMP_STREAM"
      SELECT STREAM "heartRate", "rateType", ANOMALY_SCORE
      FROM TABLE(RANDOM_CUT_FOREST(
              CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001")));

-- Sort records by descending anomaly score, insert into output stream
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM * FROM "TEMP_STREAM"
      ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), ANOMALY_SCORE DESC;
```

3. 运行 SQL 代码并在 Kinesis Data Analytics 控制台中检查结果：



下一个步骤

步骤 3：配置应用程序输出

步骤 3：配置应用程序输出

完成the section called “步骤 2：创建应用程序”后，您已拥有用于从流式传输源读取心率数据并为每条记录分配一个异常分数的应用程序代码。

您现在可以将应用程序内部流中的应用程序结果发送到外部目标，这是另一个 Kinesis 数据流 (`OutputStreamTestingAnomalyScores`)。您可以分析异常分数并确定哪种心率是异常的。然后，您可以进一步扩展此应用程序以生成警报。

执行以下步骤可配置应用程序输出：

1. 打开 Amazon Kinesis Data Analytics 控制台。在 SQL 编辑器中，在应用程序控制面板中选择 Destination 或 Add a destination。
2. 在 Connect to destination (连接到目标) 页面中，选择您在上一部分中创建的 `OutputStreamTestingAnomalyScores` 流。

   现在，您具有一个外部目标，Amazon Kinesis Data Analytics 将应用程序写入到应用程序内部流 `DESTINATION_SQL_STREAM` 的任何记录永久保存到该目标中。
3. 您可以选择配置 AWS Lambda 以监控 `OutputStreamTestingAnomalyScores` 流并发送警报。有关说明，请参阅 使用 Lambda 函数预处理数据。如果未设置警报，您可以查看 Kinesis Data Analytics 写入到外部目标（Kinesis 数据流 `OutputStreamTestingAnomalyScores`）的记录，如 步骤 4：验证输出 中所述。

下一个步骤

步骤 4：验证输出

步骤 4：验证输出

在the section called “步骤 3：配置应用程序输出”中配置应用程序输出后，使用以下 AWS CLI 命令读取应用程序在目标流中写入的记录。

1. 运行 `get-shard-iterator` 命令以获取指向输出流中的数据的指针。

```
aws kinesis get-shard-iterator \
--shard-id shardId-000000000000 \
```

```
--shard-iterator-type TRIM_HORIZON \
--stream-name OutputStreamTestingAnomalyScores \
--region us-east-1 \
--profile adminuser
```

您将获得带分片迭代器值的响应，如以下示例响应中所示：

```
  {
      "ShardIterator":
      "shard-iterator-value"   }
```

复制该分片迭代器值。

2. 运行 `get-records`命令 AWS CLI。

```
aws kinesis get-records \
--shard-iterator shared-iterator-value \
--region us-east-1 \
--profile adminuser
```

此命令将返回一页记录和另一个分片迭代器，您可在后续 `get-records` 命令中使用该迭代器来提取下一组记录。

示例：检测数据异常和获取说明 (RANDOM_CUT_FOREST_WITH_EXPLANATION 函数)

Amazon Kinesis Data Analytics 提供了 `RANDOM_CUT_FOREST_WITH_EXPLANATION` 函数，该函数根据数值列中的值为每个记录分配一个异常分数。该函数还能提供异常说明。有关更多信息，请参阅 Amazon Managed Service for Apache Flink SQL 参考中的 RANDOM_CUT_FOREST_WITH_EXPLANATION。

在本练习中，您将编写应用程序代码，以获取应用程序流式传输源中记录的异常分数。以及获取每个异常的说明。

主题

- 步骤 1：准备数据
- 步骤 2：创建分析应用程序
- 步骤 3：检查结果

第一步

步骤 1：准备数据

步骤 1：准备数据

在为此示例创建 Amazon Kinesis Data Analytics 应用程序前，需要先创建一个 Kinesis 数据流，以作为应用程序的流式传输源。另外，您还需运行 Python 代码来将模拟血压数据写入流中。

主题

- 步骤 1.1：创建 Kinesis 数据流
- 步骤 1.2：将示例记录写入输入流

步骤 1.1：创建 Kinesis 数据流

在此部分中，您创建一个名为 `ExampleInputStream` 的 Kinesis 数据流。您可以使用 AWS Management Console或 AWS CLI 创建该数据流。

- 使用控制台：
  1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
  2. 在导航窗格中，选择 数据流。然后选择创建 Kinesis 流。
  3. 对于名称，请键入 **`ExampleInputStream`**。对于分片数，请键入 **`1`**。
- 或者，要使用 AWS CLI 创建数据流，请运行以下命令：

```
$ aws kinesis create-stream --stream-name ExampleInputStream --shard-count 1
```

步骤 1.2：将示例记录写入输入流

在此步骤中，您运行 Python 代码以不断生成示例记录并将其写入您创建的数据流中。

1. 安装 Python 和 pip。

   有关安装 Python 的信息，请参阅 Python。

   您可以使用 pip 安装依赖项。有关安装 pip 的信息，请参阅 pip 文档中的安装。

2. 运行以下 Python 代码。可以将区域改为您要用于此示例的区域。代码中的 put-record 命令将 JSON 记录写入到流。

```
from enum import Enum
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


class PressureType(Enum):
    low = "LOW"
    normal = "NORMAL"
    high = "HIGH"


def get_blood_pressure(pressure_type):
    pressure = {"BloodPressureLevel": pressure_type.value}
    if pressure_type == PressureType.low:
        pressure["Systolic"] = random.randint(50, 80)
        pressure["Diastolic"] = random.randint(30, 50)
    elif pressure_type == PressureType.normal:
        pressure["Systolic"] = random.randint(90, 120)
        pressure["Diastolic"] = random.randint(60, 80)
    elif pressure_type == PressureType.high:
        pressure["Systolic"] = random.randint(130, 200)
        pressure["Diastolic"] = random.randint(90, 150)
    else:
        raise TypeError
    return pressure


def generate(stream_name, kinesis_client):
    while True:
        rnd = random.random()
        pressure_type = (
            PressureType.low
            if rnd < 0.005
            else PressureType.high
            if rnd > 0.995
            else PressureType.normal
```

```
        )
        blood_pressure = get_blood_pressure(pressure_type)
        print(blood_pressure)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(blood_pressure),
            PartitionKey="partitionkey",
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

下一个步骤

步骤 2：创建分析应用程序

步骤 2：创建分析应用程序

在本节中，您将创建一个 Amazon Kinesis Data Analytics 应用程序，然后将其配置为使用在 the section called “步骤 1：准备数据” 中作为流式传输源创建的 Kinesis 数据流。然后，运行使用 RANDOM_CUT_FOREST_WITH_EXPLANATION 函数的应用程序代码。

创建应用程序

1. 打开 Kinesis 控制台，网址为：https://console.aws.amazon.com/kinesis。
2. 在导航窗格中选择 Data Analytics (数据分析)，然后选择创建应用程序。
3. 提供应用程序名称和描述 (可选)，并选择 Create application。
4. 选择 Connect streaming data (连接流数据)，然后从列表中选择 ExampleInputStream。
5. 选择 Discover schema，并确保 Systolic 和 Diastolic 显示为 INTEGER 列。如果二者为另一种类型，则选择 Edit schema，并将 INTEGER 类型分配给二者。
6. 在 Real time analytics 下，选择 Go to SQL editor。出现提示时，选择运行您的应用程序。
7. 将以下代码粘贴到 SQL 编辑器中，然后选择 Save and run SQL。

```
--Creates a temporary stream.
CREATE OR REPLACE STREAM "TEMP_STREAM" (
         "Systolic"                  INTEGER,
         "Diastolic"                 INTEGER,
```

```
         "BloodPressureLevel"        varchar(20),
         "ANOMALY_SCORE"             DOUBLE,
         "ANOMALY_EXPLANATION"       varchar(512));

--Creates another stream for application output.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
         "Systolic"                  INTEGER,
         "Diastolic"                 INTEGER,
         "BloodPressureLevel"        varchar(20),
         "ANOMALY_SCORE"             DOUBLE,
         "ANOMALY_EXPLANATION"       varchar(512));

-- Compute an anomaly score with explanation for each record in the input stream
-- using RANDOM_CUT_FOREST_WITH_EXPLANATION
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "TEMP_STREAM"
      SELECT STREAM "Systolic", "Diastolic", "BloodPressureLevel", ANOMALY_SCORE,
 ANOMALY_EXPLANATION
      FROM TABLE(RANDOM_CUT_FOREST_WITH_EXPLANATION(
              CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"), 100, 256,
 100000, 1, true));

-- Sort records by descending anomaly score, insert into output stream
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM * FROM "TEMP_STREAM"
      ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), ANOMALY_SCORE DESC;
```

下一个步骤

步骤 3：检查结果

步骤 3：检查结果

当您对此示例运行 SQL 代码时，首先会看到异常分数等于零的行。这种情况发生在初始学习阶段。然后，您会得到类似如下的结果：

```
ROWTIME SYSTOLIC DIASTOLIC BLOODPRESSURELEVEL ANOMALY_SCORE ANOMALY_EXPLANATION
27:49.0 101      66        NORMAL             0.711460417   {"Systolic":
{"DIRECTION":"LOW","STRENGTH":"0.0922","ATTRIBUTION_SCORE":"0.3792"},"Diastolic":
{"DIRECTION":"HIGH","STRENGTH":"0.0210","ATTRIBUTION_SCORE":"0.3323"}}
```

```
27:50.0 144      123       HIGH              3.855851061   {"Systolic":
{"DIRECTION":"HIGH","STRENGTH":"0.8567","ATTRIBUTION_SCORE":"1.7447"},"Diastolic":
{"DIRECTION":"HIGH","STRENGTH":"7.0982","ATTRIBUTION_SCORE":"2.1111"}}
27:50.0 113      69        NORMAL            0.740069409   {"Systolic":
{"DIRECTION":"LOW","STRENGTH":"0.0549","ATTRIBUTION_SCORE":"0.3750"},"Diastolic":
{"DIRECTION":"LOW","STRENGTH":"0.0394","ATTRIBUTION_SCORE":"0.3650"}}
27:50.0 105      64        NORMAL            0.739644157   {"Systolic":
{"DIRECTION":"HIGH","STRENGTH":"0.0245","ATTRIBUTION_SCORE":"0.3667"},"Diastolic":
{"DIRECTION":"LOW","STRENGTH":"0.0524","ATTRIBUTION_SCORE":"0.3729"}}
27:50.0 100      65        NORMAL            0.736993425   {"Systolic":
{"DIRECTION":"HIGH","STRENGTH":"0.0203","ATTRIBUTION_SCORE":"0.3516"},"Diastolic":
{"DIRECTION":"LOW","STRENGTH":"0.0454","ATTRIBUTION_SCORE":"0.3854"}}
27:50.0 108      69        NORMAL            0.733767202   {"Systolic":
{"DIRECTION":"LOW","STRENGTH":"0.0974","ATTRIBUTION_SCORE":"0.3961"},"Diastolic":
{"DIRECTION":"LOW","STRENGTH":"0.0189","ATTRIBUTION_SCORE":"0.3377"}}
```

- RANDOM_CUT_FOREST_WITH_EXPLANATION 函数中的算法看到 Systolic (收缩压) 和 Diastolic (舒张压) 列为数字，于是将它们作为输入。
- BloodPressureLevel 列包含文本数据，因此不会被算法所考虑。该列只是一个可视化助手，用来帮助您快速发现本例中的正常、高、低血压水平。
- 在 ANOMALY_SCORE 列中，分数越高的记录越异常。此示例结果集中的第二个记录最异常，异常分数为 3.855851061。
- 要了解算法所考虑的每个数字列在多大程度上造成异常评分，请参阅 ATTRIBUTION_SCORE 列中名为 ANOMALY_SCORE 的 JSON 字段。对于该示例结果集中的第二行，Systolic 和 Diastolic 列造成异常的比例为 1.7447:2.1111。换句话说，异常分数原因的 45% 归咎于收缩压值，55% 归咎于舒张压值。
- 要确定此示例中第二行所代表的点的方向是否异常，请参阅名为 DIRECTION 的 JSON 字段。在本例中，舒张压和收缩压值均标记为 HIGH。要确定这些方向正确的置信度，请参阅名为 STRENGTH 的 JSON 字段。在此示例中，算法更加确信舒张值太高。事实上，舒张压读数的正常值通常为 60–80，而 123 远高于预期值。

示例：检测流上的热点 (HOTSPOTS 函数)

Amazon Kinesis Data Analytics 提供了 HOTSPOTS 函数，它可以查找并返回有关数据中的相对密集的区域的信息。有关更多信息，请参阅 Amazon Managed Service for Apache Flink SQL 参考中的 HOTSPOTS。

在本练习中，您将编写应用程序代码以查找应用程序的流式传输源上的热点。要设置应用程序，请执行以下步骤：

1. 设置流式传输源 - 您设置 Kinesis 流并编写示例坐标数据，如下所示：

   ```
   {"x": 7.921782426109737, "y": 8.746265312709893, "is_hot": "N"}
   {"x": 0.722248626528026, "y": 4.648868803193405, "is_hot": "Y"}
   ```

   本示例提供了用于填充流的 Python 脚本。x 和 y 值是随机生成的，一些记录集中在特定位置周围。

   如果脚本有意生成值作为热点的一部分，is_hot 字段将作为指示器提供。这可以帮助您评估热点检测函数是否正常运行。

2. 创建应用程序 – 使用 AWS Management Console，您随后创建一个 Kinesis Data Analytics 应用程序。通过将流式传输源映射到应用程序内部流 (SOURCE_SQL_STREAM_001) 来配置应用程序输入。在应用程序启动时，Kinesis Data Analytics 持续读取流式传输源，并将记录插入到应用程序内部流中。

   在本练习中，您将为应用程序使用以下代码：

   ```
   CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
       "x" DOUBLE,
       "y" DOUBLE,
       "is_hot" VARCHAR(4),
       HOTSPOTS_RESULT VARCHAR(10000)
   );
   CREATE OR REPLACE PUMP "STREAM_PUMP" AS
       INSERT INTO "DESTINATION_SQL_STREAM"
       SELECT "x", "y", "is_hot", "HOTSPOTS_RESULT"
       FROM TABLE (
           HOTSPOTS(
               CURSOR(SELECT STREAM "x", "y", "is_hot" FROM "SOURCE_SQL_STREAM_001"),
               1000,
               0.2,
               17)
       );
   ```

此代码读取 SOURCE_SQL_STREAM_001 中的行，分析它是否有大量热点，并将生成的数据写入到另一个应用程序内部流 (DESTINATION_SQL_STREAM)。您使用数据泵将流插入到应用程序内部流。有关更多信息，请参阅 应用程序内部流和数据泵。

3. 配置输出 - 您配置应用程序输出以将应用程序中的数据发送到外部目标 (另一个 Kinesis 数据流)。查看热点分数并确定哪些分数表明出现了热点 (并且您需要收到警报)。您可以使用 AWS Lambda 函数进一步处理热点信息并配置警报。
4. 验证输出 - 示例包含一个 JavaScript 应用程序，该应用程序从输出流读取数据并以图形方式显示它，因此您可以实时查看应用程序生成的热点。

本练习使用美国西部（俄勒冈州）(us-west-2) 区域创建这些流和您的应用程序。如果您使用任何其他区域，请相应地更新代码。

主题

- 步骤 1：创建输入和输出流
- 步骤 2：创建 Kinesis Data Analytics 应用程序
- 步骤 3：配置应用程序输出
- 步骤 4：验证应用程序输出

步骤 1：创建输入和输出流

在为热点示例创建 Amazon Kinesis Data Analytics 应用程序之前，您必须创建两个 Kinesis 数据流。将一个流配置为应用程序的流式传输源，并将另一个流配置为目标（Kinesis Data Analytics 在其中永久保存应用程序输出）。

主题

- 步骤 1.1：创建 Kinesis 数据流
- 步骤 1.2：将示例记录写入输入流

步骤 1.1：创建 Kinesis 数据流

在此部分中，您创建两个 Kinesis 数据流：ExampleInputStream 和 ExampleOutputStream。

使用控制台或 AWS CLI 创建这些数据流。

- 使用控制台创建数据流：

1. 登录到 AWS Management Console，然后通过以下网址打开 Kinesis 控制台：https://console.aws.amazon.com/kinesisvideo/home。
2. 在导航窗格中，选择 数据流。
3. 选择创建 Kinesis 流，然后创建带有一个名为 ExampleInputStream 的分片的流。
4. 重复上一步骤以创建带有一个名为 ExampleOutputStream 的分片的流。

- 要使用 AWS CLI 创建数据流，请执行以下操作：
  - 使用以下 Kinesis create-stream AWS CLI 命令创建流（ExampleInputStream 和 ExampleOutputStream）。要创建另一个流 (应用程序将用于写入输出)，请运行同一命令以将流名称更改为 ExampleOutputStream。

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser

$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

步骤 1.2：将示例记录写入输入流

在此步骤中，您运行 Python 代码以持续生成示例记录并将其写入 ExampleInputStream 流。

```
{"x": 7.921782426109737, "y": 8.746265312709893, "is_hot": "N"}
{"x": 0.722248626580026, "y": 4.648868803193405, "is_hot": "Y"}
```

1. 安装 Python 和 pip。

   有关安装 Python 的信息，请访问 Python 网站。

   您可以使用 pip 安装依赖项。有关安装 pip 的信息，请参阅 pip 网站上的安装。
2. 运行以下 Python 代码。此代码将执行以下操作：

- 在 (X, Y) 平面上的某个位置生成潜在热点。
- 为每个热点生成一系列点 (1000 个)。这些点中有 20% 集中在热点周围。其余的点在整个空间内随机生成。
- `put-record` 命令将 JSON 记录写入到流。

> **Important**
>
> 请勿将此文件上传到 Web 服务器，因为它包含您的 AWS 凭证。

```
import json
from pprint import pprint
import random
import time
import boto3

STREAM_NAME = "ExampleInputStream"


def get_hotspot(field, spot_size):
    hotspot = {
        "left": field["left"] + random.random() * (field["width"] - spot_size),
        "width": spot_size,
        "top": field["top"] + random.random() * (field["height"] - spot_size),
        "height": spot_size,
    }
    return hotspot


def get_record(field, hotspot, hotspot_weight):
    rectangle = hotspot if random.random() < hotspot_weight else field
    point = {
        "x": rectangle["left"] + random.random() * rectangle["width"],
        "y": rectangle["top"] + random.random() * rectangle["height"],
        "is_hot": "Y" if rectangle is hotspot else "N",
    }
    return {"Data": json.dumps(point), "PartitionKey": "partition_key"}
```

```
def generate(
    stream_name, field, hotspot_size, hotspot_weight, batch_size, kinesis_client
):
    """
    Generates points used as input to a hotspot detection algorithm.
    With probability hotspot_weight (20%), a point is drawn from the hotspot;
    otherwise, it is drawn from the base field. The location of the hotspot
    changes for every 1000 points generated.
    """
    points_generated = 0
    hotspot = None
    while True:
        if points_generated % 1000 == 0:
            hotspot = get_hotspot(field, hotspot_size)
        records = [
            get_record(field, hotspot, hotspot_weight) for _ in range(batch_size)
        ]
        points_generated += len(records)
        pprint(records)
        kinesis_client.put_records(StreamName=stream_name, Records=records)

        time.sleep(0.1)


if __name__ == "__main__":
    generate(
        stream_name=STREAM_NAME,
        field={"left": 0, "width": 10, "top": 0, "height": 10},
        hotspot_size=1,
        hotspot_weight=0.2,
        batch_size=10,
        kinesis_client=boto3.client("kinesis"),
    )
```

下一个步骤

[步骤 2：创建 Kinesis Data Analytics 应用程序]

步骤 2：创建 Kinesis Data Analytics 应用程序

在该热点示例的此部分中，您创建一个 Kinesis Data Analytics 应用程序，如下所示：

- 配置应用程序输入以将您在步骤 1 中创建的 Kinesis 数据流用作流式传输源。
- 在 AWS Management Console中使用提供的应用程序代码。

创建应用程序

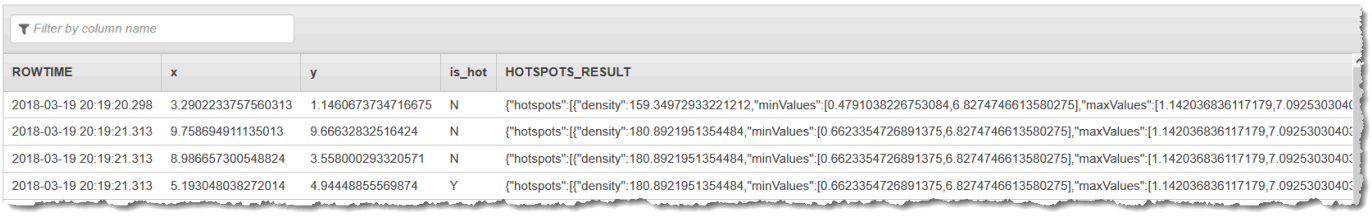
1. 按照入门练习中的步骤 1、2 和 3（请参阅 步骤 3.1：创建应用程序）创建 Kinesis Data Analytics 应用程序。

   在源配置中，执行以下操作：

   - 指定您在the section called “步骤 1：创建流”中创建的流式传输源。
   - 在控制台推断架构后编辑架构。确保 x 和 yDOUBLE 列类型设置为 ，并确保 IS_HOT 列类型设置为 VARCHAR。
2. 使用以下应用程序代码 (您可以将此代码粘贴到 SQL 编辑器中)：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "x" DOUBLE,
    "y" DOUBLE,
    "is_hot" VARCHAR(4),
    HOTSPOTS_RESULT VARCHAR(10000)
);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT "x", "y", "is_hot", "HOTSPOTS_RESULT"
    FROM TABLE (
        HOTSPOTS(
            CURSOR(SELECT STREAM "x", "y", "is_hot" FROM "SOURCE_SQL_STREAM_001"),
            1000,
            0.2,
            17)
    );
```

3. 运行 SQL 代码并审查结果。



Filter by column name

ROWTIME	x	y	is_hot	HOTSPOTS_RESULT
2018-03-19 20:19:20.298	3.2902233757560313	1.1460673734716675	N	{"hotspots":[{"density":159.34972933221212,"minValues":[0.4791038226753084,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040
2018-03-19 20:19:21.313	9.758694911135013	9.66632832516424	N	{"hotspots":[{"density":180.8921951354484,"minValues":[0.6623354726891375,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040
2018-03-19 20:19:21.313	8.986657300548824	3.558000293320571	N	{"hotspots":[{"density":180.8921951354484,"minValues":[0.6623354726891375,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040
2018-03-19 20:19:21.313	5.193048038272014	4.94448855569874	Y	{"hotspots":[{"density":180.8921951354484,"minValues":[0.6623354726891375,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040

下一个步骤

步骤 3：配置应用程序输出

步骤 3：配置应用程序输出

在热点示例的此部分中，您使用 Amazon Kinesis Data Analytics 应用程序代码查找流式传输源中的大量热点并将热分数分配给每个热点。

您现在可以将应用程序内部流中的应用程序结果发送到外部目标，这是另一个 Kinesis 数据流 (ExampleOutputStream)。然后，您可以分析热点分数并为热点热确定合适的阈值。您可以进一步扩展此应用程序以生成警报。

配置应用程序输出

1. 打开 Kinesis Data Analytics 控制台，网址为：https://console.aws.amazon.com/kinesisanalytics。
2. 在 SQL 编辑器中，在应用程序控制面板中选择 Destination 或 Add a destination。
3. 在 Add a destination (添加目标) 页面上，选择 Select from your streams (从流中选择)。然后选择在上一部分中创建的 ExampleOutputStream 流。

   现在，您具有一个外部目标，Amazon Kinesis Data Analytics 将应用程序写入到应用程序内部流 DESTINATION_SQL_STREAM 的任何记录永久保存到该目标中。
4. 您可以选择配置 AWS Lambda 以监控 ExampleOutputStream 流并发送警报。有关更多信息，请参阅 使用 Lambda 函数作为输出。您还可以查看 Kinesis Data Analytics 写入到外部目标（Kinesis 流 ExampleOutputStream）的记录，如 步骤 4：验证应用程序输出 中所述。

下一个步骤

步骤 4：验证应用程序输出

步骤 4：验证应用程序输出

在热点示例的此部分中，您将设置一个 Web 应用程序，该应用程序在可扩展矢量图形 (SVG) 控件中显示热点信息。

1. 使用以下内容创建名为 index.html 的文件：

```
<!doctype html>
<html lang=en>
```

```
<head>
    <meta charset=utf-8>
    <title>hotspots viewer</title>

    <style>
    #visualization {
      display: block;
      margin: auto;
    }

    .point {
      opacity: 0.2;
    }

    .hot {
      fill: red;
    }

    .cold {
      fill: blue;
    }

    .hotspot {
      stroke: black;
      stroke-opacity: 0.8;
      stroke-width: 1;
      fill: none;
    }
    </style>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.202.0.min.js"></script>
    <script src="https://d3js.org/d3.v4.min.js"></script>
</head>
<body>
<svg id="visualization" width="600" height="600"></svg>
<script src="hotspots_viewer.js"></script>
</body>
</html>
```

2. 在同一目录中创建一个名为 hotspots_viewer.js 的文件，该文件包含以下内容。在提供的变量中包含您的 、凭证和输出流名称。

```
// Visualize example output from the Kinesis Analytics hotspot detection algorithm.
```

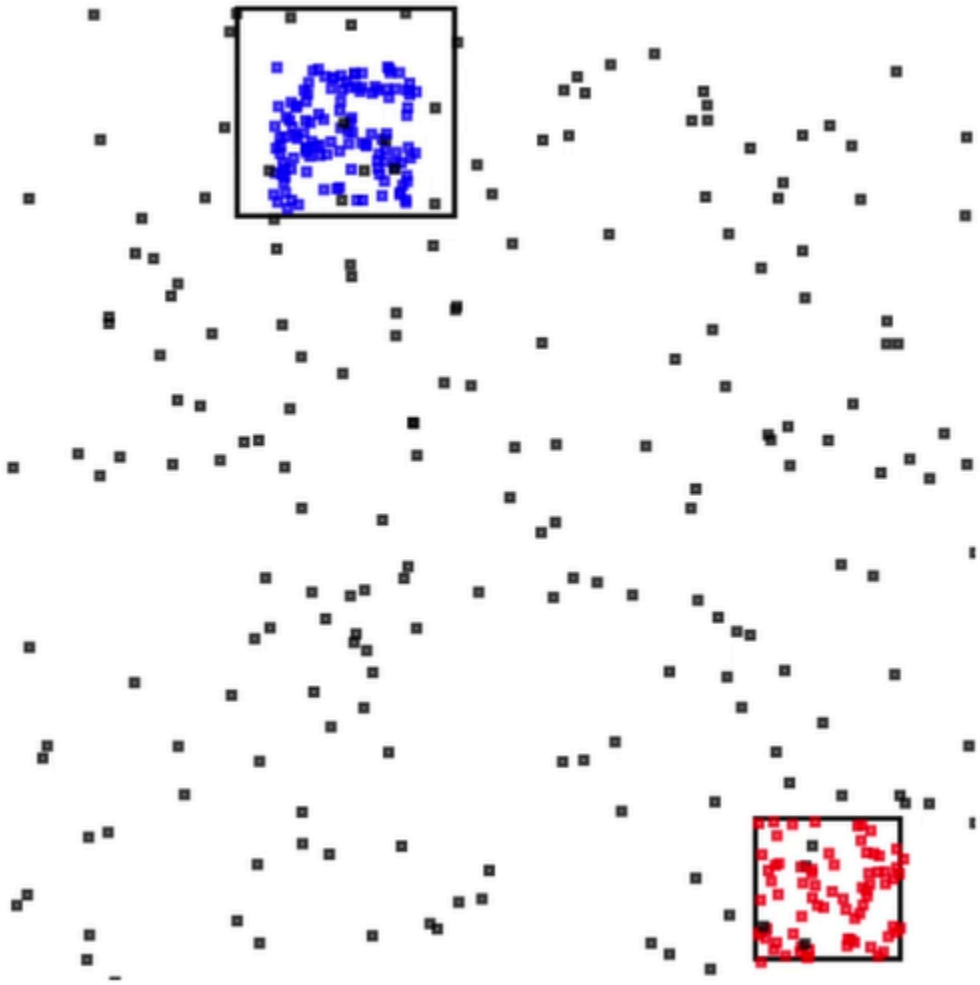
```
// This script assumes that the output stream has a single shard.

// Modify this section to reflect your AWS configuration
var awsRegion = "",        // The  where your Kinesis Analytics application is
 configured.
    accessKeyId = "",      // Your Access Key ID
    secretAccessKey = "",  // Your Secret Access Key
    outputStream = "";     // The name of the Kinesis Stream where the output from
 the HOTSPOTS function is being written

// The variables in this section should reflect way input data was generated and
 the parameters that the HOTSPOTS
// function was called with.
var windowSize = 1000, // The window size used for hotspot detection
    minimumDensity = 40,  // A filter applied to returned hotspots before
 visualization
    xRange = [0, 10],  // The range of values to display on the x-axis
    yRange = [0, 10];  // The range of values to display on the y-axis

////////////////////////////////////////////////////////////////////////////////
// D3 setup
////////////////////////////////////////////////////////////////////////////////

var svg = d3.select("svg"),
    margin = {"top": 20, "right": 20, "bottom": 20, "left": 20},
    graphWidth = +svg.attr("width") - margin.left - margin.right,
    graphHeight = +svg.attr("height") - margin.top - margin.bottom;

// Return the linear function that maps the segment [a, b] to the segment [c, d].
function linearScale(a, b, c, d) {
    var m = (d - c) / (b - a);
    return function(x) {
        return c + m * (x - a);
    };
}

// helper functions to extract the x-value from a stream record and scale it for
 output
var xValue = function(r) { return r.x; },
    xScale = linearScale(xRange[0], xRange[1], 0, graphWidth),
    xMap = function(r) { return xScale(xValue(r)); };

// helper functions to extract the y-value from a stream record and scale it for
 output
```

```
var yValue = function(r) { return r.y; },
    yScale = linearScale(yRange[0], yRange[1], 0, graphHeight),
    yMap = function(r) { return yScale(yValue(r)); };

// a helper function that assigns a CSS class to a point based on whether it was
 generated as part of a hotspot
var classMap = function(r) { return r.is_hot == "Y" ? "point hot" : "point
 cold"; };

var g = svg.append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

function update(records, hotspots) {

    var points = g.selectAll("circle")
        .data(records, function(r) { return r.dataIndex; });

    points.enter().append("circle")
        .attr("class", classMap)
        .attr("r", 3)
        .attr("cx", xMap)
        .attr("cy", yMap);

    points.exit().remove();

    if (hotspots) {
        var boxes = g.selectAll("rect").data(hotspots);

        boxes.enter().append("rect")
            .merge(boxes)
            .attr("class", "hotspot")
            .attr("x", function(h) { return xScale(h.minValues[0]); })
            .attr("y", function(h) { return yScale(h.minValues[1]); })
            .attr("width", function(h) { return xScale(h.maxValues[0]) -
 xScale(h.minValues[0]); })
            .attr("height", function(h) { return yScale(h.maxValues[1]) -
 yScale(h.minValues[1]); });

        boxes.exit().remove();
    }
}

////////////////////////////////////////////////////////////////////////////
// Use the AWS SDK to pull output records from Kinesis and update the visualization
```

```
////////////////////////////////////////////////////////////////////////////////////////////

var kinesis = new AWS.Kinesis({
    "region": awsRegion,
    "accessKeyId": accessKeyId,
    "secretAccessKey": secretAccessKey
});

var textDecoder = new TextDecoder("utf-8");

// Decode an output record into an object and assign it an index value
function decodeRecord(record, recordIndex) {
    var record = JSON.parse(textDecoder.decode(record.Data));
    var hotspots_result = JSON.parse(record.HOTSPOTS_RESULT);
    record.hotspots = hotspots_result.hotspots
        .filter(function(hotspot) { return hotspot.density >= minimumDensity});
    record.index = recordIndex
    return record;
}

// Fetch a new records from the shard iterator, append them to records, and update
 the visualization
function getRecordsAndUpdateVisualization(shardIterator, records, lastRecordIndex)
 {
    kinesis.getRecords({
        "ShardIterator": shardIterator
    }, function(err, data) {
        if (err) {
            console.log(err, err.stack);
            return;
        }

        var newRecords = data.Records.map(function(raw) { return decodeRecord(raw,
 ++lastRecordIndex); });
        newRecords.forEach(function(record) { records.push(record); });

        var hotspots = null;
        if (newRecords.length > 0) {
            hotspots = newRecords[newRecords.length - 1].hotspots;
        }

        while (records.length > windowSize) {
            records.shift();
        }
```

```
        update(records, hotspots);

        getRecordsAndUpdateVisualization(data.NextShardIterator, records,
 lastRecordIndex);
    });
}

// Get a shard iterator for the output stream and begin updating the visualization.
 Note that this script will only
// read records from the first shard in the stream.
function init() {
    kinesis.describeStream({
        "StreamName": outputStream
    }, function(err, data) {
        if (err) {
            console.log(err, err.stack);
            return;
        }

        var shardId = data.StreamDescription.Shards[0].ShardId;

        kinesis.getShardIterator({
            "StreamName": outputStream,
            "ShardId": shardId,
            "ShardIteratorType": "LATEST"
        }, function(err, data) {
            if (err) {
                console.log(err, err.stack);
                return;
            }
            getRecordsAndUpdateVisualization(data.ShardIterator, [], 0);
        })
    });
}

// Start the visualization
init();
```

3. 在第一个部分中 Python 代码运行时，在 Web 浏览器中打开 `index.html`。热点信息显示在页面上，如下所示。



示例：警报和错误

此部分提供使用警报和错误的 Kinesis Data Analytics 应用程序示例。每个示例提供分步说明和代码以帮助设置和测试 Kinesis Data Analytics 应用程序。

主题

- 示例：创建简单警报
- 示例：创建受限警报
- 示例：探索应用程序内部错误流

示例：创建简单警报

在此 Kinesis Data Analytics 应用程序中，将在通过演示流创建的应用程序内部流上持续运行查询。有关更多信息，请参阅 连续查询。

如果任何行显示股票价格变动大于 1%，这些行将被插入另一个应用程序内部流中。在本练习中，可以将应用程序输出配置为将结果保存到外部目标。随后，可以进一步调查结果。例如，您可以使用 AWS Lambda 函数处理记录和发送警报。

创建简单的警报应用程序

1. 创建分析应用程序，如 Kinesis Data Analytics 入门练习中所述。
2. 在 Kinesis Data Analytics 上的 SQL 编辑器中，将应用程序代码替换为以下内容：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
           (ticker_symbol VARCHAR(4),
            sector        VARCHAR(12),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM ticker_symbol, sector, change, price
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  (ABS(Change / (Price - Change)) * 100) > 1;
```

   应用程序代码中的 SELECT 语句可在 SOURCE_SQL_STREAM_001 中筛选出股票价格变化大于 1% 的行。之后，该代码将使用数据泵将这些行插入到另一个应用程序内部流 DESTINATION_SQL_STREAM。有关说明使用数据泵将行插入应用程序内部流中的编码模式的更多信息，请参阅应用程序代码。
3. 选择 保存并运行 SQL。
4. 添加一个目标。为此，请在 SQL 编辑器中选择 Destination (目标)，也可以在应用程序中心中选择 Add a destination (添加目标)。

   a. 在 SQL 编辑器中，选择 Destination (目标) 选项卡，然后选择 Connect to a destination (连接到目标)。

      在 Connect to destination (连接到目标) 页面中，选择 Create New (新建)。

   b. 选择 Go to Kinesis Streams。

c. 在 Amazon Kinesis Data Streams 控制台中，创建具有一个分片的新 Kinesis 流（例如，gs-destination）。请等待，直到流状态为 ACTIVE。

d. 返回 Kinesis Data Analytics 控制台。在 Connect to destination (连接到目标) 页面上，选择您创建的流。

如果流未显示，请刷新页面。

e. 选择 保存并继续。

现在，您具有一个外部目标（Kinesis 数据流），Kinesis Data Analytics 将 DESTINATION_SQL_STREAM 应用程序内部流中的应用程序输出永久保存到该目标中。

5. 配置 AWS Lambda 以监控您创建的 Kinesis 流并调用一个 Lambda 函数。

有关说明，请参阅 使用 Lambda 函数预处理数据。

示例：创建受限警报

在此 Kinesis Data Analytics 应用程序中，将在通过演示流创建的应用程序内部流上持续运行查询。有关更多信息，请参阅 连续查询。如果任何行显示股票价格变动大于 1%，这些行将被插入另一个应用程序内部流中。应用程序会限制警报，以便在股票价格变化时立即发送警报。但每个股票代号每分钟向应用程序内部流发送的警报不超过一个。

创建受限警报应用程序

1. 创建一个 Kinesis Data Analytics 应用程序，如 Kinesis Data Analytics 入门练习中所述。
2. 在 Kinesis Data Analytics 上的 SQL 编辑器中，将应用程序代码替换为以下内容：

```
CREATE OR REPLACE STREAM "CHANGE_STREAM"
           (ticker_symbol VARCHAR(4),
            sector        VARCHAR(12),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "change_pump" AS
   INSERT INTO "CHANGE_STREAM"
      SELECT STREAM ticker_symbol, sector, change, price
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  (ABS(Change / (Price - Change)) * 100) > 1;
```

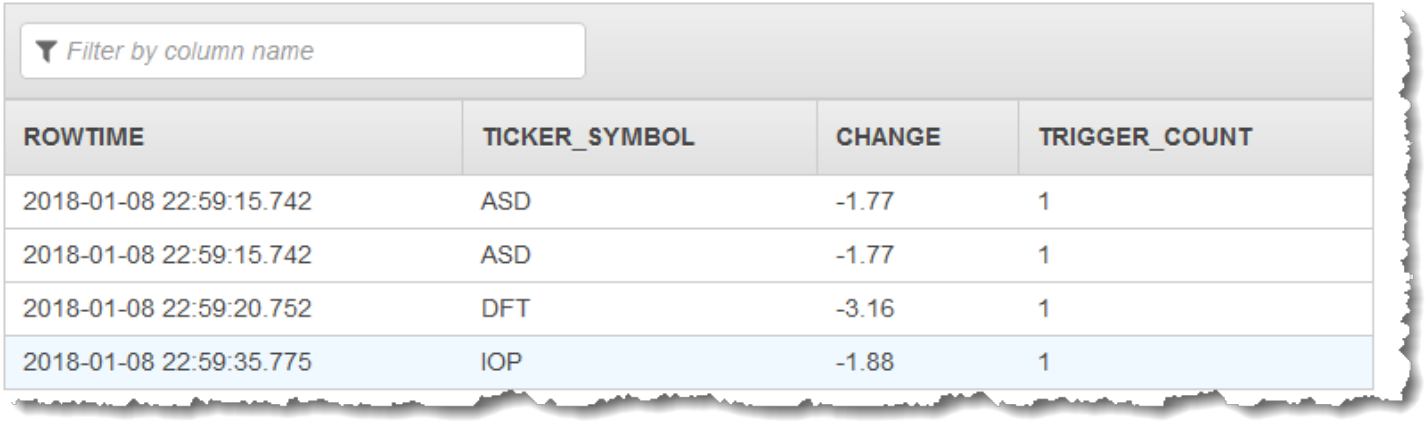
```
-- ** Trigger Count and Limit **
-- Counts "triggers" or those values that evaluated true against the previous where
 clause
-- Then provides its own limit on the number of triggers per hour per ticker symbol
 to what
-- is specified in the WHERE clause

CREATE OR REPLACE STREAM TRIGGER_COUNT_STREAM (
   ticker_symbol VARCHAR(4),
   change REAL,
   trigger_count INTEGER);

CREATE OR REPLACE PUMP trigger_count_pump AS INSERT INTO TRIGGER_COUNT_STREAM
SELECT STREAM ticker_symbol, change, trigger_count
FROM (
    SELECT STREAM ticker_symbol, change, COUNT(*) OVER W1 as trigger_count
    FROM "CHANGE_STREAM"
    --window to perform aggregations over last minute to keep track of triggers
    WINDOW W1 AS (PARTITION BY ticker_symbol RANGE INTERVAL '1' MINUTE PRECEDING)
)
WHERE trigger_count >= 1;
```

应用程序代码中的 SELECT 语句将在 SOURCE_SQL_STREAM_001 中筛选出显示股票价格更改大于 1% 的行，并使用数据泵将这些行插入另一个应用程序内部流 CHANGE_STREAM。

然后，应用程序为受限警报创建第二个名为 TRIGGER_COUNT_STREAM 的流。第二个查询从一个窗口中选择记录，每次记录被允许进入该窗口时，该窗口都向前跳，以便每个股票报价每分钟只有一个记录被写入到流中。

3. 选择 保存并运行 SQL。

该示例将流输出到与以下内容类似的 TRIGGER_COUNT_STREAM：



示例：探索应用程序内部错误流

Amazon Kinesis Data Analytics 为您创建的每个应用程序提供一个应用程序内部错误流。应用程序无法处理的所有行将发送到此错误流。您可以考虑将错误流数据保存到外部目标以便于调查。

您将在控制台中执行以下练习。在这些示例中，您将通过编辑由发现过程推断的架构将错误引入输入配置中，然后验证已发送到错误流的行。

主题

- 引入分析错误
- 引入被零除错误

引入分析错误

在本练习中，您引入了分析错误。

1. 创建一个 Kinesis Data Analytics 应用程序，如 Kinesis Data Analytics 入门练习中所述。
2. 在应用程序详细信息页面上，选择 连接流数据。
3. 如果已完成入门练习，则账户中会有一个演示流 (`kinesis-analytics-demo-stream`)。在 Connect to source (连接到源) 页面上，选择此演示流。
4. Kinesis Data Analytics 使用演示流中的示例推断它创建的应用程序内部输入流的架构。控制台在 Formatted stream sample 选项卡中显示推断的架构和示例数据。
5. 接下来，可以编辑架构并修改列类型以引入分析错误。选择 Edit schema。
6. 将 `TICKER_SYMBOL` 列类型从 `VARCHAR(4)` 更改为 `INTEGER`。

   由于创建的应用程序内部架构中的列类型无效，Kinesis Data Analytics 无法将数据添加到应用程序内部流中。而只能将行发送到错误流。

7. 选择 Save schema。
8. 选择 Refresh schema samples。

   请注意，Formatted stream 示例中没有行。不过，Error stream 选项卡将显示数据与错误消息。Error stream 选项卡将显示已发送到应用程序内部错误流的数据。

   由于已更改列数据类型，Kinesis Data Analytics 无法将数据添加到应用程序内部输入流中。而只能将数据发送到错误流。

引入被零除错误

在本练习中，您将更新应用程序代码以引入运行时错误 (被零除)。请注意，Amazon Kinesis Data Analytics 将结果行发送到应用程序内部错误流，而不是发送到要将结果写入到的 `DESTINATION_SQL_STREAM` 应用程序内部错误流。

1. 创建一个 Kinesis Data Analytics 应用程序，如 Kinesis Data Analytics 入门练习中所述。

   验证 Real-time analytics 选项卡上的结果，如下所示：

   Sour
2. 在应用程序代码中更新 `SELECT` 语句以引入被零除；例如：

   ```
   SELECT STREAM ticker_symbol, sector, change, (price / 0) as ProblemColumn
   FROM "SOURCE_SQL_STREAM_001"
   WHERE sector SIMILAR TO '%TECH%';
   ```

3. 运行应用程序。

   由于出现被零除运行时错误，Kinesis Data Analytics 将行发送到应用程序内部错误流，而不是将结果写入到 `DESTINATION_SQL_STREAM` 中。在 Real-time analytics (实时分析) 选项卡上，选择错误流，随后应用程序内部错误流中将显示行。

示例：解决方案加速器

AWS Solutions 站点提供了 AWS CloudFormation 模板，可用于快速创建完整的流数据解决方案。

可用模板如下：

实时洞察 AWS 账户 活动

此解决方案实时记录和直观显示您的 AWS 账户 的资源访问和使用指标。有关更多信息，请参阅实时洞察 AWS 账户 活动。

使用 Kinesis Data Analytics 实时监控 AWS IoT 设备

此解决方案实时收集、处理、分析和直观显示 IoT 设备连接和活动数据。有关更多信息，请参阅使用 Kinesis Data Analytics 进行实时 AWS IoT 设备监控。

使用 Kinesis Data Analytics 进行实时 Web 分析

此解决方案实时收集、处理、分析和直观显示网站点击流数据。有关更多信息，请参阅使用 Kinesis Data Analytics 进行实时 Web 分析。

Amazon Connect 车辆解决方案

此解决方案实时收集、处理、分析和直观显示车辆中的 IoT 数据。有关更多信息，请参阅 Amazon Connect 车辆解决方案。

中的安全性

云安全 AWS 是重中之重。作为 AWS 客户，您将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 的合规性计划，请参阅合规性计划范围内的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。

此文档将帮助您了解如何在使用 时应用责任共担模式。以下主题说明如何配置 以实现您的安全性和合规性目标。您还将了解如何使用其他 Amazon 服务来帮助您监控和保护您的资源。

主题

适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 数据保护

您可以使用提供的工具保护您的数据 AWS。Kinesis Data Analytics 可以与支持加密数据的服务配合使用，包括 Kinesis Data Streams、Firehose 和 Amazon S3。

Kinesis Data Analytics 数据加密

静态加密

在使用 Kinesis Data Analytics 静态加密数据时，请注意以下事项：

- 您可以使用对传入的 Kinesis 数据流上的数据进行加密。StartStreamEncryption有关更多信息，请参阅什么是 Kinesis 数据流的服务器端加密？。
- 可以使用 Firehose 对输出数据进行静态加密，将数据存储在加密的 Amazon S3 存储桶中。您可以指定您的 Amazon S3 存储桶使用的加密密钥。有关更多信息，请参阅使用具有 KMS 托管密钥的服务器端加密（SSE-KMS）保护数据。
- 您的应用程序的代码是静态加密的。
- 您的应用程序的参考数据是静态加密的。

传输中加密

Kinesis Data Analytics 对传输中的所有数据进行加密。传输中加密对所有 Kinesis Data Analytics 应用程序都处于启用状态，无法禁用。

Kinesis Data Analytics 在以下场景中对传输中的数据进行加密：

- 从 Kinesis Data Streams 传输到 Kinesis Data Analytics 的数据。
- Kinesis Data Analytics 的内部组件之间的传输中的数据。
- 在 Kinesis Data Analytics 和 Firehose 之间传输的数据。

密钥管理

Kinesis Data Analytics 中的数据加密使用服务托管的密钥。客户管理的密钥不受支持。

Kinesis Data Analytics 中的身份和访问管理

Amazon Kinesis Data Analytics 需要适当权限才能从应用程序输入配置中指定的流式传输源读取记录。Amazon Kinesis Data Analytics 还需要权限才能将您在应用程序输出写入您在应用程序输出配置中指定的流。

您可以创建 Amazon Kinesis Data Analytics 可担任的 IAM 角色以授予这些权限。您为该角色授予的权限决定了 Amazon Kinesis Data Analytics 服务在担任该角色时可以执行的操作。

> **Note**
>
> 如果要自己创建 IAM 角色，此部分中的信息是非常有用的。在 Amazon Kinesis Data Analytics 控制台中创建应用程序时，控制台可以在此时为您创建一个 IAM 角色。对于创建的 IAM 角色，控制台使用以下命名约定：
>
> ```
> kinesis-analytics-ApplicationName
> ```
>
> 在创建角色后，您可以在 IAM 控制台中查看该角色和附加的策略。

每个 IAM 角色附加了两个策略。在信任策略中，您可以指定谁可以代入该角色。在权限策略（可以有一个或多个）中，您应当指定要向此角色授予的权限。以下部分说明了这些策略，您可以在创建 IAM 角色时使用这些策略。

信任策略

要为 Amazon Kinesis Data Analytics 授予代入某个角色的权限以访问流式传输源或引用源，您可以将以下信任策略附加到 角色：

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

权限策略

如果要创建 IAM 角色以允许 Amazon Kinesis Data Analytics 从应用程序流式传输源中读取，您必须授予相关读取操作的权限。根据您的来源（例如 Kinesis 流、Firehose 传输流或 Amazon S3 存储桶中的参考来源），您可以附加以下权限策略。

读取 Kinesis 流的权限策略

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputKinesis",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": [
                "arn:aws:kinesis:aws-region:aws-account-id:stream/inputStreamName"
            ]
        }
    ]
}
```

读取 Firehose 传送流的权限策略

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputFirehose",
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:Get*"
            ],
            "Resource": [
                "arn:aws:firehose:aws-region:aws-account-
id:deliverystream/inputFirehoseName"
            ]
        }
    ]
}
```

> **Note**
>
> `firehose:Get*` 权限是指 Kinesis Data Analytics 用于访问流的内部访问器。Firehose 直播没有公共访问器。

如果您指示 Amazon Kinesis Data Analytics 将输出写入到应用程序输出配置中的外部目标，您需要为 IAM 角色授予以下权限。

写入到 Kinesis 流的权限策略

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "WriteOutputKinesis",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:aws-region:aws-account-id:stream/output-stream-name"
            ]
        }
    ]
}
```

写入到 Firehose 传输流的权限策略

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "WriteOutputFirehose",
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
```

```
            ],
            "Resource": [
                "arn:aws:firehose:aws-region:aws-account-id:deliverystream/output-
firehose-name"
            ]
        }
    ]
}
```

从 Amazon S3 存储桶中读取引用数据源的权限策略

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

防止跨服务混淆代理

在中 AWS，当一个服务（调用服务）调用另一个服务（被调用的服务）时，可能会发生跨服务模拟。尽管调用服务不应具有适当的权限，但仍可操纵以对另一个客户的资源进行操作，这会导致代理混淆。

为了防止众议员感到困惑，我们 AWS 提供了一些工具，这些工具可帮助您使用已获准访问您账户中资源的服务委托人保护所有服务的数据。本节重点介绍的是 Kinesis Data Analytics 特有的跨服务混淆代理人问题防范功能，您可以在 IAM 用户指南的混淆代理人问题部分了解更多相关信息。

在 Kinesis Data Analytics for SQL 的背景下，我们建议在角色信任策略中使用 a w s SourceArnSourceAccount: 和 aws: 全局条件上下文密钥，将对角色的访问权限限制为仅限预期资源生成的那些请求。

如果您只希望将一个资源与跨服务访问相关联，请使用 `aws:SourceArn`。如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用 `aws:SourceAccount`。

aws:SourceArn 的值必须是 Kinesis Data Analytics 使用资源的 ARN，其指定格式为：arn:aws:kinesisanalytics:region:account:resource。

防范混淆代理问题的建议方法是使用 aws:SourceArn 全局条件上下文键和资源的完整 ARN。

如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符 (*) 的 aws:SourceArn 键。例如：arn:aws:kinesisanalytics::111122223333:*。

虽然 Kinesis Data Analytics for SQL API 中的大多数操作（CreateApplication例如AddApplicationInputDeleteApplication和）都是在特定应用程序的上下文中执行的，但DiscoverInputSchema该操作不会在任何应用程序的上下文中执行。这意味着此操作中使用的角色不得在 SourceArn 条件键中完全指定资源。以下是使用通配符 ARN 的示例：

```
{
   ...
   "ArnLike":{
      "aws:SourceArn":"arn:aws:kinesisanalytics:us-east-1:123456789012:*"
   }
   ...
}
```

Kinesis Data Analytics for SQL 生成的默认角色使用此通配符。这可确保控制台发现输入架构的无缝体验。但是，建议在发现架构后编辑信任策略，从而使用完整 ARN，充分缓解混淆代理人问题。

您向 Kinesis Data Analytics 提供的角色策略以及为您生成的角色的信任策略可以使用 awsSourceArn: 和 a w sSourceAccount: 条件密钥。

为了防止出现代理混淆的问题，请执行以下步骤：

防止出现代理混淆问题

1. 登录 AWS 管理控制台并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 选择角色，然后选择要修改的角色。
3. 选择编辑信任策略。
4. 在编辑信任策略页面上，将默认 JSON 策略替换为使用 aws:SourceArn 和 aws:SourceAccount 全局条件上下文密钥中的一个或两个的策略。请参阅以下示例策略：
5. 选择更新策略。

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"kinesisanalytics.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"Account ID"
            },
            "ArnEquals":{
               "aws:SourceArn":"arn:aws:kinesisanalytics:us-
east-1:123456789012:application/my-app"
            }
         }
      }
   ]
}
```

的身份验证和访问控制

访问 需要凭证。这些证书必须具有访问 AWS 资源的权限，例如应用程序或亚马逊弹性计算云 (Amazon EC2) 实例。下列各节详述如何使用 AWS Identity and Access Management (IAM) 和 来帮助保护对您的资源的访问。

访问控制

您可以使用有效的凭证来对自己的请求进行身份验证，但您还必须拥有权限才能创建或访问 资源。例如，您必须具有创建 应用程序的权限。

下面几节介绍如何管理 的权限。我们建议您先阅读概述。

- 管理 资源的访问权限概述
- 为 使用基于身份的策略（IAM 策略）
- API 权限：操作、权限和资源参考

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证 和《IAM 用户指南》中的在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，我们建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账

户 和应用程序中使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果有一些特定的使用场景需要长期凭证以及 IAM 用户，我们建议轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个用于指定一组 IAM 用户的身份。您不能使用群组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人担任。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问——要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限——IAM 用户或角色可代入 IAM 角色，以暂时获得针对特定任务的不同权限。
- 跨账户访问——您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问账户中的资源。角色是授予跨账户存取权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 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，此操作然后在不同服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以担任代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的何时创建 IAM 角色（而不是用户）。

管理 资源的访问权限概述

> **Warning**
>
> 对于新项目，建议您使用全新适用于 Apache Flink Studio 的托管服务，而不是适用于 SQL 应用程序的托管服务。Managed Service for Apache Flink Studio 不仅操作简单，还具有高级分析功能，使您能够在几分钟内构建复杂的流处理应用程序。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以代入的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户群组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

> **Note**
>
> 账户管理员（或管理员用户）是具有管理员权限的用户。有关更多信息，请参阅《IAM 用户指南》中的 IAM 最佳实操。

主题

资源和操作

在 中，主要资源是应用程序。在策略中，您可以使用 Amazon 资源名称（ARN）标识策略应用到的资源。

这些资源具有关联的唯一 Amazon 资源名称 (ARN)，如下表所示。

资源类型	ARN 格式
应用程序	`arn:aws:kinesisanalytics:` *`region`*`:`*`account-id`*`:application/`*`application-name`

提供一组操作用来处理资源。有关可用操作的列表，请参阅 操作。

了解资源所有权

AWS 账户 拥有在账户中创建的资源，无论谁创建了这些资源。具体而言，资源所有者是 AWS 账户 对资源创建请求进行身份验证的委托人实体（即根账户、用户或 IAM 角色）。以下示例说明了它的工作原理：

- 如果您使用您的 AWS 账户 根账户证书创建应用程序，则您 AWS 账户 就是该资源的所有者。（在中，资源是应用程序。）
- 如果您在中创建用户 AWS 账户 并向该用户授予创建应用程序的权限，则该用户可以创建应用程序。但是，用户所属的您 AWS 账户拥有应用程序资源。强烈建议您向角色而不是用户授予权限。
- 如果您在中创建 AWS 账户 具有创建应用程序权限的 IAM 角色，则任何能够担任该角色的人都可以创建应用程序。用户 AWS 账户所属的您拥有应用程序资源。

管理对资源的访问

权限策略规定谁可以访问哪些内容。下一节介绍创建权限策略时的可用选项。

> **Note**
>
> 本节讨论如何在 范围内使用 IAM。这里不提供有关 IAM 服务的详细信息。有关完整的 IAM 文档，请参阅《IAM 用户指南》中的什么是 IAM？。有关 IAM policy 语法和说明的信息，请参阅 IAM 用户指南中的 IAM JSON 策略参考。

附加到 IAM 身份的策略称作基于身份的策略 (IAM policy)。附加到资源的策略称作基于资源的策略。仅支持基于身份的策略 (IAM policy)。

主题

- 基于身份的策略（IAM 策略）
- 基于资源的策略

基于身份的策略（IAM 策略）

您可以向 IAM 身份附加策略。例如，您可以执行以下操作：

- 将权限策略附加到您账户中的用户或组 - 要向用户授予创建 资源 (例如应用程序) 的权限，您可以将权限策略附加到用户或用户所属的组。

- 向角色附加权限策略（授予跨账户权限）– 您可以向 IAM 角色附加基于身份的权限策略，以授予跨账户的权限。例如，账户 A 中的管理员可以创建一个角色来向其他账户 AWS 账户 （例如账户 B）或 Amazon 服务授予跨账户权限，如下所示：

  1. 账户 A 管理员可以创建一个 IAM 角色，然后向该角色附加授予其访问账户 A 中资源的权限策略。
  2. 账户 A 管理员可以向角色挂载信任策略，将账户 B 标识为能够担任该角色的委托人。
  3. 之后，账户 B 管理员可以委托权限，指派账户 B 中的任何用户代入该角色。这样，账户 B 中的用户就可以在账户 A 中创建或访问资源了。如果您需要授予 Amazon 服务权限来代入该角色，则信任策略中的委托人也可以是 Amazon 服务委托人。

  有关使用 IAM 委托权限的更多信息，请参阅 IAM 用户指南中的访问权限管理。

下面是为 `kinesisanalytics:CreateApplication` 操作授予权限的示例策略，需要具有该权限才能创建 应用程序。

> **Note**
>
> 这是介绍性示例策略。当您将策略附加到用户时，用户将能够使用 AWS CLI 或 AWS SDK 创建应用程序。但用户需要更多权限才能配置输入和输出。此外，使用控制台时，用户需要更多权限。后续章节将提供更多相关信息。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1473028104000",
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:CreateApplication"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

有关对 使用基于身份的策略的更多信息，请参阅 为 使用基于身份的策略（IAM 策略） 。有关用户、组、角色和权限的更多信息，请参阅《IAM 用户指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id.html中的身份（用户、组和角色）。

基于资源的策略

其他服务（如 Amazon S3）还支持基于资源的权限策略。例如，您可以将基于资源的策略附加到 S3 存储桶以管理对该存储桶的访问权限。 不支持基于资源的策略。

指定策略元素：操作、效果和主体

对于每种 资源，该服务都定义了一组 API 操作。为授予这些 API 操作的权限， 定义了一组您可以在策略中指定的操作。某些 API 操作可能需要多个操作的权限才能执行 API 操作。有关资源和 API 操作的更多信息，请参阅 资源和操作 和 操作。

以下是最基本的策略元素：

- 资源 – 您使用 Amazon 资源名称（ARN）来标识策略应用到的资源。有关更多信息，请参阅 资源和操作。
- 操作 – 您可以使用操作关键字标识要允许或拒绝的资源操作。例如，您可以使用 `create` 允许用户创建应用程序。
- 效果 – 用于指定用户请求特定操作时的效果（可以是允许或拒绝）。如果没有显式授予（允许）对资源的访问权限，则隐式拒绝访问。您也可显式拒绝对资源的访问，这样可确保用户无法访问该资源，即使有其他策略授予了访问权限的情况下也是如此。
- 主体 – 在基于身份的策略 (IAM policy) 中，附加了策略的用户是隐式主体。对于基于资源的策略，您可以指定要接收权限的用户、账户、服务或其他实体（仅适用于基于资源的策略）。 不支持基于资源的策略。

有关 IAM policy 语法和介绍的更多信息，请参阅 IAM 用户指南中的 IAM JSON 策略参考。

有关显示所有 API 操作及其适用于的资源的列表，请参阅 API 权限：操作、权限和资源参考。

在策略中指定条件

当您授予权限时，可使用访问策略语言来指定规定策略何时生效的条件。例如，您可能希望策略仅在特定日期后应用。有关使用策略语言指定条件的更多信息，请参阅《IAM 用户指南》中的条件。

要表示条件，您可以使用预定义的条件键。没有特定于 的条件键。但是，您可以根据需要使用 AWS范围内的条件键。有关 AWS范围密钥的完整列表，请参阅 IAM 用户指南中的条件可用密钥。

为 使用基于身份的策略（IAM 策略）

下面提供了基于身份的策略的示例，这些示例展示了账户管理员如何将权限策略附加到 IAM 身份（即用户、组和角色），从而授予对资源执行操作的权限。

> **Important**
>
> 我们建议您首先阅读以下介绍性主题，这些主题说明了可用于管理 资源访问的基本概念和选项。有关更多信息，请参阅管理 资源的访问权限概述。

主题

- 使用 控制台所需要的权限
- 适用于 的 Amazon 托管（预定义）策略
- 客户托管策略示例

下面介绍权限策略示例。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1473028104000",
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:CreateApplication"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

该策略包含一个语句：

- 第一个语句使用应用程序的 Amazon 资源名称 (ARN) 授予权限以对资源执行一个 操作 (`kinesisanalytics:CreateApplication`)。此示例中的 ARN 指定通配符 (*)，表示为任何资源授予相应权限。

有关显示所有 API 操作及其适用资源的表，请参阅 API 权限：操作、权限和资源参考。

使用 控制台所需要的权限

您必须为用户授予所需的权限才能使用 控制台。例如，如果希望用户拥有相应权限以创建应用程序，请授予显示账户中的流式传输源的权限，以便用户可以在控制台中配置输入和输出。

我们建议执行下列操作：

- 使用 Amazon 托管策略向用户授权。有关可用的策略，请参阅适用于 的 Amazon 托管（预定义）策略。
- 创建自定义策略。在本示例中，我们建议您查看此部分中提供的示例。有关更多信息，请参阅 客户托管策略示例。

适用于 的 Amazon 托管（预定义）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。这些 Amazon 托管策略可针对常用案例授予必要的权限，使您免去调查所需权限的工作。有关更多信息，请参阅《IAM 用户指南》中的 Amazon 托管策略。

以下 Amazon 托管策略（可附加到您账户中的用户）特定于：

- **`AmazonKinesisAnalyticsReadOnly`** - 为操作授予权限以允许用户列出 应用程序并查看输入/输出配置。它还授予权限，允许用户查看 Kinesis 直播和 Firehose 直播流列表。应用程序运行时，用户可以在控制台中查看源数据和实时分析结果。

- **`AmazonKinesisAnalyticsFullAccess`** - 为所有操作授予权限，并授予所有其他权限以允许用户创建和管理 应用程序。但是，请注意以下事项：

  - 如果用户要在控制台中创建一个新的 IAM 角色，这些权限并不能满足需求（这些权限允许用户选择现有角色）。如果您希望用户能够在控制台中创建 IAM 角色，请添加 `IAMFullAccess` Amazon 托管策略。

- 在配置应用程序时，用户必须具有 `iam:PassRole` 操作权限才能指定 IAM 角色。此 Amazon 托管策略将 `iam:PassRole` 操作的权限仅授予给以前缀 `service-role/kinesis-analytics` 开头的 IAM 角色中的用户。

  如果用户要为 应用程序配置的角色没有该前缀，您必须先在特定角色中为用户明确授予 `iam:PassRole` 操作的权限。

此外，您还可以创建您自己的自定义 IAM policy，以授予 操作和资源的相关权限。您可以将这些自定义策略附加到需要这些权限的 用户或组。

客户托管策略示例

此部分中的示例提供了一组可附加到用户的示例策略。如果您是首次创建策略，建议您先在账户中创建一个用户。然后，按顺序将策略附加到用户，如此部分中的步骤所述。然后，在将每个策略附加到用户时，可使用控制台验证该策略的效果。

最初，用户没有权限并且无法在控制台中执行任何操作。在将策略附加到用户时，可以验证用户是否能在控制台中执行各种操作。

建议使用两种浏览器窗口。在一个窗口中，创建用户和授予权限。另一方面， AWS Management Console 使用用户的证书登录，并在您授予权限时验证权限。

有关说明如何创建可用作资源执行角色的 IAM 角色的示例，请参阅《IAM 用户指南》中的创建 IAM 角色。

示例步骤

- 步骤 1：创建一个 IAM 用户
- 步骤 2：为用户授予非 特定的操作的权限
- 步骤 3：允许用户查看应用程序列表和查看详细信息
- 步骤 4：允许用户启动特定应用程序
- 步骤 5：允许用户创建 应用程序
- 步骤 6：允许应用程序使用 Lambda 预处理

步骤 1：创建一个 IAM 用户

首先，您需要创建一个用户，将该用户添加到具有管理权限的 IAM 组，然后向您创建的用户授予管理权限。然后，您可以使用特殊的 URL 和该用户的凭据 AWS 进行访问。

有关说明，请参考《IAM 用户指南》中的创建您的第一个 IAM 用户和管理员组。

步骤 2：为用户授予非 特定的操作的权限

首先，为用户授予非 特定的所有操作的权限，在用户使用 应用程序时需要具有这些权限。其中包括处理流的权限（Amazon Kinesis Data Streams 操作、Amazon Data Firehose 操作）和操作权限。CloudWatch 将下面的策略附加到用户。

您需要通过提供要向其授予 `iam:PassRole` 权限的 IAM 角色名来更新策略，或者指定通配符 (*) 来表示所有 IAM 角色。这不是安全做法；但是，您可能未在此测试期间创建特定 IAM 角色。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:CreateStream",
                "kinesis:DeleteStream",
                "kinesis:DescribeStream",
                "kinesis:ListStreams",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:ListDeliveryStreams"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "logs:GetLogEvents",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListPolicyVersions",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/service-role/role-name"
        }
    ]
}
```

步骤 3：允许用户查看应用程序列表和查看详细信息

以下策略为用户授予以下权限：

- kinesisanalytics:ListApplications 操作的权限，以便用户可以查看应用程序列表。这是服务级别的 API 调用，因此您应当指定“*”作为 Resource 值。
- kinesisanalytics:DescribeApplication 操作的权限，以便您可以获取任何应用程序的相关信息。

将此策略添加到用户。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:ListApplications"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:DescribeApplication"
            ],
            "Resource": "arn:aws:kinesisanalytics:aws-region:aws-account-
id:application/*"
        }
    ]
}
```

使用用户凭证登录到控制台以验证这些权限。

步骤 4：允许用户启动特定应用程序

如果您希望用户可以启动某个现有 应用程序，请将以下策略附加到用户。该策略提供 kinesisanalytics:StartApplication 操作的权限：您必须通过提供您的账户 ID、 AWS 地区和应用程序名称来更新政策。

```
{
    "Version": "2012-10-17",
    "Statement": [
                {
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:StartApplication"
            ],
            "Resource": "arn:aws:kinesisanalytics:aws-region:aws-account-
id:application/application-name"
        }
    ]
}
```

步骤 5：允许用户创建 应用程序

如果您希望用户创建 应用程序，您可以将以下策略附加到该用户。您必须更新政策并提供 AWS 区域、您的账户 ID 以及您希望用户创建的特定应用程序名称，或者 “*” 以便用户可以指定任何应用程序名称（从而创建多个应用程序）。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "Stmt1473028104000",
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:CreateApplication"
            ],
            "Resource": [
                "*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:StartApplication",
                "kinesisanalytics:UpdateApplication",
                "kinesisanalytics:AddApplicationInput",
                "kinesisanalytics:AddApplicationOutput"
            ],
            "Resource": "arn:aws:kinesisanalytics:aws-region:aws-account-
id:application/application-name"
        }
    ]
}
```

步骤 6：允许应用程序使用 Lambda 预处理

如果希望应用程序能够使用 Lambda 预处理，请将以下策略附加到角色。

```
    {
      "Sid": "UseLambdaFunction",
      "Effect": "Allow",
      "Action": [
          "lambda:InvokeFunction",
          "lambda:GetFunctionConfiguration"
      ],
      "Resource": "<FunctionARN>"
  }
```

API 权限：操作、权限和资源参考

在设置 访问控制 和编写可附加到 IAM 身份的权限策略（基于身份的策略）时，可使用下面的 列表作为参考。包括每个 API 操作、您可以为其授予执行操作权限的相应操作，以及您可以为其授予权限的

AWS 资源。您可以在策略的 Action 字段中指定这些操作，并在策略的 Resource 字段中指定资源值。

您可以在策略中使用 AWS-wide 条件键来表达条件。有关 AWS范围密钥的完整列表，请参阅 IAM 用户指南中的可用密钥。

> **Note**
>
> 要指定操作，请在 API 操作名称之前使用 kinesisanalytics 前缀（例如，kinesisanalytics:AddApplicationInput）。

API 和必需的操作权限

API 操作：

所需权限（API 操作）：

资源：

API 和必需的操作权限

Amazon RDS API 和必需的操作权限

API 操作：AddApplicationInput

操作：kinesisanalytics:AddApplicationInput

资源：

arn:aws:kinesisanalytics: *region*:*accountId*:application/*application-name*

GetApplicationState

控制台使用名为 GetApplicationState 的内部方法对应用程序数据进行采样或访问。您的服务应用程序需要具有内部 kinesisanalytics:GetApplicationState API 的权限，才能通过 AWS Management Console对应用程序数据进行采样或访问。

监控

为您的应用程序提供监控功能。有关更多信息，请参阅监控。

Amazon Kinesis Data Analytics for SQL Applications 合规验证

作为 AWS 多项合规计划的一部分，第三方审计机构评估亚马逊 Kinesis Data Analytics 的安全性和合规性。其中包括 SOC、PCI、HIPAA 等。

有关特定合规计划范围内的 AWS 服务列表，请参阅按合规计划划分的范围内的亚马逊服务。有关一般信息，请参阅 AWS 合规性计划。

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 AWS Artifact。

使用 Kinesis Data Analytics 时，您的合规责任根据您的数据敏感性、公司的合规目标以及适用的法律法规决定。如果您对 Kinesis Data Analytics 的使用需遵守 HIPAA 或 PCI 等标准， AWS 将提供以下有用资源：

- 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，并提供了在上部署以安全性和合规性为重点的基准环境的步骤。 AWS
- HIPAA 安全与合规架构白皮书 — 本白皮书描述了公司如何使用来 AWS 创建符合 HIPAA 标准的应用程序。
- AWS 合规资源 — 此工作簿和指南集可能适用于您所在的行业和所在地。
- AWS Config— 该 AWS 服务评估您的资源配置在多大程度上符合内部实践、行业指导方针和法规。
- AWS Security Hub— 此 AWS 服务可全面了解您的安全状态 AWS ，帮助您检查是否符合安全行业标准和最佳实践。

Amazon Kinesis Data Analytics 故障恢复能力

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故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

除了 AWS 全球基础架构外，Kinesis Data Analytics 还提供多项功能来帮助支持您的数据弹性和备份需求。

灾难恢复

Kinesis Data Analytics 可在无服务器模式下运行，并通过执行自动迁移来处理主机降级、可用区可用性以及其他基础设施相关问题。当发生这种情况，Kinesis Data Analytics 可确保应用程序被处理，而不发生任何数据丢失。有关更多信息，请参阅将应用程序输出永久保存到外部目标的传输模型。

Kinesis Data Analytics for SQL Applications 中的基础设施安全

作为一项托管服务，Amazon Kinesis Data Analytics 受 AWS 亚马逊网络服务：安全流程概述白皮书中描述的全球网络安全程序的保护。

您可以使用 AWS 已发布的 API 调用通过网络访问 Kinesis Data Analytics。客户端必须支持传输层安全性（TLS）1.2 或更高版本。客户端还必须支持具有完全向前保密（PFS）的密码套件，例如 Ephemeral Diffie-Hellman（DHE）或 Elliptic Curve Ephemeral Diffie-Hellman（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 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

Kinesis Data Analytics 最佳安全实践

Amazon Kinesis Data Analytics 提供了许多安全功能，供您在开发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。这些最佳实践可能不适合环境或不满足环境要求，请将其视为有用的考虑因素而不是惯例。

使用 IAM 角色访问其他 Amazon 服务

您的 Kinesis Data Analytics 应用程序必须具有有效的凭证才能访问其他服务中的资源，例如 Kinesis 数据流、Firehose 交付流或 Amazon S3 存储桶。您不应将 AWS 证书直接存储在应用程序或 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 角色来管理访问其他资源的应用程序的临时凭证。在使用角色时，您不必使用长期凭证来访问其他资源。

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

实施从属资源中的服务器端加密

静态数据和传输中的数据在 Kinesis Data Analytics 中加密，并且无法禁用此加密。您应该在依赖资源（例如 Kinesis 数据流、Firehose 交付流和 Amazon S3 存储桶）中实现服务器端加密。有关在从属资源中实施服务器端加密的更多信息，请参阅 数据保护。

CloudTrail 用于监控 API 调用

Kinesis Data Analytics AWS CloudTrail与一项服务集成，该服务记录用户、角色或亚马逊服务在 Kinesis Data Analytics 中采取的操作。

使用收集的信息 CloudTrail，您可以确定向 Kinesis Data Analytics 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 AWS CloudTrail”。

监控 for SQL 应用程序

要保持 和 应用程序的可靠性、可用性和性能，监控是一个重要环节。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。不过，在开始监控 之前，您应制定一个监控计划并在计划中回答下列问题：

- 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，通过在不同时间和不同负载条件下测量性能，在您的环境中建立正常 性能的基准。在监控时，您可以存储历史监控数据。如果您这样做，则可以将历史监控数据与当前性能数据进行比较，确定性能的正常模式和性能异常，并找出解决问题的方法。

通过使用 ，您可以监控应用程序。该应用程序处理数据流（输入或输出），这两个数据流都包含标识符，您可以使用这些标识符来缩小对 CloudWatch 日志的搜索范围。有关 如何处理数据流的信息，请参阅 适用于 SQL 应用程序的 Amazon Kinesis Data Analytics：工作原理。

最重要的指标是 `millisBehindLatest`，表示应用程序读取流式传输源的滞后程度。通常情况下，滞后时间应当为零或接近零毫秒。通常会出现短暂峰值，`millisBehindLatest` 中会出现增长。

我们建议您设置一个 CloudWatch 警报，当应用程序在读取直播源时延迟超过一个小时时触发该警报。对于某些几乎要求实时处理的使用情况（例如，将已处理的数据发送到实时应用程序），您可以选择将警报设置为更低值，如 5 分钟。

主题

- 监控工具
- 使用 Amazon 进行监控 CloudWatch
- 使用 AWS CloudTrail 记录 API 调用

监控工具

AWS 提供了各种可用于监控的工具。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。

自动监控工具

您可以使用以下自动化监控工具来监控 并在出现错误时报告：

- 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。该操作是发送到亚马逊简单通知服务 (Amazon SNS) Simple Notification Scaling 主题或亚马逊 EC2 Auto Scaling 策略的通知。 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。
- Amazon CloudWatch Logs — 监控、存储和访问来自 AWS CloudTrail 或其他来源的日志文件。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的监控日志文件。
- Amazon CloudWatch Events — 匹配事件并将其路由到一个或多个目标函数或流，以进行更改、捕获状态信息并采取纠正措施。有关更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的什么是亚马逊 CloudWatch 活动。
- AWS CloudTrail 日志监控-在账户之间共享日志文件，通过将 CloudTrail 日志文件发送到 “日志” 来实时监控CloudWatch 日志文件，用 Java 编写日志处理应用程序，并验证您的日志文件在传送后是否未更改 CloudTrail。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的 “使用CloudTrail 日志文件”。

手动监控工具

监控的另一个重要部分是手动监控 CloudWatch 警报未涵盖的项目。、 CloudWatch Trusted Advisor、和其他 AWS Management Console 仪表板提供了 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
  - 搜索并浏览您的所有指标
  - 创建和编辑告警以接收问题通知
- AWS Trusted Advisor 可以帮助您监控以提高性能、可靠性、安全性和成本效益。所有用户可以使用 4 项 Trusted Advisor 检查。具有商业或企业支持计划的用户可以使用超过 50 个检查。有关更多信息，请参阅AWS Trusted Advisor。

使用 Amazon 进行监控 CloudWatch

您可以使用 Amazon 监控应用程序 CloudWatch。 CloudWatch 收集原始数据并将其处理为可读的、近乎实时的指标。这些统计数据会保存两周。这样您能够访问历史信息，并能够更好地了解您的 Web 应用程序或服务的执行情况。默认情况下，指标数据会自动发送到 CloudWatch。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

- 指标与维度
- 查看 指标和维度
- 创建要监控的 CloudWatch 警报
- 使用 Amazon CloudWatch 日志

指标与维度

AWS/KinesisAnalytics 命名空间包括以下指标。

指标	描述
Bytes	读取（每个输入流）或写入（每个输出流）的字节数。 级别：每个输入流和每个输出流
KPUs	用于运行您的流式处理应用程序的 Kinesis 处理单元数量。每小时所用的 KPU 平均数量决定了您的应用程序所需的费用。

指标	描述
	级别：应用程序级
MillisBehindLatest	表示应用程序读取流式源的时间相对于当前时间的延迟。 级别：应用程序级
Records	读取（每个输入流）或写入（每个输出流）的记录数。 级别：每个输入流和每个输出流
Success	1 表示到为您的应用程序配置的目的地的每个成功交付尝试；0 表示每个失败交付尝试。该指标的平均值表示执行了多少成功的交付。 级别：按目的地。
InputProcessing.Duration	执行的每次 AWS Lambda 函数调用所花费的时间。 级别：每个输入流
InputProcessing.OkRecords	Lambda 函数返回的标有 Ok 状态的记录的数量。 级别：每个输入流
InputProcessing.OkBytes	Lambda 函数返回的标有 Ok 状态的记录的字节总数。 级别：每个输入流
InputProcessing.DroppedRecords	Lambda 函数返回的标有 Dropped 状态的记录的数量。 级别：每个输入流
InputProcessing.ProcessingFailedRecords	Lambda 函数返回的标有 ProcessingFailed 状态的记录的数量。 级别：每个输入流

指标	描述
InputProcessing.Success	执行的成功 Lambda 调用的数量。 级别：每个输入流
LambdaDelivery.OkRecords	Lambda 函数返回的标有 Ok 状态的记录的数量。 级别：按 Lambda 目的地
LambdaDelivery.DeliveryFailedRecords	Lambda 函数返回的标有 DeliveryFailed 状态的记录的数量。 级别：按 Lambda 目的地
LambdaDelivery.Duration	为 所执行的每个 Lambda 函数调用花费的时间。 级别：按 Lambda 目的地

为以下维度提供指标。

维度	描述
Flow	每个输入流：输入 每个输出流：输出
Id	每个输入流：输入 ID 每个输出流：输出 ID

查看 指标和维度

当您的应用程序处理数据流时，会将以下指标和维度发送到 CloudWatch。您可以使用以下流程查看 的指标

在控制台中，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。

使用 CloudWatch 控制台查看指标

1. 打开 CloudWatch 控制台，网址为 https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择指标。
3. 在的 “按类别划分的CloudWatch 指标” 窗格中，选择一个指标类别。
4. 在上方窗格中，滚动以查看完整指标列表。

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/KinesisAnalytics" --region region
```

指标是在以下级别收集的：

- 应用程序
- 输入流
- 输出流

创建要监控的 CloudWatch 警报

您可以创建一个 Amazon CloudWatch 警报，当警报状态发生变化时，该警报会发送 Amazon SNS 消息。警报会在您规定的时间范围内监控某一项指标。它在多个时间段内根据相对于给定阈值的指标值，执行一项或多项操作。操作是一个发送到 Amazon SNS 主题或 Auto Scaling 策略的通知。

告警仅为持续状态更改调用操作。要使 CloudWatch 警报调用操作，状态必须已更改并维持了指定的时间。

您可以使用 AWS Management Console、 CloudWatch AWS CLI或 CloudWatch API 设置警报，如下所述。

使用 CloudWatch 控制台设置警报

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为 https://console.aws.amazon.com/cloudwatch/。
2. 选择创建警报。Create Alarm Wizard (创建警报向导) 启动。

3. 选择 Kinesis Analytics Metrics (Kinesis Analytics 指标)。然后，滚动 指标找到您想要设置报警器的指标的位置。

   要仅显示 指标，请搜索文件系统的文件系统 ID。选择要为其创建警报的指标，然后选择 Next (下一步)。

4. 输入指标的 Name (名称)、Description (描述) 和 Whenever (每当) 值。

5. 如果 CloudWatch 要在达到警报状态时向您发送电子邮件，请在 “每当此警报：” 字段中，选择 “状态为警报”。在 Send notification to (发送通知到) 字段中，选择一个现有 SNS 主题。如果您选择创建主题，那么您就可以为新电子邮件订阅列表设置名称和电子邮件地址。此列表将保存下来并会在将来的警报字段中显示出来。

   > **Note**
   >
   > 如果您使用 Create topic (创建主题) 创建一个新 Amazon SNS 主题，那么电子邮件地址在接收通知之前必须通过验证。当警报进入警报状态时，才会发送电子邮件。如果在验证电子邮件地址之前警报状态发生了变化，那么它们不会接收到通知。

6. 在 Alarm Preview 部分中，预览您即将创建的警报。

7. 选择 Create Alarm 以创建警报。

使用 CloudWatch CLI 设置警报

- 调用 `mon-put-metric-alarm`。有关更多信息，请参阅 Amazon CloudWatch CLI 参考。

使用 CloudWatch API 设置警报

- 调用 `PutMetricAlarm`。有关更多信息，请参阅 Amazon CloudWatch API 参考。

使用 Amazon CloudWatch 日志

如果未正确配置 应用程序，它可能会在应用程序启动期间转变为运行状态。或者它可以更新，但不会处理进入应用程序内部输入流的任何数据。通过向应用程序添加 CloudWatch 日志选项，您可以监控应用程序配置问题。

可能会在以下情况下产生配置错误：

- 用于输入的 Kinesis 数据流不存在。

- 用于输入的 Amazon Data Firehose 传输流不存在。
- 用作引用数据源的 Amazon S3 存储桶不存在。
- S3 存储桶的引用数据源中的指定文件不存在。
- 管理相关权限的 AWS Identity and Access Management (IAM) 角色中未定义正确的资源。
- 管理相关权限的 IAM 角色中未定义正确权限。
- 无权担任管理相关权限的 IAM 角色。

有关亚马逊的更多信息 CloudWatch，请参阅亚马逊 CloudWatch 用户指南。

添加 PutLogEvents 策略操作

需要权限才能将错误配置错误写入。 CloudWatch您可以将这些权限添加到 担任的 IAM 角色中，如下所述。有关将 IAM 角色用于 的更多信息，请参阅 Kinesis Data Analytics 中的身份和访问管理。

信任策略

要为 授予权限以担任 IAM 角色，您可以将以下信任策略附加到该角色。

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

权限策略

要向应用程序授予 CloudWatch 从资源写入日志事件的权限，您可以使用以下 IAM 权限策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt0123456789000",
```

```
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:my-log-group:log-
stream:my-log-stream*"
            ]
        }
    ]
}
```

添加配置错误监控

使用以下 API 操作向新应用程序或现有应用程序添加 CloudWatch 日志选项或更改现有应用程序的日志选项。

> **Note**
>
> 目前，您只能使用 API 操作向应用程序添加 CloudWatch 日志选项。您无法使用控制台添加 CloudWatch 日志选项。

创建应用程序时添加 CloudWatch 日志选项

以下代码示例演示了在创建应用程序时如何使用`CreateApplication`操作添加 CloudWatch 日志选项。有关 `Create_Application` 的更多信息，请参阅 CreateApplication。

```
{
   "ApplicationCode": "<The SQL code the new application will run on the input
 stream>",
   "ApplicationDescription": "<A friendly description for the new application>",
   "ApplicationName": "<The name for the new application>",
   "Inputs": [ ... ],
   "Outputs": [ ... ],
   "CloudWatchLoggingOptions": [{
      "LogStreamARN": "<Amazon Resource Name (ARN) of the CloudWatch log stream to add
 to the new application>",
      "RoleARN": "<ARN of the role to use to access the log>"
 }]
}
```

向现有应用程序添加 CloudWatch 日志选项

以下代码示例说明了如何使用 AddApplicationCloudWatchLoggingOption 操作将 CloudWatch 日志选项添加到现有应用程序中。有关 AddApplicationCloudWatchLoggingOption 的更多信息，请参阅AddApplicationCloudWatchLoggingOption。

```
{
   "ApplicationName": "<Name of the application to add the log option to>",
   "CloudWatchLoggingOption": {
      "LogStreamARN": "<ARN of the log stream to add to the application>",
      "RoleARN": "<ARN of the role to use to access the log>"
   },
   "CurrentApplicationVersionId": <Version of the application to add the log to>
}
```

更新现有 CloudWatch 日志选项

以下代码示例演示如何使用UpdateApplication操作修改现有 CloudWatch 日志选项。有关 UpdateApplication 的更多信息，请参阅UpdateApplication。

```
{
   "ApplicationName": "<Name of the application to update the log option for>",
   "ApplicationUpdate": {
      "CloudWatchLoggingOptionUpdates": [
         {
            "CloudWatchLoggingOptionId": "<ID of the logging option to modify>",
            "LogStreamARNUpdate": "<ARN of the new log stream to use>",
            "RoleARNUpdate": "<ARN of the new role to use to access the log stream>"
         }
      ],
   },
   "CurrentApplicationVersionId": <ID of the application version to modify>
}
```

从应用程序中删除 CloudWatch 日志选项

以下代码示例演示如何使用DeleteApplicationCloudWatchLoggingOption操作删除现有 CloudWatch 日志选项。有关 DeleteApplicationCloudWatchLoggingOption 的更多信息，请参阅DeleteApplicationCloudWatchLoggingOption。

```
{
   "ApplicationName": "<Name of application to delete log option from>",
   "CloudWatchLoggingOptionId": "<ID of the application log option to delete>",
   "CurrentApplicationVersionId": <Version of the application to delete the log option
 from>
}
```

配置错误

以下各节详细介绍了您可能在 Amazon L CloudWatch ogs 中看到的因应用程序配置错误而出现的错误。

错误消息格式

由应用程序错误配置产生的错误消息将采用以下格式显示。

```
{
"applicationARN": "string",
"applicationVersionId": integer,
"messageType": "ERROR",
"message": "string",
"inputId": "string",
"referenceId": "string",
"errorCode": "string"
"messageSchemaVersion": "integer",
}
```

错误消息中的字段包含以下信息：

- applicationARN：生成应用程序的 Amazon 资源名称 (ARN)，例如：arn:aws:kinesisanalytics:us-east-1:112233445566:application/sampleApp
- applicationVersionId：遇到错误时的应用程序版本。有关更多信息，请参阅ApplicationDetail。
- messageType：消息类型。目前，此类型只能是 ERROR。
- message：错误的详细信息，例如：

```
There is a problem related to the configuration of your input. Please check that the
 resource exists, the role has the correct permissions to access the resource and
 that Kinesis Analytics can assume the role provided.
```

- `inputId`：与应用程序输入关联的 ID。仅当此输入为错误的原因时，此值才存在。如果 `referenceId` 存在，则此值不会存在。有关更多信息，请参阅DescribeApplication。
- `referenceId`：与应用程序引用数据源关联的 ID。仅当此源为错误的原因时，此值才会存在。如果 `inputId` 存在，则此值不会存在。有关更多信息，请参阅DescribeApplication。
- `errorCode`：错误的标识符。此 ID 为 `InputError` 或 `ReferenceDataError`。
- `messageSchemaVersion`：指定最新消息架构版本的值，当前为 1。您可以检查此值以了解错误消息架构是否已更新。

错误

CloudWatch 日志中可能出现的错误包括以下内容。

资源不存在

如果为 Kinesis 输入流指定的 ARN 不存在，但 ARN 在语法上正确，则会产生类似于下面的错误。

```
{
"applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
"applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your input. Please
 check that the resource exists, the role has the correct permissions to access the
 resource and that Kinesis Analytics can assume the role provided.",
 "inputId":"1.1",
 "errorCode": "InputError",
 "messageSchemaVersion": "1"
}
```

如果用于引用数据的 Amazon S3 文件键不正确，则会产生类似于下面的错误。

```
{
 "applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
 "applicationVersionId": "5",
```

```
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your reference data.
 Please check that the bucket and the file exist, the role has the correct permissions
 to access these resources and that Kinesis Analytics can assume the role provided.",
 "referenceId":"1.1",
 "errorCode": "ReferenceDataError",
 "messageSchemaVersion": "1"
}
```

角色不存在

如果为不存在的 IAM 输入角色指定了 ARN，但 ARN 在语法上正确，则会产生类似于下面的错误。

```
{
 "applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
 "applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your input. Please
 check that the resource exists, the role has the correct permissions to access the
 resource and that Kinesis Analytics can assume the role provided.",
 "inputId":null,
 "errorCode": "InputError",
 "messageSchemaVersion": "1"
}
```

角色无权访问资源

如果使用的输入角色无权访问输入资源（如 Kinesis 源流），则会产生类似于下面的错误。

```
{
 "applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
 "applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your input. Please
 check that the resource exists, the role has the correct permissions to access the
 resource and that Kinesis Analytics can assume the role provided.",
 "inputId":null,
 "errorCode": "InputError",
 "messageSchemaVersion": "1"
}
```

使用 AWS CloudTrail 记录 API 调用

与 AWS CloudTrail 集成，后者是在 中记录用户、角色或 AWS 服务所执行操作的服务。CloudTrail 将的所有 API 调用作为事件捕获。捕获的调用包含来自 控制台和代码的 API 操作调用。如果您创建跟踪，则可以使 CloudTrail 事件持续传送到 Amazon S3 存储桶（包括 的事件）。如果您不配置跟踪记录，则仍可在 CloudTrail 控制台中的 Event history（事件历史记录）中查看最新事件。使用 CloudTrail 收集的信息，您可以确定向 发出了什么请求、发出请求的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息，请参阅《AWS CloudTrail 用户指南》。

CloudTrail 中的 信息

在您创建 AWS 账户时，将在该账户上启用 CloudTrail。当 中发生活动时，该活动将记录在 CloudTrail 事件中，并与其它 AWS 服务事件一同保存在 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 CloudTrail 事件历史记录查看事件。

要持续记录 AWS 账户中的事件（包括 的事件），请创建跟踪。通过跟踪记录，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有。此跟踪记录在 AWS 分区中记录所有区域中的事件，并将日志文件传送至您指定的 Simple Storage Service（Amazon S3）桶。此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关更多信息，请参阅下列内容：

- 创建跟踪概览
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件

CloudTrail 记录所有操作，API 参考中介绍了这些操作。例如，对 `CreateApplication` 和 `UpdateApplication` 操作的调用会在 CloudTrail 日志文件中生成条目。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用 AWS 账户根用户 还是用户凭证发出的。
- 请求是使用角色还是联合身份用户的临时安全凭证发出的。
- 请求是否由其它 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

了解 日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 桶。CloudTrail 日志文件包含一个或多个日志条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

以下示例显示了一个 CloudTrail 日志条目，它说明了 AddApplicationCloudWatchLoggingOption 和 DescribeApplication 操作。

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2019-03-14T01:03:00Z",
            "eventSource": "kinesisanalytics.amazonaws.com",
            "eventName": "AddApplicationCloudWatchLoggingOption",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "currentApplicationVersionId": 1,
                "cloudWatchLoggingOption": {
                    "roleARN": "arn:aws:iam::012345678910:role/cloudtrail_test",
                    "logStreamARN": "arn:aws:logs:us-east-1:012345678910:log-
group:cloudtrail-test:log-stream:sql-cloudwatch"
                },
                "applicationName": "cloudtrail-test"
            },
            "responseElements": null,
            "requestID": "e897cd34-45f4-11e9-8912-e52573a36cd9",
            "eventID": "57fe50e9-c764-47c3-a0aa-d0c271fa1cbb",
            "eventType": "AwsApiCall",
```

```
            "apiVersion": "2015-08-14",
            "recipientAccountId": "303967445486"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2019-03-14T05:37:20Z",
            "eventSource": "kinesisanalytics.amazonaws.com",
            "eventName": "DescribeApplication",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "applicationName": "cloudtrail-test"
            },
            "responseElements": null,
            "requestID": "3b74eb29-461b-11e9-a645-fb677e53d147",
            "eventID": "750d0def-17b6-4c20-ba45-06d9d45e87ee",
            "eventType": "AwsApiCall",
            "apiVersion": "2015-08-14",
            "recipientAccountId": "012345678910"
        }
    ]
}
```

Limits

在使用 Amazon Kinesis Data Analytics for SQL 应用程序时，请注意以下限制：

- Kinesis Data Analytics for SQL 可在以下 AWS 地区使用：美国东部（弗吉尼亚州北部）、美国东部（俄亥俄州）、美国西部（俄勒冈州）、加拿大（中部）、欧洲地区（巴黎）、欧洲地区（爱尔兰）、欧洲地区（法兰克福）、欧洲地区（伦敦）、亚太地区（香港）、亚太地区（孟买）、亚太地区（悉尼）、亚太地区（新加坡）、亚太地区（首尔）、亚太地区（东京）、南美洲（圣保罗）、AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）。我们未计划在 AWS 其他区域发布 Kinesis Data Analytics for SQL。
- 2023 年 6 月 28 日之后，如果您尚未使用 Kinesis Data Analytics for SQL，则将无法使用 AWS 管理控制台创建新的 Kinesis Data Analytics for SQL 应用程序。如果您在 2023 年 6 月 28 日之前创建了 Kinesis Data Analytics for SQL 应用程序，那么在您已经使用 Kinesis Data Analytics for SQL 的 AWS 区域，您当前创建和运行应用程序的方式不会发生变化。但是，在您未使用 Kinesis Data Analytics for SQL 的区域，您将不能再使用 AWS 控制台创建新的应用程序。
- 2023 年 9 月 12 日之后，如果您尚未使用适用于 SQL 的 Kinesis Data Analytics，则将无法使用 Kinesis Data Firehose 作为来源创建新应用程序。使用 `KinesisFirehoseInput` 对 Kinesis Data Analytics for SQL 应用程序进行操作的现有客户可以继续使用 `KinesisFirehoseInput` 在使用 Kinesis Data Analytics 的现有账户内添加应用程序。如果您是现有客户，并希望使用 `KinesisFirehoseInput` 在 Kinesis Data Analytics for SQL 应用程序中创建新账户，则可以打开支持案例。有关更多信息，请参阅 AWS Support中心。
- 应用程序内部流中的单个行大小限制为 512 KB。Kinesis Data Analytics 最多使用 1 KB 来存储元数据。此元数据计入行限制。
- 应用程序中的 SQL 代码限制为 100 KB。
- 对于窗口化查询，我们建议的最长时间为 1 小时。应用程序内流存储在易失性存储中，在出现意外的应用程序中断时，会导致应用程序从易失性存储中的源数据重建流。
- 对于单个应用程序内流，我们建议的最大吞吐量为 2 到 20 MB/ 秒，具体取决于应用程序查询的复杂性。

- 在您的账户的每个 AWS 区域中，您最多可以创建 50 个 Kinesis Data Analytics 应用程序。可以创建一个案例，通过服务限制增加表来申请其他应用程序。有关更多信息，请参阅 AWS Support中心。

- 适用于 SQL 的 Kinesis Data Analytics 单个应用程序可以处理的最大流式处理吞吐量约为 100 MB/秒。这假设您已将应用程序内部流的数量增加到最大值 64，并且您已将 KPU 限制增加到 8 以上（有关详细信息，请参阅以下限制）。如果您的应用程序需要处理超过 100 MB /秒的输入，请执行以下操作之一：
  - 使用多个 Kinesis Data Analytics for SQL 应用程序处理输入
  - 如果您希望继续使用单个流和应用程序，请使用适用于 Java 的 Managed Service for Apache Flink 应用程序。

  > **Note**
  >
  > 建议您定期检查应用程序的 `InputProcessing.0kBytes` 指标，以便您可以规划使用多个 SQL 应用程序，或者如果应用程序的预计输入吞吐量将超过 100 MB/秒，则可以迁移到适用于 Java 的 Managed Service for Apache Flink。此外，还建议您在 `InputProcessing.0kBytes` 上创建 CloudWatch 告警，以便在应用程序接近输入吞吐量限制时收到通知。您可以通过这种方式更新应用程序查询，从而获得更高的吞吐量，避免分析时的反向压力和延迟。有关更多信息，请参阅故障排除。警报也适用于具备上游吞吐量降低机制这一情况。

- Kinesis 处理单元 (KPU) 数限制为 8 个。有关申请提高此限制的说明，请参阅 Amazon 服务限制中的申请提高限制。

  使用 Kinesis Data Analytics，您可以按实际用量付费。将根据运行流处理应用程序所使用的 KPU 平均数量来按小时费率计费。一个 KPU 可为您提供 1 个 vCPU 和 4 GB 内存。

- 每个应用程序可以具有一个流式传输源和最多一个引用数据源。

- 您最多可以为 Kinesis Data Analytics 应用程序配置 3 个目标。建议您使用这些目标中的一个来永久保存应用程序内部错误流数据。

- 存储引用数据的 Amazon S3 对象的大小最多为 1 GB。

- 如果在将 S3 存储桶中存储的引用数据上传到应用程序内部表后更改此数据，您需要使用 UpdateApplication 操作（使用 API 或 AWS CLI）以在应用程序内部表中刷新数据。目前，AWS Management Console在应用程序中不支持刷新引用数据。

- 目前，Kinesis Data Analytics 不支持 Amazon Kinesis 创建器库 (KPL) 生成的数据。

- 您可以为每个应用程序分配最多 50 个标签。

最佳实操

此部分介绍了使用 Amazon Kinesis Data Analytics 应用程序时的最佳实践。

主题

管理应用程序

在管理 Amazon Kinesis Data Analytics 应用程序时，请遵循以下最佳实践：

- 设置亚马逊 CloudWatch 警报 — 您可以使用Kinesis Data Analytics提供的 CloudWatch 指标来监控以下内容：
  - 输入字节和输入记录（输入应用程序的字节和记录的数目）
  - 输出字节和输出记录
  - `MillisBehindLatest`（从流式传输源进行读取时的应用程序滞后）

  我们建议您针对生产中的应用程序的以下指标设置至少两个 CloudWatch 警报：
  - `MillisBehindLatest` – 大多数情况下，建议您将此警报设置为在应用程序滞后于最新数据 1 小时的情况下触发（平均每分钟触发 1 次）。对于 end-to-end 处理需求较低的应用程序，您可以将其调整为较低的容差。此警报可帮助确保应用程序读取最新数据。

- 为避免出现 `ReadProvisionedThroughputException` 异常，请将从同一 Kinesis 数据流中读取的生产应用程序数限制为 2 个应用程序。

> **Note**
>
> 在这种情况下，应用程序 指可从流式传输源读取的任何应用程序。只有 Kinesis Data Analytics 应用程序可以从 Firehose 传输流中读取数据。但是，许多应用程序可以从 Kinesis 数据流中读取数据，例如 Kinesis Data Analytics 应用程序或。 AWS Lambda建议的应用程序限制指的是配置为从流式传输源读取的总应用程序数。

Amazon Kinesis Data Analytics 大约每秒为每个应用程序读取一次流式传输源。不过，滞后的应用程序可以更快的速率读取数据以便保持一致。要允许应用程序的足够吞吐量以便保持一致，请限制读取同一数据源的应用程序的数目。

- 将从同一 Firehose 交付流中读取的生产应用程序数量限制为一个应用程序。

  Firehose 传输流可以写入到亚马逊 S3 和亚马逊 Redshift 等目的地。它还可以作为 Kinesis Data Analytics 应用程序的流式传输源。因此，我们建议您不要为每个 Firehose 交付流配置多个 Kinesis Data Analytics 应用程序。这有助于确保传输流还可以传输到其他目标。

扩展应用程序

通过从默认值（一）开始主动增加应用程序内输入流的数量，设置应用程序以满足您未来的扩展需求。我们建议根据应用程序的吞吐量选择以下语言：

- 如果您的应用程序的扩展需求超过 100 MB/秒，请使用多个流和适用于 SQL 应用程序的 Kinesis Data Analytics。
- 如果您想使用单个流和应用程序，请使用适用于 Apache Flink 的托管服务应用程序。

> **Note**
>
> 建议您定期检查应用程序的 `InputProcessing.OkBytes` 指标，以便您可以规划使用多个 SQL 应用程序，或者如果应用程序的预计输入吞吐量将超过 100 MB/秒，则可以迁移到 managed-flink/latest/java/。

监控应用程序

我们建议您创建 CloudWatch 警报，`InputProcessing.0kBytes`以便在应用程序接近输入吞吐量限制时收到通知。您可以通过这种方式更新应用程序查询，从而获得更高的吞吐量，避免分析时的反向压力和延迟。有关更多信息，请参阅故障排除。这种方式也适用于具备上游吞吐量降低机制这一情况。

- 对于单个应用程序内流，我们建议的最大吞吐量为 2 到 20 MB/ 秒，具体取决于应用程序查询的复杂性。
- 适用于 SQL 的 Kinesis Data Analytics 单个应用程序可以处理的最大流式处理吞吐量约为 100 MB/秒。假设您已将应用程序内部流的数量增加到最大值 64，并且您的 KPU 限制已超过 8。有关更多信息，请参阅限制。

> **Note**
>
> 建议您定期检查应用程序的 `InputProcessing.0kBytes` 指标，以便您可以规划使用多个 SQL 应用程序，或者如果应用程序的预计输入吞吐量将超过 100 MB/秒，则可以迁移到 managed-flink/latest/java/。

定义输入架构

在控制台中配置应用程序输入时，您首先指定流式传输源。随后，控制台将使用发现 API（请参阅 DiscoverInputSchema）对流式传输源上的记录采样来推断架构。此外，架构在产生的应用程序内部流中定义列的名称和数据类型。控制台将显示架构。建议您对此推断的架构执行以下操作：

- 充分测试推断的架构。发现过程仅使用流式传输源上的记录示例来推断架构。如果您的流式传输源有多种记录类型，则发现 API 可能错过了一种或多种记录类型的采样。这种情况会导致架构不能准确反映流式传输源上的数据。

  当您的应用程序启动时，这些丢失的记录类型可能会导致解析错误。Amazon Kinesis Data Analytics 将这些记录发送到应用程序内部错误流。要减少这些解析错误，建议您在控制台中以交互方式测试推断的架构，并监控应用程序内部流是否缺少记录。

- Kinesis Data Analytics API 不支持在输入配置中指定列的 `NOT NULL` 约束。如果您希望在应用程序内部流中指定列的 `NOT NULL` 约束，请使用应用程序代码创建这些应用程序内部流。随后，您可以将数据从一个应用程序内部流复制到另一个应用程序内部流，之后将强制实施该约束。

在需要值时尝试在行中插入 NULL 值会导致出现错误。Kinesis Data Analytics 会将这些错误发送到应用程序内部错误流。

- 放宽发现过程所推断的数据类型。发现过程将根据流式传输源中记录的随机采样来建议列和数据类型。建议您仔细审查这些列和数据类型，并考虑放宽这些数据类型以涵盖输入中所有可能的记录情况。这可确保应用程序在运行时出现的分析错误更少。例如，如果推断的架构具有 SMALLINT 作为列类型，则可考虑将列类型更改为 INTEGER。

- 在应用程序代码中使用 SQL 函数来处理任何非结构化的数据或列。您的输入中可包含非结构化的数据或列（例如，日志数据）。有关示例，请参阅示例：转换 DateTime 值。处理此类数据的一种方法是，定义仅具有一个类型 VARCHAR(N) 的列的架构，其中 N 是您应在流中看到的可能最大的行。随后，可在您的应用程序代码中读取传入记录，并使用 String 和 Date Time 函数来解析和架构化原始数据。

- 确保您能够完全处理包含两个级别以上的嵌套的流式传输源数据。当源数据为 JSON 时，您可以使用嵌套。发现 API 会推断可展平一个嵌套层的架构。对于两层嵌套，发现 API 也将尝试展平嵌套。在超出两层嵌套的情况下，对展平的支持是有限的。要完全处理嵌套，您必须手动修改推断的架构来满足您的需求。可使用下列任一策略执行此操作：

  - 使用 JSON 行路径可选择性地只提取应用程序所需的键值对。JSON 行路径为要引入应用程序中的特定键值对提供了一个指针。您可为任何级别的嵌套执行此操作。
  - 使用 JSON 行路径可选择性地提取复杂的 JSON 对象，然后在应用程序代码中使用字符串处理函数提取所需的特定数据。

连接到输出

建议每个应用程序具有至少两个输出：

- 使用第一个目标插入 SQL 查询的结果。
- 使用第二个目标插入整个错误流，并通过 Firehose 传输流将其发送到 S3 存储桶。

创作应用程序代码

我们建议执行下列操作：

- 在 SQL 语句中，不要指定超过 1 个小时的基于时间的窗口，原因如下：
  - 有时，需要重新启动应用程序，这是因为您更新了应用程序或出于 Kinesis Data Analytics 内部原因。在重新启动时，将从流式数据源中重新读取窗口中包含的所有数据。这需要等待一段时间，然后 Kinesis Data Analytics 才能发送该窗口的输出。
  - Kinesis Data Analytics 必须将与应用程序状态相关的所有内容（包括相关数据）保留一段时间。这会消耗大量的 Kinesis Data Analytics 处理单元。
- 在开发期间，在 SQL 语句中设置较小的窗口以便更快地查看结果。当您将应用程序部署到生产环境时，可以设置适当的窗口大小。
- 不要使用单个复杂的 SQL 语句，而是在将结果保存到中间应用程序内部流的每个步骤中将此语句分成多个语句。这可帮助您加快调试速度。
- 在您使用滚动窗口时，建议您使用两个窗口，一个用于处理时间，另一个用于逻辑时间（接收时间或事件时间）。有关更多信息，请参阅时间戳和 ROWTIME 列。

测试应用程序

在更改 Kinesis Data Analytics 应用程序的架构或应用程序代码时，我们建议使用测试应用程序验证您的更改，然后再将其部署到生产环境中。

设置测试应用程序

您可以通过控制台或使用 AWS CloudFormation 模板设置测试应用程序。使用 AWS CloudFormation 模板有助于确保您对测试应用程序和实时应用程序所做的代码更改保持一致。

设置测试应用程序时，您可以将应用程序连接到活动数据，或者使用模拟数据来填充流以进行测试。建议通过两种方法来使用模拟数据填充流：

- 使用 Kinesis Data Generator (KDG)。KDG 使用数据模板将随机数据发送到 Kinesis 流。KDG 易于使用，但不适合测试数据项之间的复杂关系，例如检测数据热点或异常的应用程序。
- 使用自定义 Python 应用程序以将更复杂的数据发送到 Kinesis 数据流。Python 应用程序可以生成数据项之间的复杂关系，例如热点或异常。有关将数据集群化发送到数据热点的 Python 应用程序示例，请参阅示例：检测流上的热点 (HOTSPOTS 函数)。

运行测试应用程序时，请使用目标（例如到 Amazon Redshift 数据库的 Firehose 传输流）查看结果，而不是在控制台上查看应用程序内流。控制台上显示的数据是流的采样，并未包含所有记录。

测试架构更改

更改应用程序的输入流架构时，使用测试应用程序来验证以下情况：

- 来自您的流中的数据强制转换为正确的数据类型。例如，确保日期时间数据未作为字符串接收到应用程序中。
- 进行解析的数据强制转换为您需要的数据类型。如果出现解析或强制转换错误，您可以在控制台上查看，或者为错误流分配目标并在目标存储中查看错误。
- 字符数据的数据字段具有足够的长度，并且应用程序未截断字符数据。您可以在目标存储中查看数据记录，验证未截断您的应用程序数据。

测试代码更改

测试对 SQL 代码的更改时，需要了解关于您应用程序的领域的一些知识。您必须确定需要能够测试哪些输出以及正确的输出应该是什么。有关在修改应用程序的 SQL 代码时可能需要验证的问题领域，请参阅适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 故障排除。

适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 故障排除

以下内容可以帮助您解决在适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 中可能遇到的问题。

主题

已关停的应用程序

- 什么是已关停的适用于 SQL 的 Kinesis Data Analytics 应用程序？

  已关停的应用程序是至少三个月未处理任何记录的应用程序。即使客户没有使用适用于 SQL 的 Kinesis Data Analytics 资源，但仍在为其支付费用。
- AWS 什么时候开始关停空闲的应用程序？

  AWS 将在 2023 年 11 月 14 日至 2023 年 11 月 21 日期间关停空闲的应用程序。我们将在所在区域的工作时间段关停空闲的应用程序。
- 是否可以重新启动已关停的适用于 SQL 的 Kinesis Data Analytics 应用程序？

  是。您可以在需要时照常重新启动应用程序。无需创建支持票证。
- AWS 停用空闲应用程序后，我的查询结果是否会随之删除？

  不会。首先，由于您的应用程序处于空闲状态，因此无法对查询进行处理。其次，您的查询结果不会存储在适用于 SQL 的 Kinesis Data Analytics 中。您可以为适用于 SQL 的 Kinesis Data Analytics 应

用程序配置接收器 (目的地)，例如，在 Amazon S3 或其他数据流中。因此，您保留对数据的完整所有权，并且根据该存储服务的条款，这些数据仍可被检索。

- 我不希望应用程序被停用，我该怎么做？

  在 2023 年 11 月 10 日之前任何时候，你可以发送电子邮件给服务团队 (kda-sql-questions@amazon.com)，要求不停用您的应用程序。电子邮件中应包含您的账户 ID 和应用程序 ARN。

无法运行 SQL 代码

如果需要了解如何使特定 SQL 语句正常工作，可以在使用 Kinesis Data Analytics 时利用几个不同的资源：

- 有关 SQL 语句的更多信息，请参阅适用于 SQL 的 Kinesis Data Analytics 示例。此部分提供了一些可使用的 SQL 示例。
- Amazon Kinesis Data Analytics SQL 参考提供了编写流式 SQL 语句的详细指南。
- 如果仍遇到问题，我们建议您在 Kinesis Data Analytics 论坛上提问。

无法检测到或发现我的架构

在某些情况下，Kinesis Data Analytics 无法检测或发现架构。在很多此类情况下，您仍然可以使用 Kinesis Data Analytics。

假设您具有不使用分隔符的 UTF-8 编码数据，或具有使用非逗号分隔值 (CSV) 格式的数据，或发现 API 未发现您的架构。在这些情况下，可以手动定义架构或使用字符串操作函数来构建数据。

为了发现您的数据流的架构，Kinesis Data Analytics 会随机对流中的最新数据进行采样。如果您无法始终如一地向您的流发送数据，Kinesis Data Analytics 可能无法检索样本和检测架构。有关更多信息，请参阅针对流数据使用架构发现功能。

引用数据已过时

在启动或更新应用程序时或在服务问题导致的应用程序中断期间，引用数据将从 Amazon Simple Storage Service (Amazon S3) 对象加载到应用程序中。

在更新基础 Amazon S3 对象时，不会将引用数据加载到应用程序中。

如果应用程序中的引用数据不是最新的，可以通过以下步骤重新加载这些数据：

1. 在 Kinesis Data Analytics 控制台上，在列表中选择应用程序名称，然后选择应用程序详细信息。
2. 选择 Go to SQL editor (转到 SQL 编辑器) 打开应用程序的 Real-time analytics (实时分析) 页面。
3. 在 Source Data (源数据) 视图中，选择引用数据表名称。
4. 依次选择 Actions (操作)、Synchronize reference data table (同步引用数据表)。

应用程序不写入到目标

如果数据未被写入到目标，请检查以下各项：

- 验证应用程序的角色具有足够权限来访问目标。有关更多信息，请参阅 写入到 Kinesis 流的权限策略 或 写入到 Firehose 传输流的权限策略。
- 验证是否已正确配置应用程序目标，并且应用程序正在使用正确的输出流名称。
- 检查输出流的 Amazon CloudWatch 指标以查看是否正在写入数据。有关使用 CloudWatch 指标的信息，请参阅使用 Amazon 进行监控 CloudWatch。
- 使用 the section called “AddApplicationCloudWatchLoggingOption” 添加 CloudWatch 日志流。您的应用程序将配置错误写入日志流。

如果该角色和目标配置看起来正确，请尝试重启应用程序，同时为 `LAST_STOPPED_POINT` 指定 InputStartingPositionConfiguration。

要监控的重要应用程序运行状况参数

要确保您的应用程序正常运行，我们建议您监控某些重要参数。

要监控的最重要的参数是 Amazon CloudWatch 指标 `MillisBehindLatest`。此指标表示落后于您从流中读取的当前时间多久。此指标可帮助确定您是否足够快速地处理源流中的记录。

一般来说，应将 CloudWatch 警报设置为在滞后超过 1 小时时触发。但是，时间量取决于您的使用案例。您可以按需进行调整。

有关更多信息，请参阅最佳实操。

在运行应用程序时出现无效代码错误

如果无法保存和运行 Amazon Kinesis Data Analytics 应用程序的 SQL 代码，常见原因如下：

- 在 SQL 代码中重新定义了流 – 在创建流和与流关联的数据泵后，不能在代码中重新定义相同的流。有关创建流的更多信息，请参阅 Amazon Kinesis Data Analytics SQL 参考中的 CREATE STREAM。有关创建数据泵的更多信息，请参阅 CREATE PUMP。
- GROUP BY 子句使用多个 ROWTIME 列 - 您只能在 GROUP BY 子句中指定一个 ROWTIME 列。有关更多信息，请参阅 Amazon Kinesis Data Analytics SQL 参考 中的 GROUP BY 和 ROWTIME。
- 一个或多个数据类型有无效转换 - 在这种情况下，您的代码有无效的隐式转换。例如，您可能在代码中将 `timestamp` 转换成 `bigint`。
- 流的名称与服务预留流名称相同 - 流的名称不能与服务预留流 `error_stream` 的名称相同。

应用程序正在将错误写入到错误流

如果您的应用程序正在将错误写入到应用程序内部错误流，则可以使用标准库解码 `DATA_ROW` 字段中的值。有关错误流的更多信息，请参阅错误处理。

吞吐量不足或 MillisBehindLatest 较高

如果应用程序的 MillisBehindLatest 指标稳步增加或始终高于 1000 (1 秒)，这可能是以下原因造成的：

- 检查应用程序的 InputBytes CloudWatch 指标。如果提取速度超过 4 MB/秒，这可能会导致 `MillisBehindLatest` 增加。要提高应用程序的吞吐量，请增加 `InputParallelism` 参数值。有关更多信息，请参阅并行处理输入流以增加吞吐量。
- 检查应用程序的输出传输 Success 指标以确定传输到目标是否失败。确认您已正确配置输出，并且您的输出流具有足够的容量。
- 如果您的应用程序使用 AWS Lambda 函数进行预处理或作为输出，请检查应用程序的 InputProcessing.Duration 或 LambdaDelivery.Duration CloudWatch 指标。如果 Lambda 函数调用持续时间超过 5 秒，请考虑执行以下操作：
  - 提高 Lambda 函数的内存分配。您可以在 AWS Lambda 控制台的 Configuration (配置) 页面上的 Basic settings (基本设置) 中执行此操作。有关更多信息，请参阅 https://docs.aws.amazon.com/lambda/latest/dg/resource-model.html 开发人员指南 中的AWS Lambda配置 Lambda 函数。
  - 在应用程序的输入流中提高分片数。这会提高应用程序将调用的并行函数的数量，从而可能提高吞吐量。
  - 确认函数没有进行影响性能的阻止性调用，如同步的外部资源请求。
  - 检查 AWS Lambda 函数以确定是否具有可提高性能的其他方面。查看应用程序 Lambda 函数的 CloudWatch 日志。有关更多信息，请参阅《AWS Lambda 开发人员指南》中的访问适用于 的 Amazon CloudWatch 指标。

- 确认您的应用程序未达到 Kinesis 处理单元 (KPU) 的默认限制。如果您的应用程序达到该限制，您可以请求提高限制。有关更多信息，请参阅自动扩展应用程序以提高吞吐量。
- 增加 KPU 限制后，如果您的应用程序仍然存在问题，请检查应用程序的输入吞吐量，确认其是否超过 100 MB/秒。如果超过 100 MB/秒，建议进行更改，降低整体吞吐量，从而稳定应用程序，例如通过减少发送至数据源 (Kinesis Data Analytics SQL 应用程序的读取来源) 的数据量。此外，还可以使用其他方法，包括增加应用程序的并行性、缩短计算时间、将列式数据类型从 VARCHAR 更改为更小的数据类型 (例如 INTEGER、LONG 等)，以及减少通过采样或过滤处理的数据。

> **Note**
>
> 建议您定期检查应用程序的 `InputProcessing.OkBytes` 指标，以便您可以规划使用多个 SQL 应用程序，或者如果应用程序的预计输入吞吐量将超过 100 MB/秒，则可以迁移到 managed-flink/latest/java/。

Kinesis Data Analytics SQL 参考

有关 支持的 SQL 语言元素的信息，请参阅 Kinesis Data Analytics SQL 参考。

API 参考

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅Amazon Managed Service for Apache Flink API 文件 (版本 2)。

你可以使用 AWS CLI 来探索亚马逊 Kinesis Data Analytics API。本指南提供了使用 AWS CLI的 适用于 SQL 应用程序的 Amazon Kinesis Data Analytics 入门 练习。

主题

操作

支持以下操作：

- ListApplications
- ListTagsForResource
- StartApplication
- StopApplication
- TagResource
- UntagResource
- UpdateApplication

AddApplicationCloudWatchLoggingOption

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

添加 CloudWatch 日志流来监控应用程序配置错误。有关在 Amazon Kinesis Analytics 应用程序中使用 CloudWatch 日志流的更多信息，请参阅使用 Amazon CloudWatch Logs 。

请求语法

```
{
   "ApplicationName": "string",
   "CloudWatchLoggingOption": {
      "LogStreamARN": "string",
      "RoleARN": "string"
   },
   "CurrentApplicationVersionId": number
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

Kinesis Analytics 应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

CloudWatchLoggingOption

提供 CloudWatch 日志流的 Amazon 资源名称 (ARN) 和 IAM 角色 ARN。注意：如需向 CloudWatch 写入应用程序消息，所使用的 IAM 角色必须启用 `PutLogEvents` 策略操作。

类型：CloudWatchLoggingOption 对象

必需：是

CurrentApplicationVersionId

Kinesis Analytics 应用程序的版本 ID。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

AddApplicationInput

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

向您的 Amazon Kinesis 应用程序添加流式传输源。有关概念信息，请参阅配置应用程序输入。

可以在创建应用程序时添加流式传输源，也可以在创建应用程序后使用此操作添加流式传输源。有关更多信息，请参阅 CreateApplication。

任何配置更新（包括使用此操作添加流式传输源）都会生成新版本的应用程序。您可以使用 DescribeApplication 操作来找到当前应用程序版本。

此操作需要执行 `kinesisanalytics:AddApplicationInput` 操作的权限。

请求语法

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "Input": {
      "InputParallelism": {
         "Count": number
      },
      "InputProcessingConfiguration": {
         "InputLambdaProcessor": {
            "ResourceARN": "string",
            "RoleARN": "string"
         }
      },
      "InputSchema": {
         "RecordColumns": [
            {
               "Mapping": "string",
               "Name": "string",
               "SqlType": "string"
            }
```

```
            ],
            "RecordEncoding": "string",
            "RecordFormat": {
               "MappingParameters": {
                  "CSVMappingParameters": {
                     "RecordColumnDelimiter": "string",
                     "RecordRowDelimiter": "string"
                  },
                  "JSONMappingParameters": {
                     "RecordRowPath": "string"
                  }
               },
               "RecordFormatType": "string"
            }
         },
         "KinesisFirehoseInput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "KinesisStreamsInput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "NamePrefix": "string"
      }
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

您要向其添加流式传输源的现有 Amazon Kinesis Analytics 应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

CurrentApplicationVersionId

您的 Amazon Kinesis Analytics 应用程序的当前版本。您可以使用 DescribeApplication 操作来找到当前应用程序版本。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

Input

要添加的输入。

类型：Input 对象

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

CodeValidationException

用户提供的应用程序代码（查询）无效。这可能是一个简单的语法错误。

HTTP 状态代码：400

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

AddApplicationInputProcessingConfiguration

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

向应用程序添加 InputProcessingConfiguration。在应用程序的 SQL 代码执行之前，输入处理器会预处理输入流上的记录。目前，唯一可用的输入处理器为 AWS Lambda。

请求语法

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "InputId": "string",
   "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
         "ResourceARN": "string",
         "RoleARN": "string"
      }
   }
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

要将输出配置添加到的应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：[a-zA-Z0-9_.-]+

必需：是

CurrentApplicationVersionId

要将输出配置添加到的应用程序的名称。您可以使用 DescribeApplication 操作来获得当前应用程序版本。如果指定的版本不是当前版本，则返回 ConcurrentModificationException。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

InputId

要向其添加输入处理配置的输入配置的 ID。您可以使用 DescribeApplication 操作获取应用程序的输入 ID 列表。

类型：字符串

长度限制：长度下限为 1。最大长度为 50。

模式：[a-zA-Z0-9_.-]+

必需：是

InputProcessingConfiguration

要添加到应用程序中的 InputProcessingConfiguration。

类型：InputProcessingConfiguration 对象

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

AddApplicationOutput

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

将外部目标添加到您的 Amazon Kinesis Analytics 应用程序。

如果您希望 Amazon Kinesis Analytics 将数据从应用程序中的应用程序内部流传递到外部目标（例如 Amazon Kinesis 流、Amazon Kinesis Firehose 传输流或 AWS Lambda 函数），则可以使用此操作将相关配置添加到您的应用程序。您可以为您的应用程序配置一个或多个输出。每个输出配置都映射一个应用程序内部流和一个外部目标。

您可以使用其中一个输出配置将数据从应用程序内的错误流传输到外部目标，以便您分析错误。有关更多信息，请参阅了解应用程序输出（目标）。

任何配置更新（包括使用此操作添加流式传输源）都会生成新版本的应用程序。您可以使用 DescribeApplication 操作来找到当前应用程序版本。

有关可配置的应用程序输入和输出数量的限制，请参阅限制。

此操作需要执行 `kinesisanalytics:AddApplicationOutput` 操作的权限。

请求语法

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "Output": {
      "DestinationSchema": {
         "RecordFormatType": "string"
      },
      "KinesisFirehoseOutput": {
         "ResourceARN": "string",
         "RoleARN": "string"
      },
      "KinesisStreamsOutput": {
         "ResourceARN": "string",
```

```
            "RoleARN": "string"
        },
        "LambdaOutput": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "Name": "string"
    }
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

要将输出配置添加到的应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

CurrentApplicationVersionId

要将输出配置添加到的应用程序的名称。您可以使用 DescribeApplication 操作来获得当前应用程序版本。如果指定的版本不是当前版本，则返回 `ConcurrentModificationException`。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

Output

对象的数组，每个对象描述一项输出配置。在输出配置中，指定应用程序内部流的名称、目标（即 Amazon Kinesis 流、Amazon Kinesis Firehose 传输流或 AWS Lambda 函数），并记录在写入该目标时要使用的格式。

类型：Output 对象

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

AddApplicationReferenceDataSource

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

将引用数据源添加到现有应用程序。

Amazon Kinesis Analytics 读取参考数据（即 Amazon S3 对象），并在应用程序中创建应用程序内部表。在请求中，您提供源（S3 存储桶名称和对象键名称），要创建的应用程序内部表的名称，以及描述 Amazon S3 对象中的数据如何映射到所生成应用程序内部表中的列的必要映射信息。

有关概念信息，请参阅配置应用程序输入。有关您可以添加到应用程序的数据源的限制，请参阅限制。

此操作需要执行 `kinesisanalytics:AddApplicationOutput` 操作的权限。

请求语法

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "ReferenceDataSource": {
      "ReferenceSchema": {
         "RecordColumns": [
            {
               "Mapping": "string",
               "Name": "string",
               "SqlType": "string"
            }
         ],
         "RecordEncoding": "string",
         "RecordFormat": {
            "MappingParameters": {
               "CSVMappingParameters": {
                  "RecordColumnDelimiter": "string",
                  "RecordRowDelimiter": "string"
               },
               "JSONMappingParameters": {
                  "RecordRowPath": "string"
```

```
                }
            },
            "RecordFormatType": "string"
         }
      },
      "S3ReferenceDataSource": {
         "BucketARN": "string",
         "FileKey": "string",
         "ReferenceRoleARN": "string"
      },
      "TableName": "string"
   }
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

现有应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

CurrentApplicationVersionId

要为其添加引用数据来源的应用程序的版本。您可以使用 DescribeApplication 操作来获得当前应用程序版本。如果指定的版本不是当前版本，则返回 `ConcurrentModificationException`。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

ReferenceDataSource

参考数据源可以是 Amazon S3 存储桶中的对象。Amazon Kinesis Analytics 读取对象，并将数据复制到创建的应用程序内部表。您需要提供 S3 存储桶、对象键名称和创建的结果应用程序内部表。

您还必须提供具有必要权限的 IAM 角色，Amazon Kinesis Analytics 可以代入该角色代表您从 S3 存储桶中读取对象。

类型：ReferenceDataSource 对象

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

CreateApplication

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

创建 Amazon Kinesis Analytics 应用程序。您可以对每个应用程序进行配置，将流式传输源用作输入，配置处理输入的应用程序代码，以及您希望 Amazon Kinesis Analytics 写入应用程序输出数据的目的地 (最多 3 个)。有关概述，请参阅工作原理。

在输入配置过程中，您将流式传输源映射到应用程序内部流 (您可以将其视为连续更新表)。在映射过程中，您必须为应用程序内部流提供架构，并将应用程序内部流中的每个数据列与流式传输源中的数据元素一一映射。

您的应用程序代码是一个或多个读取输入数据、转换数据并生成输出的 SQL 语句。您的应用程序代码可以创建一个或多个 SQL 构件，例如 SQL 流或泵。

在输出配置中，您可以通过配置应用程序将应用程序中创建的应用程序内部流的数据写入目的地 (最多 3 个)。

Amazon Kinesis Analytics 需要获得您的许可，才可读取源流的数据或将数据写入目标流。您可以通过创建 IAM 角色授予这些权限。此操作需要执行 `kinesisanalytics:CreateApplication` 操作的权限。

有关创建 Amazon Kinesis Analytics 应用程序的入门练习，请参阅入门。

请求语法

```
{
   "ApplicationCode": "string",
   "ApplicationDescription": "string",
   "ApplicationName": "string",
   "CloudWatchLoggingOptions": [
      {
         "LogStreamARN": "string",
         "RoleARN": "string"
      }
   ],
```

```
   "Inputs": [
      {
         "InputParallelism": {
            "Count": number
         },
         "InputProcessingConfiguration": {
            "InputLambdaProcessor": {
               "ResourceARN": "string",
               "RoleARN": "string"
            }
         },
         "InputSchema": {
            "RecordColumns": [
               {
                  "Mapping": "string",
                  "Name": "string",
                  "SqlType": "string"
               }
            ],
            "RecordEncoding": "string",
            "RecordFormat": {
               "MappingParameters": {
                  "CSVMappingParameters": {
                     "RecordColumnDelimiter": "string",
                     "RecordRowDelimiter": "string"
                  },
                  "JSONMappingParameters": {
                     "RecordRowPath": "string"
                  }
               },
               "RecordFormatType": "string"
            }
         },
         "KinesisFirehoseInput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "KinesisStreamsInput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "NamePrefix": "string"
      }
   ],
```

```
    "Outputs": [
       {
          "DestinationSchema": {
             "RecordFormatType": "string"
          },
          "KinesisFirehoseOutput": {
             "ResourceARN": "string",
             "RoleARN": "string"
          },
          "KinesisStreamsOutput": {
             "ResourceARN": "string",
             "RoleARN": "string"
          },
          "LambdaOutput": {
             "ResourceARN": "string",
             "RoleARN": "string"
          },
          "Name": "string"
       }
    ],
    "Tags": [
       {
          "Key": "string",
          "Value": "string"
       }
    ]
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationCode

一个或多个读取输入数据、转换数据并生成输出的 SQL 语句。例如，您可以编写一条 SQL 语句，该语句从一个应用程序内部流中读取数据，生成供应商广告点击次数的运行平均值，并使用泵将结果行插入另一个应用程序内部流中。有关典型模式的更多信息，请参阅应用程序代码。

您可以提供这样的一系列 SQL 语句，其中一条语句的输出可以用作下一条语句的输入。您可以通过创建应用程序内部流和泵来存储中间结果。

请注意，应用程序代码必须使用在 Outputs 中指定的名称创建流。例如，如果您的 Outputs 定义了名为 ExampleOutputStream1 和 ExampleOutputStream2 的输出流，则您的应用程序代码必须创建这两个流。

类型：字符串

长度限制：长度下限为 0。最大长度为 102400。

必需：否

ApplicationDescription

应用程序的摘要描述。

类型：字符串

长度限制：长度下限为 0。最大长度为 1024。

必需：否

ApplicationName

Amazon Kinesis Analytics 应用程序的名称（例如，sample-app）。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：[a-zA-Z0-9_.-]+

必需：是

CloudWatchLoggingOptions

使用此参数配置 CloudWatch 日志流来监控应用程序配置错误。有关更多信息，请参阅使用 Amazon CloudWatch Logs。

类型：CloudWatchLoggingOption 对象数组

必需：否

Inputs

使用该参数可配置应用程序输入。

您可以将您的应用程序配置为接收来自单个流式传输源的输入。在此配置中，您会将此流式传输源映射到已创建的应用程序内部流。然后，您的应用程序代码可以像对表一样查询应用程序内部流（您可以将其视为连续更新的表）。

对于流式传输源，您可以在该流上提供其 Amazon 资源名称（ARN）和数据格式（例如，JSON、CSV 等）。您还必须提供可由 Amazon Kinesis Analytics 代入以代表您读取此流的 IAM 角色。

要创建应用程序内部流，您需要指定一个架构来将您的数据转换成 SQL 中使用的架构化版本。在架构中，您需要提供流式传输源中数据元素的必要映射，以记录应用程序内部流中的列。

类型：Input 对象数组

必需：否

Outputs

您可以通过配置应用程序输出，将任一应用程序内部流的数据写入最多 3 个目的地。

写入的目的地可以是 Amazon Kinesis 流、Amazon Kinesis Firehose 传输流、AWS Lambda 目的地或三者的任意组合。

在配置中，您可以指定应用程序内部流的名称、目标流或 Lambda 函数的 Amazon 资源名称 (ARN)，以及写入数据时使用的格式。您还必须提供 Amazon Kinesis Analytics 代表您写入到目标流或 Lambda 函数时代入的 IAM 角色。

在输出配置中，您也可以指定输出流或 Lambda 函数 ARN。对于流目标，您可以指定流中数据的格式 (比如JSON、CSV)。您还必须提供 Amazon Kinesis Analytics 代表您写入到流或 Lambda 函数时代入的 IAM 角色。

类型：Output 对象数组

必需：否

Tags

分配给应用程序的一个或多个标签的列表。标签是用于标识应用程序的键/值对。请注意，应用程序标签的最大数量包括系统标签。用户定义的应用程序标签的最大数量为 50。有关更多信息，请参阅使用标签。

类型：Tag 对象数组

数组成员：最少 1 个项目。最多 200 项。

必需：否

响应语法

```
{
   "ApplicationSummary": {
      "ApplicationARN": "string",
      "ApplicationName": "string",
      "ApplicationStatus": "string"
   }
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

ApplicationSummary

针对您的 `CreateApplication` 请求，Amazon Kinesis Analytics 会返回响应，其中包含其创建的应用程序摘要，包括应用程序的Amazon 资源名称 (ARN)、名称和状态。

类型：ApplicationSummary 对象

错误

CodeValidationException

用户提供的应用程序代码（查询）无效。这可能是一个简单的语法错误。

HTTP 状态代码：400

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

LimitExceededException

已超过允许的应用程序数量。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

TooManyTagsException

创建的应用程序具有太多标签，或者向应用程序添加过多的标签。请注意，应用程序标签的最大数量包括系统标签。用户定义的应用程序标签的最大数量为 50。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

DeleteApplication

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

删除指定的应用程序。Amazon Kinesis Analytics 会停止应用程序的执行并删除应用程序，包括任何应用程序构件 (例如应用程序内部流、引用表和应用程序代码)。

此操作需要执行 `kinesisanalytics:DeleteApplication` 操作的权限。

请求语法

```
{
   "ApplicationName": "string",
   "CreateTimestamp": number
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

要删除的 Amazon Kinesis Analytics 应用程序名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

CreateTimestamp

要获取此值，您可以使用 `DescribeApplication` 命令。

类型：时间戳

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK

- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

DeleteApplicationCloudWatchLoggingOption

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

从应用程序中删除 CloudWatch 日志流。有关在 Amazon Kinesis Analytics 应用程序中使用 CloudWatch 日志流的更多信息，请参阅使用 Amazon CloudWatch Logs 。

请求语法

```
{
   "ApplicationName": "string",
   "CloudWatchLoggingOptionId": "string",
   "CurrentApplicationVersionId": number
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

Kinesis Analytics 应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

CloudWatchLoggingOptionId

要删除的 CloudWatch 日志记录选项的 `CloudWatchLoggingOptionId`。您可以使用 DescribeApplication 操作获取 `CloudWatchLoggingOptionId`。

类型：字符串

长度限制：长度下限为 1。最大长度为 50。

模式：[a-zA-Z0-9_.-]+

必需：是

CurrentApplicationVersionId

Kinesis Analytics 应用程序的版本 ID。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

DeleteApplicationInputProcessingConfiguration

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

删除输入中的 InputProcessingConfiguration。

请求语法

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "InputId": "string"
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

Kinesis Analytics 应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

CurrentApplicationVersionId

Kinesis Analytics 应用程序的版本 ID。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

InputId

要从中删除输入处理配置的输入配置的 ID。您可以使用 DescribeApplication 操作获取应用程序的输入 ID 列表。

类型：字符串

长度限制：长度下限为 1。最大长度为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

DeleteApplicationOutput

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

从应用程序配置中删除输出目标配置。Amazon Kinesis Analytics 不再将数据从相应的应用程序内部流写入外部输出目标。

此操作需要执行 `kinesisanalytics:DeleteApplicationOutput` 操作的权限。

请求语法

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "OutputId": "string"
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

Amazon Kinesis Analytics 应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

CurrentApplicationVersionId

Amazon Kinesis Analytics 应用程序的版本。您可以使用 DescribeApplication 操作来获得当前应用程序版本。如果指定的版本不是当前版本，则返回 `ConcurrentModificationException`。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

OutputId

要删除的配置的 ID。在创建应用程序时或稍后使用 AddApplicationOutput 操作添加到应用程序的每个输出配置都具有唯一的ID。您需要提供 ID 以唯一标识要从应用程序配置中删除的输出配置。可以使用 DescribeApplication 操作来获取具体的 `OutputId`。

类型：字符串

长度限制：长度下限为 1。最大长度为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

DeleteApplicationReferenceDataSource

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

从指定的应用程序配置中删除引用数据源配置。

如果应用程序正在运行，Amazon Kinesis Analytics 会立即删除您使用 AddApplicationReferenceDataSource 操作创建的应用程序内部表。

此操作需要执行 `kinesisanalytics.DeleteApplicationReferenceDataSource` 操作的权限。

请求语法

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "ReferenceId": "string"
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

现有应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

CurrentApplicationVersionId

应用程序的版本。您可以使用 DescribeApplication 操作来获得当前应用程序版本。如果指定的版本不是当前版本，则返回 ConcurrentModificationException。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

ReferenceId

引用数据源的 ID。当您使用 AddApplicationReferenceDataSource 向应用程序添加引用数据源时，Amazon Kinesis Analytics 会分配一个 ID。您可以使用 DescribeApplication 操作来获取参考 ID。

类型：字符串

长度限制：长度下限为 1。最大长度为 50。

模式：[a-zA-Z0-9_.-]+

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

DescribeApplication

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

返回特定 Amazon Kinesis Analytics 应用程序的相关信息。

如果您想检索账户中所有应用程序的列表，请使用 ListApplications 操作。

此操作需要执行 `kinesisanalytics:DescribeApplication` 操作的权限。您可以使用 `DescribeApplication` 获取当前应用程序的版本 ID，从而调用其他操作，例如 `Update`。

请求语法

```
{
   "ApplicationName": "string"
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

响应语法

```
{
```

```
   "ApplicationDetail": {
      "ApplicationARN": "string",
      "ApplicationCode": "string",
      "ApplicationDescription": "string",
      "ApplicationName": "string",
      "ApplicationStatus": "string",
      "ApplicationVersionId": number,
      "CloudWatchLoggingOptionDescriptions": [
         {
            "CloudWatchLoggingOptionId": "string",
            "LogStreamARN": "string",
            "RoleARN": "string"
         }
      ],
      "CreateTimestamp": number,
      "InputDescriptions": [
         {
            "InAppStreamNames": [ "string" ],
            "InputId": "string",
            "InputParallelism": {
               "Count": number
            },
            "InputProcessingConfigurationDescription": {
               "InputLambdaProcessorDescription": {
                  "ResourceARN": "string",
                  "RoleARN": "string"
               }
            },
            "InputSchema": {
               "RecordColumns": [
                  {
                     "Mapping": "string",
                     "Name": "string",
                     "SqlType": "string"
                  }
               ],
               "RecordEncoding": "string",
               "RecordFormat": {
                  "MappingParameters": {
                     "CSVMappingParameters": {
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                     },
                     "JSONMappingParameters": {
```

```
                        "RecordRowPath": "string"
                     }
                  },
                  "RecordFormatType": "string"
               }
            },
            "InputStartingPositionConfiguration": {
               "InputStartingPosition": "string"
            },
            "KinesisFirehoseInputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "KinesisStreamsInputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "NamePrefix": "string"
         }
      ],
      "LastUpdateTimestamp": number,
      "OutputDescriptions": [
         {
            "DestinationSchema": {
               "RecordFormatType": "string"
            },
            "KinesisFirehoseOutputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "KinesisStreamsOutputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "LambdaOutputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "Name": "string",
            "OutputId": "string"
         }
      ],
      "ReferenceDataSourceDescriptions": [
         {
```

```
            "ReferenceId": "string",
            "ReferenceSchema": {
               "RecordColumns": [
                  {
                     "Mapping": "string",
                     "Name": "string",
                     "SqlType": "string"
                  }
               ],
               "RecordEncoding": "string",
               "RecordFormat": {
                  "MappingParameters": {
                     "CSVMappingParameters": {
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                     },
                     "JSONMappingParameters": {
                        "RecordRowPath": "string"
                     }
                  },
                  "RecordFormatType": "string"
               }
            },
            "S3ReferenceDataSourceDescription": {
               "BucketARN": "string",
               "FileKey": "string",
               "ReferenceRoleARN": "string"
            },
            "TableName": "string"
         }
      ]
   }
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

ApplicationDetail

提供对应用程序的描述，例如应用程序 Amazon 资源名称（ARN）、状态、最新版本以及输入和输出配置详情。

类型：ApplicationDetail 对象

错误

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

DiscoverInputSchema

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

通过评估指定流式传输源 (Amazon Kinesis 流或 Amazon Kinesis Firehose 传输流) 或 S3 对象上的示例记录，推断架构。在响应中，该操作返回推断的架构以及该操作用于推断架构的示例记录。

在为应用程序配置流式传输来源时，您可以使用此推断的架构。有关概念信息，请参阅配置应用程序输入。注意：使用 Amazon Kinesis Analytics 控制台创建应用程序时，控制台使用此操作来推断架构并将其显示在控制台用户界面中。

此操作需要执行 `kinesisanalytics:DiscoverInputSchema` 操作的权限。

请求语法

```
{
   "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
         "ResourceARN": "string",
         "RoleARN": "string"
      }
   },
   "InputStartingPositionConfiguration": {
      "InputStartingPosition": "string"
   },
   "ResourceARN": "string",
   "RoleARN": "string",
   "S3Configuration": {
      "BucketARN": "string",
      "FileKey": "string",
      "RoleARN": "string"
   }
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

InputProcessingConfiguration

InputProcessingConfiguration 用于在发现记录架构之前对记录进行预处理。

类型：InputProcessingConfiguration 对象

必需：否

InputStartingPositionConfiguration

您希望 Amazon Kinesis Analytics 根据指定流式传输源发现目的开始读取记录的时刻。

类型：InputStartingPositionConfiguration 对象

必需：否

ResourceARN

流式传输来源的 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。

模式：`arn:.*`

必需：否

RoleARN

可由 Amazon Kinesis Analytics 代入以代表您访问流的 IAM 角色的 ARN。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

模式：`arn:.*`

必需：否

S3Configuration

指定此参数可从 Amazon S3 对象中的数据发现架构。

类型：S3Configuration 对象

必需：否

响应语法

```
{
   "InputSchema": {
      "RecordColumns": [
         {
            "Mapping": "string",
            "Name": "string",
            "SqlType": "string"
         }
      ],
      "RecordEncoding": "string",
      "RecordFormat": {
         "MappingParameters": {
            "CSVMappingParameters": {
               "RecordColumnDelimiter": "string",
               "RecordRowDelimiter": "string"
            },
            "JSONMappingParameters": {
               "RecordRowPath": "string"
            }
         },
         "RecordFormatType": "string"
      }
   },
   "ParsedInputRecords": [
      [ "string" ]
   ],
   "ProcessedInputRecords": [ "string" ],
   "RawInputRecords": [ "string" ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

InputSchema

从流式传输来源推断出的架构。它识别流式传输来源中数据的格式，以及每个数据元素如何映射到在应用程序内部流中创建的相应列。

类型：SourceSchema 对象

ParsedInputRecords

一个元素数组，其中每个元素对应于流记录中的一行（流记录可以有多行）。

类型：字符串数组的数组。

ProcessedInputRecords

由 `InputProcessingConfiguration` 参数中指定的处理器修改的流数据。

类型：字符串数组

RawInputRecords

为推断架构而采样的原始流数据。

类型：字符串数组

错误

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceProvisionedThroughputExceededException

由于 Amazon Kinesis Streams ProvisionedThroughputExceededException，Discovery 未能从流式传输源获得记录。有关更多信息，请参阅《Amazon Kinesis Streams API 参考》中的 GetRecords。

HTTP 状态代码：400

ServiceUnavailableException

该服务不可用。退后一步，重试该操作。

HTTP 状态代码：500

UnableToDetectSchemaException

数据格式无效。Amazon Kinesis Analytics 无法检测给定流式传输源的架构。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

ListApplications

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

返回您账户中的 Amazon Kinesis Analytics 应用程序列表。每个应用程序的响应都包括应用程序的名称、Amazon 资源名称（ARN）以及状态。如果响应返回 `HasMoreApplications` 值为 true，则可以通过在请求正文中添加 `ExclusiveStartApplicationName` 来发送另一个请求，然后将其值设置为上一个响应中的最后一个应用程序名称。

如果您想了解有关特定应用程序的详细信息，请使用 DescribeApplication。

此操作需要执行 `kinesisanalytics:ListApplications` 操作的权限。

请求语法

```
{
   "ExclusiveStartApplicationName": "string",
   "Limit": number
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ExclusiveStartApplicationName

列表开头应用程序的名称。在使用分页检索列表时，不需要在第一个请求中指定此参数。但是，在随后的请求中，您可以通过添加上一个响应中的最后一个应用程序名称，获取下一页的应用程序。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：否

Limit

应用程序的最大数量

类型：整数

有效范围：最小值为 1。最大值为 50。

必需：否

响应语法

```
{
   "ApplicationSummaries": [
      {
         "ApplicationARN": "string",
         "ApplicationName": "string",
         "ApplicationStatus": "string"
      }
   ],
   "HasMoreApplications": boolean
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

ApplicationSummaries

ApplicationSummary 对象的列表。

类型：ApplicationSummary 对象数组

HasMoreApplications

如需检索更多应用程序，则返回 true。

类型：布尔值

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

ListTagsForResource

检索分配给应用程序的键值标签列表。有关更多信息，请参阅使用标记。

请求语法

```
{
   "ResourceARN": "string"
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ResourceARN

要检索其标签的应用程序的 ARN。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

模式：arn:.*

必需：是

响应语法

```
{
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

Tags

分配给应用程序的键值标签。

类型：Tag 对象数组

数组成员：最少 1 个项目。最多 200 项。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK

- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

StartApplication

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

启动指定的 Amazon Kinesis Analytics 应用程序。创建应用程序后，必须以独占方式调用此操作才能启动应用程序。

应用程序启动后，开始使用输入数据，对其进行处理，然后将输出写入配置的目的地。

应用程序在 READY 状态下才能启动。您可以在控制台中或通过使用 DescribeApplication 操作查看应用程序状态。

启动应用程序后，您可以通过调用 stopApplication 操作来使应用程序暂停处理输入。

此操作需要执行 kinesisanalytics:StartApplication 操作的权限。

请求语法

```
{
   "ApplicationName": "string",
   "InputConfigurations": [
      {
         "Id": "string",
         "InputStartingPositionConfiguration": {
            "InputStartingPosition": "string"
         }
      }
   ]
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

InputConfigurations

使用 ID 标识应用程序最开始使用的输入。Amazon Kinesis Analytics 开始读取与输入相关的流式传输源。您还可以指定流式传输源中 Amazon Kinesis Analytics 读取起始位置。

类型：InputConfiguration 对象数组

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

InvalidApplicationConfigurationException

用户提供的应用程序配置无效。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

StopApplication

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

暂停应用程序处理输入数据。只有当应用程序处于运行状态时，才可以暂停该应用程序。您可以使用 DescribeApplication 操作来找到应用程序状态。应用程序暂停后，Amazon Kinesis Analytics 不再从输入中读取数据，应用程序不再处理数据，并且不产生写入目标的输出。

此操作需要执行 `kinesisanalytics:StopApplication` 操作的权限。

请求语法

```
{
   "ApplicationName": "string"
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

要停止的运行中应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

TagResource

向 Kinesis Analytics 应用程序添加一个或多个键值标签。请注意，应用程序标签的最大数量包括系统标签。用户定义的应用程序标签的最大数量为 50。有关更多信息，请参阅使用标签。

请求语法

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ResourceARN

向其分配标签的应用程序的 ARN。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

模式：`arn:.*`

必需：是

Tags

分配给应用程序的键值标签。

类型：Tag 对象数组

数组成员：最少 1 个项目。最多 200 项。

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

TooManyTagsException

创建的应用程序具有太多标签，或者向应用程序添加过多的标签。请注意，应用程序标签的最大数量包括系统标签。用户定义的应用程序标签的最大数量为 50。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK

- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

UntagResource

删除 Kinesis Analytics 应用程序中的一个或多个标签。有关更多信息，请参阅使用标记。

请求语法

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ResourceARN

要删除标签的 Kinesis Analytics 应用程序 ARN。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

模式：`arn:.*`

必需：是

TagKeys

要从指定应用程序中移除的标签键列表。

类型：字符串数组

数组成员：最少 1 个项目。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

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

TooManyTagsException

创建的应用程序具有太多标签，或者向应用程序添加过多的标签。请注意，应用程序标签的最大数量包括系统标签。用户定义的应用程序标签的最大数量为 50。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3

- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

UpdateApplication

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

更新现有的 Amazon Kinesis Analytics 应用程序。使用此 API，您可以更新应用程序代码、输入配置和输出配置。

请注意，每次更新应用程序时，Amazon Kinesis Analytics 都会更新 `CurrentApplicationVersionId`。

此操作需要 `kinesisanalytics:UpdateApplication` 操作权限。

请求语法

```
{
   "ApplicationName": "string",
   "ApplicationUpdate": {
      "ApplicationCodeUpdate": "string",
      "CloudWatchLoggingOptionUpdates": [
         {
            "CloudWatchLoggingOptionId": "string",
            "LogStreamARNUpdate": "string",
            "RoleARNUpdate": "string"
         }
      ],
      "InputUpdates": [
         {
            "InputId": "string",
            "InputParallelismUpdate": {
               "CountUpdate": number
            },
            "InputProcessingConfigurationUpdate": {
               "InputLambdaProcessorUpdate": {
                  "ResourceARNUpdate": "string",
                  "RoleARNUpdate": "string"
               }
            },
```

```
            "InputSchemaUpdate": {
               "RecordColumnUpdates": [
                  {
                     "Mapping": "string",
                     "Name": "string",
                     "SqlType": "string"
                  }
               ],
               "RecordEncodingUpdate": "string",
               "RecordFormatUpdate": {
                  "MappingParameters": {
                     "CSVMappingParameters": {
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                     },
                     "JSONMappingParameters": {
                        "RecordRowPath": "string"
                     }
                  },
                  "RecordFormatType": "string"
               }
            },
            "KinesisFirehoseInputUpdate": {
               "ResourceARNUpdate": "string",
               "RoleARNUpdate": "string"
            },
            "KinesisStreamsInputUpdate": {
               "ResourceARNUpdate": "string",
               "RoleARNUpdate": "string"
            },
            "NamePrefixUpdate": "string"
         }
      ],
      "OutputUpdates": [
         {
            "DestinationSchemaUpdate": {
               "RecordFormatType": "string"
            },
            "KinesisFirehoseOutputUpdate": {
               "ResourceARNUpdate": "string",
               "RoleARNUpdate": "string"
            },
            "KinesisStreamsOutputUpdate": {
               "ResourceARNUpdate": "string",
```

```
            "RoleARNUpdate": "string"
         },
         "LambdaOutputUpdate": {
            "ResourceARNUpdate": "string",
            "RoleARNUpdate": "string"
         },
         "NameUpdate": "string",
         "OutputId": "string"
      }
   ],
   "ReferenceDataSourceUpdates": [
      {
         "ReferenceId": "string",
         "ReferenceSchemaUpdate": {
            "RecordColumns": [
               {
                  "Mapping": "string",
                  "Name": "string",
                  "SqlType": "string"
               }
            ],
            "RecordEncoding": "string",
            "RecordFormat": {
               "MappingParameters": {
                  "CSVMappingParameters": {
                     "RecordColumnDelimiter": "string",
                     "RecordRowDelimiter": "string"
                  },
                  "JSONMappingParameters": {
                     "RecordRowPath": "string"
                  }
               },
               "RecordFormatType": "string"
            }
         },
         "S3ReferenceDataSourceUpdate": {
            "BucketARNUpdate": "string",
            "FileKeyUpdate": "string",
            "ReferenceRoleARNUpdate": "string"
         },
         "TableNameUpdate": "string"
      }
   ]
},
```

```
    "CurrentApplicationVersionId": number
}
```

请求参数

请求接受采用 JSON 格式的以下数据。

ApplicationName

要更新的 Amazon Kinesis Analytics 应用程序的名称。

类型：字符串

长度限制：长度下限为 1。长度上限为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

ApplicationUpdate

描述应用程序的更新。

类型：ApplicationUpdate 对象

必需：是

CurrentApplicationVersionId

当前应用程序版本 ID。你可以使用 DescribeApplication 操作来获取这个值。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 200 响应。

错误

CodeValidationException

用户提供的应用程序代码（查询）无效。这可能是一个简单的语法错误。

HTTP 状态代码：400

ConcurrentModificationException

由于对应用程序进行并发修改而引发的异常。例如，两个人尝试同时编辑同一个应用程序。

HTTP 状态代码：400

InvalidArgumentException

指定的输入参数值无效。

HTTP 状态代码：400

ResourceInUseException

该应用程序不可用于此操作。

HTTP 状态代码：400

ResourceNotFoundException

无法找到指定的应用程序。

HTTP 状态代码：400

UnsupportedOperationException

请求被拒绝，因为不支持指定的参数或指定的资源对此操作无效。

HTTP 状态代码：400

另请参阅

有关在特定语言的 AWS SDK 中使用此 API 的更多信息，请参阅以下内容：

- AWS 命令行界面
- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- AWS 适用于 JavaScript 的开发工具包 V3
- 适用于 PHP V3 的 AWS SDK

- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

数据类型

支持以下数据类型：

- ApplicationDetail
- ApplicationSummary
- ApplicationUpdate
- CloudWatchLoggingOption
- CloudWatchLoggingOptionDescription
- CloudWatchLoggingOptionUpdate
- CSVMappingParameters
- DestinationSchema
- Input
- InputConfiguration
- InputDescription
- InputLambdaProcessor
- InputLambdaProcessorDescription
- InputLambdaProcessorUpdate
- InputParallelism
- InputParallelismUpdate
- InputProcessingConfiguration
- InputProcessingConfigurationDescription
- InputProcessingConfigurationUpdate
- InputSchemaUpdate
- InputStartingPositionConfiguration
- InputUpdate
- JSONMappingParameters
- KinesisFirehoseInput

- KinesisFirehoseInputDescription
- KinesisFirehoseInputUpdate
- KinesisFirehoseOutput
- KinesisFirehoseOutputDescription
- KinesisFirehoseOutputUpdate
- KinesisStreamsInput
- KinesisStreamsInputDescription
- KinesisStreamsInputUpdate
- KinesisStreamsOutput
- KinesisStreamsOutputDescription
- KinesisStreamsOutputUpdate
- LambdaOutput
- LambdaOutputDescription
- LambdaOutputUpdate
- MappingParameters
- Output
- OutputDescription
- OutputUpdate
- RecordColumn
- RecordFormat
- ReferenceDataSource
- ReferenceDataSourceDescription
- ReferenceDataSourceUpdate
- S3Configuration
- S3ReferenceDataSource
- S3ReferenceDataSourceDescription
- S3ReferenceDataSourceUpdate
- SourceSchema
- Tag

ApplicationDetail

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

提供对应用程序的描述，包括应用程序 Amazon 资源名称（ARN）、状态、最新版本以及输入和输出配置。

目录

ApplicationARN

应用程序的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

ApplicationName

应用程序的名称。

类型：字符串

长度限制：最小长度为 1。最大长度为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

ApplicationStatus

应用程序的状态。

类型：字符串

有效值：DELETING | STARTING | STOPPING | READY | RUNNING | UPDATING | AUTOSCALING

必需：是

ApplicationVersionId

请注意当前应用程序版本。

类型：长整型

有效范围：最小值为 1。最大值为 999999999。

必需：是

ApplicationCode

返回应用程序中您对任何应用程序内部流执行数据分析而提供的应用程序代码。

类型：字符串

长度约束：最小长度为 0。最大长度为 102400。

必需：否

ApplicationDescription

应用程序的描述。

类型：字符串

长度约束：最小长度为 0。长度上限为 1024。

必需：否

CloudWatchLoggingOptionDescriptions

描述配置用于接收应用程序消息的 CloudWatch 日志流。有关在 Amazon Kinesis Analytics 应用程序中使用 CloudWatch 日志流的更多信息，请参阅使用 Amazon CloudWatch Logs 。

类型：CloudWatchLoggingOptionDescription 对象数组

必需：否

CreateTimestamp

创建应用程序版本的时间戳。

类型：时间戳

必需：否

InputDescriptions

描述应用程序输入配置。有关更多信息，请参阅配置应用程序输入。

类型：InputDescription 对象数组

必需：否

LastUpdateTimestamp

上次更新应用程序的时间戳。

类型：时间戳

必需：否

OutputDescriptions

描述应用程序输出配置。有关更多信息，请参阅配置应用程序输出。

类型：OutputDescription 对象数组

必需：否

ReferenceDataSourceDescriptions

描述为应用程序配置的引用数据来源。有关更多信息，请参阅配置应用程序输入。

类型：ReferenceDataSourceDescription 对象数组

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

ApplicationSummary

> **Note**
>
> 本文档适用于 Amazon Kinesis Data Analytics API 版本 1，该版本仅支持 SQL 应用程序。版本 2 的 API 支持 SQL 和 Java 应用程序。有关版本 2 的更多信息，请参阅 Amazon Kinesis Data Analytics API V2 文档。

提供应用程序摘要信息，包括应用程序 Amazon 资源名称（ARN）、名称和状态。

目录

ApplicationARN

应用程序的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

ApplicationName

应用程序的名称。

类型：字符串

长度限制：最小长度为 1。最大长度为 128。

模式：`[a-zA-Z0-9_.-]+`

必需：是

ApplicationStatus

应用程序的状态。

类型：字符串

有效值：DELETING | STARTING | STOPPING | READY | RUNNING | UPDATING | AUTOSCALING

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

ApplicationUpdate

描述适用于现有 Amazon Kinesis Analytics 应用程序的更新。

目录

ApplicationCodeUpdate

描述应用程序代码更新。

类型：字符串

长度约束：最小长度为 0。最大长度为 102400。

必需：否

CloudWatchLoggingOptionUpdates

描述应用程序 CloudWatch 日志记录选项更新。

类型：CloudWatchLoggingOptionUpdate 对象数组

必需：否

InputUpdates

描述应用程序输入配置更新。

类型：InputUpdate 对象数组

必需：否

OutputUpdates

描述应用程序输出配置更新。

类型：OutputUpdate 对象数组

必需：否

ReferenceDataSourceUpdates

描述应用程序参考数据源更新。

类型：ReferenceDataSourceUpdate 对象数组

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

CloudWatchLoggingOption

提供有关 CloudWatch 日志记录选项的描述，包括日志流 Amazon 资源名称 (ARN) 以及角色 ARN。

目录

LogStreamARN

接收应用程序消息的 CloudWatch 日志的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

RoleARN

用于发送应用程序消息的角色的 IAM ARN。注意：如需向 CloudWatch 写入应用程序消息，所使用的 IAM 角色必须启用 `PutLogEvents` 策略操作。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

CloudWatchLoggingOptionDescription

CloudWatch 日志记录选项的描述。

目录

LogStreamARN

接收应用程序消息的 CloudWatch 日志的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

RoleARN

用于发送应用程序消息的角色的 IAM ARN。注意：如需向 CloudWatch 写入应用程序消息，所使用的 IAM 角色必须启用 `PutLogEvents` 策略操作。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

CloudWatchLoggingOptionId

CloudWatch 日志记录选项描述的 ID。

类型：字符串

长度限制：最小长度为 1。长度上限为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

CloudWatchLoggingOptionUpdate

描述 CloudWatch 日志记录选项更新。

目录

CloudWatchLoggingOptionId

要更新的 CloudWatch 日志记录选项的 ID

类型：字符串

长度限制：最小长度为 1。长度上限为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：是

LogStreamARNUpdate

接收应用程序消息的 CloudWatch 日志的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARNUpdate

用于发送应用程序消息的角色的 IAM ARN。注意：如需向 CloudWatch 写入应用程序消息，所使用的 IAM 角色必须启用 `PutLogEvents` 策略操作。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

CSVMappingParameters

提供当记录格式使用带分隔符的格式（如 CSV）时的其他映射信息。例如，以下示例记录使用 CSV 格式，其中记录使用“\n”作为行分隔符，使用逗号（“,”）作为列分隔符：

```
"name1", "address1"

"name2", "address2"
```

目录

RecordColumnDelimiter

列分隔符。例如，在 CSV 格式中，逗号（“,”）是典型列分隔符。

类型：字符串

长度限制：最小长度为 1。

必需：是

RecordRowDelimiter

行分隔符。例如，在 CSV 格式中，“\n”是典型行分隔符。

类型：字符串

长度限制：最小长度为 1。

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

DestinationSchema

描述将记录写入目标时的数据格式。有关更多信息，请参阅配置应用程序输出。

目录

RecordFormatType

指定输出流上的记录格式。

类型：字符串

有效值：`JSON | CSV`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

Input

配置应用程序输入时，请指定流式源、创建的应用程序内部流名称以及二者之间的映射。有关更多信息，请参阅配置应用程序输入。

目录

InputSchema

描述流式源中的数据的格式以及每个数据元素映射到所创建应用程序内部流中的相应列的方式。

还用于描述引用数据来源的格式。

类型：SourceSchema 对象

必需：是

NamePrefix

创建应用程序内部流时要使用的名称前缀。假设您指定了前缀“MyInApplicationStream”。Amazon Kinesis Analytics 随后创建一个或多个（根据您指定的 `InputParallelism` 计数）应用程序内部流，其名称分别为“MyInApplicationStream_001”、“MyInApplicationStream_002”，以此类推。

类型：字符串

长度限制：最小长度为 1。最大长度为 32。

必需：是

InputParallelism

描述要创建的应用程序内部流的数量。

您的源中的数据将路由到这些应用程序内部输入流。

（请参阅配置应用程序输入。）

类型：InputParallelism 对象

必需：否

InputProcessingConfiguration

输入的 InputProcessingConfiguration。在应用程序的 SQL 代码执行之前，输入处理器会在从流收到记录时转换记录。目前，唯一可用的输入处理配置为 InputLambdaProcessor。

类型：InputProcessingConfiguration 对象

必需：否

KinesisFirehoseInput

如果流式源是一个 Amazon Kinesis Firehose 传输流，则标识该传输流的 ARN 和可让 Amazon Kinesis Analytics 代表您访问该流的 IAM 角色。

注意：需要 `KinesisStreamsInput` 或 `KinesisFirehoseInput` 之一。

类型：KinesisFirehoseInput 对象

必需：否

KinesisStreamsInput

如果流式源是一个 Amazon Kinesis 流，则标识该传输流的 Amazon 资源名称（ARN）和可让 Amazon Kinesis Analytics 代表您访问该流的 IAM 角色。

注意：需要 `KinesisStreamsInput` 或 `KinesisFirehoseInput` 之一。

类型：KinesisStreamsInput 对象

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputConfiguration

启动应用程序时，您需要提供此配置，该配置用于标识输入源以及您希望应用程序开始处理记录的输入源中的时间点。

目录

Id

输入源 ID。您可以通过调用 DescribeApplication 操作来获取此 ID。

类型：字符串

长度限制：最小长度为 1。长度上限为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：是

InputStartingPositionConfiguration

您希望应用程序开始处理来自流式传输源的记录的时间点。

类型：InputStartingPositionConfiguration 对象

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputDescription

描述应用程序输入配置。有关更多信息，请参阅配置应用程序输入。

目录

InAppStreamNames

返回映射到流式传输源的应用程序内流名称。

类型：字符串数组

长度限制：最小长度为 1。最大长度为 32。

必需：否

InputId

输入与应用程序输入关联的 ID。这是 Amazon Kinesis Analytics 为您添加到您应用程序中的每个输入配置分配的 ID。

类型：字符串

长度限制：最小长度为 1。长度上限为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：否

InputParallelism

描述配置的并行度 (映射到流式传输源的应用程序内流的数量)。

类型：InputParallelism 对象

必需：否

InputProcessingConfigurationDescription

应用程序代码运行之前，在此输入中的记录上执行的预处理器的描述。

类型：InputProcessingConfigurationDescription 对象

必需：否

InputSchema

描述流式源中的数据的格式以及每个数据元素映射到所创建应用程序内部流中的相应列的方式。

类型：SourceSchema 对象

必需：否

InputStartingPositionConfiguration

将应用程序配置为从输入流中读取数据的时间点。

类型：InputStartingPositionConfiguration 对象

必需：否

KinesisFirehoseInputDescription

如果 Amazon Kinesis Firehose 传输流被配置为流式源，则标识该传输流的 ARN 和可让 Amazon Kinesis Analytics 代表您访问该流的 IAM 角色。

类型：KinesisFirehoseInputDescription 对象

必需：否

KinesisStreamsInputDescription

如果 Amazon Kinesis 流被配置为流式源，则标识 Amazon Kinesis 流的 Amazon 资源名称 (ARN) 和可让 Amazon Kinesis Analytics 代表您访问该流的 IAM 角色。

类型：KinesisStreamsInputDescription 对象

必需：否

NamePrefix

应用程序内部名称前缀。

类型：字符串

长度限制：最小长度为 1。最大长度为 32。

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputLambdaProcessor

一个对象，其中包含用于预处理流中记录的 AWS Lambda 函数的 Amazon 资源名称（ARN），以及用于访问 AWS Lambda 函数的 IAM 角色的 ARN。

目录

ResourceARN

用于处理流中记录的 AWS Lambda 函数的 ARN。

> **Note**
>
> 要指定相比最新版本较早的 Lambda 函数版本，请在 Lambda 函数 ARN 中包括 Lambda 函数版本。有关 Lambda ARN 的更多信息，请参阅示例 ARN：AWS Lambda

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

RoleARN

用于访问 AWS Lambda 函数的 IAM 角色的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK

- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputLambdaProcessorDescription

一个对象，其中包含用于预处理流中记录的 AWS Lambda 函数的 Amazon 资源名称（ARN），以及用于访问 AWS Lambda 表达的 IAM 角色的 ARN。

目录

ResourceARN

用于预处理流中记录的 AWS Lambda 函数的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARN

用于访问 AWS Lambda 函数的 IAM 角色的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputLambdaProcessorUpdate

表示用于预处理流中记录的 InputLambdaProcessor 的更新。

目录

ResourceARNUpdate

用于预处理流中记录的新 AWS Lambda 函数的 Amazon 资源名称 (ARN)。

> **Note**
> 要指定相比最新版本较早的 Lambda 函数版本，请在 Lambda 函数 ARN 中包括 Lambda 函数版本。有关 Lambda ARN 的更多信息，请参阅示例 ARN：AWS Lambda

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARNUpdate

用于访问 AWS Lambda 函数的新 IAM 角色的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK

- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputParallelism

描述要为指定流式源创建的应用程序内部流的数量。有关并行的信息，请参阅配置应用程序输入。

目录

Count

要创建的应用程序内部流的数量。有关更多信息，请参阅限制。

类型：整数

有效范围：最小值为 1。最大值为 64。

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputParallelismUpdate

提供并行度计数的更新。

目录

CountUpdate

为指定流式传输源创建的应用程序内部流的数量。

类型：整数

有效范围：最小值为 1。最大值为 64。

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputProcessingConfiguration

提供处理器的描述，该处理器用于在您的应用程序代码处理流中记录之前预处理这些记录。目前，唯一可用的输入处理器为 AWS Lambda。

目录

InputLambdaProcessor

用于在您的应用程序代码处理流中记录之前预处理这些记录的 InputLambdaProcessor。

类型：InputLambdaProcessor 对象

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputProcessingConfigurationDescription

提供有关输入处理器的配置信息。目前，唯一可用的输入处理器为 AWS Lambda。

目录

InputLambdaProcessorDescription

提供相关联的 InputLambdaProcessorDescription 操作的相关配置信息

类型：InputLambdaProcessorDescription 对象

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputProcessingConfigurationUpdate

描述 InputProcessingConfiguration 的更新。

目录

InputLambdaProcessorUpdate

提供 InputLambdaProcessor 的更新信息。

类型：InputLambdaProcessorUpdate 对象

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputSchemaUpdate

描述应用程序输入架构的更新。

目录

RecordColumnUpdates

`RecordColumn` 对象的列表。每个对象描述流式传输源元素到应用程序内部流中的相应列的映射。

类型：RecordColumn 对象数组

数组成员：最少 1 项。最多 1000 项。

必需：否

RecordEncodingUpdate

指定流式源中的记录的编码。例如，UTF-8。

类型：字符串

模式：`UTF-8`

必需：否

RecordFormatUpdate

指定流式源上的记录的格式。

类型：RecordFormat 对象

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputStartingPositionConfiguration

描述应用程序从流式传输源读取数据的时刻。

目录

InputStartingPosition

流上的开始位置。

- NOW - 在流中最新的记录之后开始读取，从客户发出的请求时间戳开始。
- TRIM_HORIZON - 在流中最后一条未修剪的记录处开始读取，这是流中最早的记录。该选项不可用于 Amazon Kinesis Data Firehose 传输流。
- LAST_STOPPED_POINT - 从应用程序上次停止读取的位置继续读取。

类型：字符串

有效值：NOW | TRIM_HORIZON | LAST_STOPPED_POINT

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

InputUpdate

描述特定输入配置 (由应用程序 `InputId` 标识) 的更新。

目录

InputId

要更新的应用程序输入的输入 ID。

类型：字符串

长度限制：最小长度为 1。长度上限为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：是

InputParallelismUpdate

描述并行度更新 (Amazon Kinesis Analytics 为特定流式传输源创建的应用程序内部流的数量)。

类型：InputParallelismUpdate 对象

必需：否

InputProcessingConfigurationUpdate

描述输入处理配置的更新。

类型：InputProcessingConfigurationUpdate 对象

必需：否

InputSchemaUpdate

描述流式传输源中的数据格式，以及流式传输源上的每个记录元素映射到所创建应用程序内部流中相应列的方式。

类型：InputSchemaUpdate 对象

必需：否

KinesisFirehoseInputUpdate

如果 Amazon Kinesis Firehose 传输流是要更新的流式传输源，则会提供更新的流 ARN 和 IAM 角色 ARN。

类型：KinesisFirehoseInputUpdate 对象

必需：否

KinesisStreamsInputUpdate

如果 Amazon Kinesis 流是要更新的流式传输源，则提供更新的流的 Amazon 资源名称 (ARN) 和 IAM 角色 ARN。

类型：KinesisStreamsInputUpdate 对象

必需：否

NamePrefixUpdate

Amazon Kinesis Analytics 为特定流式传输源创建的应用程序内部流的名称前缀。

类型：字符串

长度限制：最小长度为 1。最大长度为 32。

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

JSONMappingParameters

提供当 JSON 是流式源上的记录格式时的其他映射信息。

目录

RecordRowPath

指向包含记录的顶层父级的路径。

类型：字符串

长度限制：最小长度为 1。

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisFirehoseInput

将 Amazon Kinesis Firehose 传输流标识为流式源。您提供传输流的 Amazon 资源名称（ARN）和可让 Amazon Kinesis Analytics 代表您访问该流的 IAM 角色 ARN。

目录

ResourceARN

输入传输流的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

RoleARN

可由 Amazon Kinesis Analytics 代入以代表您访问流的 IAM 角色的 ARN。您需要确保该角色有必要的权限以访问流。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisFirehoseInputDescription

描述在应用程序输入配置中配置为流式传输源的 Amazon Kinesis Firehose 传输流。

目录

ResourceARN

Amazon Kinesis Firehose 传输流的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARN

可由 Amazon Kinesis Analytics 代入以访问流的 IAM 角色的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisFirehoseInputUpdate

更新应用程序输入配置时，提供有关 Amazon Kinesis Firehose 传输流作为流式传输源的信息。

目录

ResourceARNUpdate

要读取的输入 Amazon Kinesis Firehose 传输流的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARNUpdate

可由 Amazon Kinesis Analytics 代入以代表您访问流的 IAM 角色的 ARN。您需要向此角色授予必需的权限。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisFirehoseOutput

配置应用程序输出时，将 Amazon Kinesis Firehose 传输流标识为目标。您提供流的 Amazon 资源名称（ARN）和可让 Amazon Kinesis Analytics 代表您写入流的 IAM 角色。

目录

ResourceARN

要写入的目标 Amazon Kinesis Firehose 传输流的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

RoleARN

可由 Amazon Kinesis Analytics 代入以代表您对目标流进行写入的 IAM 角色的 ARN。您需要向此角色授予必需的权限。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisFirehoseOutputDescription

对于应用程序输出，将 Amazon Kinesis Firehose 传输流标识为其目标。

目录

ResourceARN

Amazon Kinesis Firehose 传输流的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARN

可由 Amazon Kinesis Analytics 代入以访问流的 IAM 角色的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisFirehoseOutputUpdate

使用 UpdateApplication 操作更新输出配置时，提供有关配置为目标的 Amazon Kinesis Firehose 传输流的信息。

目录

ResourceARNUpdate

要写入的 Amazon Kinesis Firehose 传输流的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARNUpdate

可由 Amazon Kinesis Analytics 代入以代表您访问流的 IAM 角色的 ARN。您需要向此角色授予必需的权限。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisStreamsInput

将 Amazon Kinesis 流标识为流式源。您提供流的 Amazon 资源名称（ARN）和可让 Amazon Kinesis Analytics 代表您访问该流的 IAM 角色 ARN。

目录

ResourceARN

要读取的输入 Amazon Kinesis 流的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

RoleARN

可由 Amazon Kinesis Analytics 代入以代表您访问流的 IAM 角色的 ARN。您需要向此角色授予必需的权限。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisStreamsInputDescription

描述在应用程序输入配置中配置为流式传输源的 Amazon Kinesis 流。

目录

ResourceARN

Amazon Kinesis 流的 Amazon 资源名称（ARN）。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARN

可由 Amazon Kinesis Analytics 代入以访问流的 IAM 角色的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisStreamsInputUpdate

更新应用程序输入配置时，提供有关 Amazon Kinesis 流作为流式传输源的信息。

目录

ResourceARNUpdate

要读取的输入 Amazon Kinesis 流的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARNUpdate

可由 Amazon Kinesis Analytics 代入以代表您访问流的 IAM 角色的 ARN。您需要向此角色授予必需的权限。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisStreamsOutput

在配置应用程序输出时，将 Amazon Kinesis 流标识为目标。您提供流的 Amazon 资源名称（ARN），以及用于让 Amazon Kinesis Analytics 代表您写入流的 IAM 角色 ARN。

目录

ResourceARN

要写入的目标 Amazon Kinesis 流的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

RoleARN

可由 Amazon Kinesis Analytics 代入以代表您对目标流进行写入的 IAM 角色的 ARN。您需要向此角色授予必需的权限。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisStreamsOutputDescription

描述了配置为应用程序输出目标的 Amazon Kinesis 流。

目录

ResourceARN

Amazon Kinesis 流的 Amazon 资源名称（ARN）。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARN

可由 Amazon Kinesis Analytics 代入以访问流的 IAM 角色的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

KinesisStreamsOutputUpdate

使用 UpdateApplication 操作更新输出配置时，提供有关配置为目标的 Amazon Kinesis 流的信息。

目录

ResourceARNUpdate

要向其写入的目标 Amazon Kinesis 流的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARNUpdate

可由 Amazon Kinesis Analytics 代入以代表您访问流的 IAM 角色的 ARN。您需要向此角色授予必需的权限。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

LambdaOutput

在配置应用程序输出时，将 AWS Lambda 函数标识为目标。您提供函数的 Amazon 资源名称（ARN），以及用于让 Amazon Kinesis Analytics 代表您写入函数的 IAM 角色 ARN。

目录

ResourceARN

要向其写入的目标 Lambda 函数的 Amazon 资源名称（ARN）。

> **Note**
>
> 要指定相比最新版本较早的 Lambda 函数版本，请在 Lambda 函数 ARN 中包括 Lambda 函数版本。有关 Lambda ARN 的更多信息，请参阅示例 ARN：AWS Lambda

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

RoleARN

可由 Amazon Kinesis Analytics 代入以代表您对目标函数进行写入的 IAM 角色的 ARN。您需要向此角色授予必需的权限。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

LambdaOutputDescription

描述了配置为应用程序输出目标的 AWS Lambda 函数。

目录

ResourceARN

目标 Lambda 函数的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARN

可由 Amazon Kinesis Analytics 代入以对目标函数进行写入的 IAM 角色的 ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

LambdaOutputUpdate

使用 UpdateApplication 操作更新输出配置时，提供有关配置为目标的 AWS Lambda 函数的信息。

目录

ResourceARNUpdate

目标 Lambda 函数的 Amazon 资源名称 (ARN)。

> **Note**
> 要指定相比最新版本较早的 Lambda 函数版本，请在 Lambda 函数 ARN 中包括 Lambda 函数版本。有关 Lambda ARN 的更多信息，请参阅示例 ARN：AWS Lambda

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

RoleARNUpdate

可由 Amazon Kinesis Analytics 代入以代表您对目标函数进行写入的 IAM 角色的 ARN。您需要向此角色授予必需的权限。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK

- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

MappingParameters

如果在创建或更新应用程序时配置应用程序输入，请提供特定于流式源上的记录格式（如 JSON、CSV 或由某个分隔符分隔的记录字段）的其他映射信息。

目录

CSVMappingParameters

提供当记录格式使用带分隔符的格式（如 CSV）时的其他映射信息。

类型：CSVMappingParameters 对象

必需：否

JSONMappingParameters

提供当 JSON 是流式源上的记录格式时的其他映射信息。

类型：JSONMappingParameters 对象

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

Output

描述应用程序输出配置，在其中您标识应用程序内部流以及希望应用程序内部流数据写入到的目标。目标可以是 Amazon Kinesis 流或 Amazon Kinesis Firehose 传输流。

有关应用程序可以写入的目标数的限制以及其他限制，请参阅限制。

目录

DestinationSchema

描述将记录写入目标时的数据格式。有关更多信息，请参阅配置应用程序输出。

类型：DestinationSchema 对象

必需：是

Name

应用程序内部流的名称。

类型：字符串

长度限制：最小长度为 1。最大长度为 32。

必需：是

KinesisFirehoseOutput

将 Amazon Kinesis Firehose 传输流标识为目标。

类型：KinesisFirehoseOutput 对象

必需：否

KinesisStreamsOutput

将 Amazon Kinesis 流标识为目标。

类型：KinesisStreamsOutput 对象

必需：否

LambdaOutput

将 AWS Lambda 函数标识为目标。

类型：LambdaOutput 对象

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

OutputDescription

描述应用程序输出配置，其中包括应用程序内部流名称和写入流数据的目标。目标可以是 Amazon Kinesis 流或 Amazon Kinesis Firehose 传输流。

目录

DestinationSchema

用于将数据写入目标的数据格式。

类型：DestinationSchema 对象

必需：否

KinesisFirehoseOutputDescription

描述配置为输出写入目标的 Amazon Kinesis Firehose 传输流。

类型：KinesisFirehoseOutputDescription 对象

必需：否

KinesisStreamsOutputDescription

描述配置为输出写入目标的 Amazon Kinesis 流。

类型：KinesisStreamsOutputDescription 对象

必需：否

LambdaOutputDescription

描述配置为输出写入目标的 AWS Lambda 函数。

类型：LambdaOutputDescription 对象

必需：否

Name

配置为输出的应用程序内部流的名称。

类型：字符串

长度限制：最小长度为 1。最大长度为 32。

必需：否

OutputId

输出配置的唯一标识符。

类型：字符串

长度限制：最小长度为 1。长度上限为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

OutputUpdate

描述 `OutputId` 标识的输出配置的更新。

目录

OutputId

标识要更新的特定输出配置。

类型：字符串

长度限制：最小长度为 1。长度上限为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：是

DestinationSchemaUpdate

描述将记录写入目标时的数据格式。有关更多信息，请参阅配置应用程序输出。

类型：DestinationSchema 对象

必需：否

KinesisFirehoseOutputUpdate

描述作为输出目标的 Amazon Kinesis Firehose 传输流

类型：KinesisFirehoseOutputUpdate 对象

必需：否

KinesisStreamsOutputUpdate

描述作为输出目标的 Amazon Kinesis 流。

类型：KinesisStreamsOutputUpdate 对象

必需：否

LambdaOutputUpdate

描述作为输出目标的 AWS Lambda 函数。

类型：LambdaOutputUpdate 对象

必需：否

NameUpdate

如果要为此输出配置指定不同的应用程序内部流，请使用此字段指定新的应用程序内部流名称。

类型：字符串

长度限制：最小长度为 1。最大长度为 32。

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

RecordColumn

描述流式源中的每个数据元素到应用程序内部流中的相应列的映射。

还用于描述引用数据来源的格式。

目录

Name

在应用程序内部输入流或引用表中创建的列的名称。

类型：字符串

必需：是

SqlType

在应用程序内部输入流或引用表中创建的列的类型。

类型：字符串

长度限制：最小长度为 1。

必需：是

Mapping

对流输入中数据元素的引用，或者引用数据来源。如果 RecordFormatType 是 JSON，则此元素是必需的。

类型：字符串

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK

- 适用于 Ruby V3 的 AWS SDK

RecordFormat

描述应该应用的记录格式和相关映射信息，以便在流中架构化记录。

目录

RecordFormatType

记录格式的类型。

类型：字符串

有效值：`JSON | CSV`

必需：是

MappingParameters

如果在创建或更新应用程序时配置应用程序输入，请提供特定于流式源上的记录格式（如 JSON、CSV 或由某个分隔符分隔的记录字段）的其他映射信息。

类型：MappingParameters 对象

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

ReferenceDataSource

描述引用数据来源，方式是提供源信息（S3 桶名称和对象键名称）、创建的结果应用程序内部表名称以及要将 Amazon S3 对象中的对象素映射到应用程序内部表的所需架构。

目录

ReferenceSchema

描述流式源中的数据的格式，以及每个数据元素如何映射到在应用程序内部流中创建的相应列。

类型：SourceSchema 对象

必需：是

TableName

要创建的应用程序内部表的名称。

类型：字符串

长度限制：最小长度为 1。最大长度为 32。

必需：是

S3ReferenceDataSource

标识 S3 桶和包含引用数据的对象。此外，还标识可由 Amazon Kinesis Analytics 代入以代表您读取此对象的 IAM 角色。Amazon Kinesis Analytics 应用程序仅加载一次引用数据。如果数据更改，您可以调用 `UpdateApplication` 操作来触发将数据重新加载到应用程序。

类型：S3ReferenceDataSource 对象

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK

- 适用于 Ruby V3 的 AWS SDK

ReferenceDataSourceDescription

描述为应用程序配置的引用数据来源。

目录

ReferenceId

引用数据源的 ID。这是当您使用 AddApplicationReferenceDataSource 操作向应用程序添加引用数据源时，Amazon Kinesis Analytics 分配的 ID。

类型：字符串

长度限制：最小长度为 1。长度上限为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：是

S3ReferenceDataSourceDescription

提供 S3 存储桶名称，即包含引用数据的对象键名称。它还提供了 IAM 角色的 Amazon 资源名称 (ARN)，Amazon Kinesis Analytics 可担任此角色读取 Amazon S3 对象以及填充应用程序内部引用表。

类型：S3ReferenceDataSourceDescription 对象

必需：是

TableName

由特定引用数据来源配置创建的应用程序内部表名称。

类型：字符串

长度限制：最小长度为 1。最大长度为 32。

必需：是

ReferenceSchema

描述流式源中的数据的格式，以及每个数据元素如何映射到在应用程序内部流中创建的相应列。

类型：SourceSchema 对象

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

ReferenceDataSourceUpdate

更新应用程序的引用数据来源配置时，此对象会提供所有更新的值 (例如源存储桶名称和对象键名称)、创建的应用程序内部表名称，以及要将 Amazon S3 对象中的数据映射到所创建应用程序内部引用表的更新映射信息。

目录

ReferenceId

正在更新的引用数据来源的 ID。您可以使用 DescribeApplication 操作来获取此值。

类型：字符串

长度限制：最小长度为 1。长度上限为 50。

模式：`[a-zA-Z0-9_.-]+`

必需：是

ReferenceSchemaUpdate

描述流式源中的数据的格式，以及每个数据元素如何映射到在应用程序内部流中创建的相应列。

类型：SourceSchema 对象

必需：否

S3ReferenceDataSourceUpdate

描述 S3 存储桶名称、对象键名称和 IAM 角色，Amazon Kinesis Analytics 可以担任此角色以代表您读取 Amazon S3 对象并填充应用程序内部引用表。

类型：S3ReferenceDataSourceUpdate 对象

必需：否

TableNameUpdate

此更新创建的应用程序内部表名称。

类型：字符串

长度限制：最小长度为 1。最大长度为 32。

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

S3Configuration

提供 Amazon S3 数据源的描述，包括 S3 桶的 Amazon 资源名称 (ARN)、用于访问存储桶的 IAM 角色的 ARN 以及包含数据的 Amazon S3 对象的名称。

目录

BucketARN

包含数据的 S3 存储桶的名称。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

FileKey

包含数据的对象的名称。

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

必需：是

RoleARN

用于访问数据的角色的 IAM ARN。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

S3ReferenceDataSource

标识 S3 桶和包含引用数据的对象。此外，还标识可由 Amazon Kinesis Analytics 代入以代表您读取此对象的 IAM 角色。

Amazon Kinesis Analytics 应用程序仅加载一次引用数据。如果数据更改，您可以调用 UpdateApplication 操作来触发将数据重新加载到应用程序。

目录

BucketARN

S3 桶的 Amazon 资源名称（ARN）。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

FileKey

包含引用数据的对象键名称。

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

必需：是

ReferenceRoleARN

可由此服务代入以代表您读取数据的 IAM 角色的 ARN。此角色必须具有对象上的 `s3:GetObject` 操作权限以及允许 Amazon Kinesis Analytics 服务委托人代入此角色的信任策略。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

S3ReferenceDataSourceDescription

提供存储桶名称，以及存储引用数据的对象键名称。

目录

BucketARN

S3 桶的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

FileKey

Amazon S3 对象键名称

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

必需：是

ReferenceRoleARN

IAM 角色的 ARN，Amazon Kinesis Analytics 可以担任此角色以代表您读取 Amazon S3 对象并填充应用程序内部引用表。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：是

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

S3ReferenceDataSourceUpdate

描述 S3 存储桶名称、对象键名称和 IAM 角色，Amazon Kinesis Analytics 可以担任此角色以代表您读取 Amazon S3 对象并填充应用程序内部引用表。

目录

BucketARNUpdate

S3 桶的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

FileKeyUpdate

对象键名称。

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

必需：否

ReferenceRoleARNUpdate

IAM 角色的 ARN，Amazon Kinesis Analytics 可以担任此角色读取 Amazon S3 对象并填充应用程序内部。

类型：字符串

长度限制：最小长度为 1。最大长度为 2048。

模式：`arn:.*`

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

SourceSchema

描述流式源中的数据的格式，以及每个数据元素如何映射到在应用程序内部流中创建的相应列。

目录

RecordColumns

RecordColumn 对象的列表。

类型：RecordColumn 对象数组

数组成员：最少 1 项。最多 1000 项。

必需：是

RecordFormat

指定流式源上的记录的格式。

类型：RecordFormat 对象

必需：是

RecordEncoding

指定流式源中的记录的编码。例如，UTF-8。

类型：字符串

模式：UTF-8

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

Tag

您可以定义并分配给 AWS 资源的键值对（值可选）。如果您指定的标签已经存在，则标签值将替换为您在请求中指定的值。请注意，应用程序标签的最大数量包括系统标签。用户定义的应用程序标签的最大数量为 50。有关更多信息，请参阅使用标记。

目录

Key

键值标签的键。

类型：字符串

长度限制：最小长度为 1。最大长度为 128。

必需：是

Value

键值标签的值。值是可选的。

类型：字符串

长度约束：最小长度为 0。长度上限为 256。

必需：否

另请参阅

有关在特定语言的 AWS 软件开发工具包中使用此 API 的更多信息，请参阅以下内容：

- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

Amazon Kinesis Data Analytics 文档历史记录

下表列出了自 Amazon Kinesis Data Analytics 上一次发布以来对文档所做的重要更改。

- API 版本：2015-08-14
- 文档最近更新时间：2019 年 5 月 8 日

更改	说明	日期
标记 Kinesis Data Analytics 应用程序	使用应用程序标记来确定每个应用程序的成本，控制访问，或用于用户定义的目的。有关更多信息，请参阅 使用标记。	2019 年 5 月 8 日
使用 AWS CloudTrail 记录 Kinesis Data Analytics API 调用	Amazon Kinesis Data Analytics 已与 AWS CloudTrail 集成，后者是一种服务，提供 Kinesis Data Analytics 中用户、角色或 AWS 服务所执行操作的记录。有关更多信息，请参阅 使用 AWS CloudTrail。	2019 年 3 月 22 日
Kinesis Data Analytics 在法兰克福区域可用	Kinesis Analytics 现在在欧洲地区（法兰克福）可用。有关更多信息，请参阅 和端点：Kinesis Data Analytics。	2018 年 7 月 18 日
在控制台中使用引用数据	现在，您可以在控制台中使用应用程序引用数据。有关更多信息，请参阅 示例：在 应用程序中添加引用数据。	2018 年 7 月 13 日
窗口式查询示例	适用于窗口和聚合的示例应用程序 有关更多信息，请参阅 示例：窗口和聚合。	2018 年 7 月 9 日

更改	说明	日期
测试应用程序	测试对应用程序架构和代码的更改的指导。有关更多信息，请参阅 测试应用程序。	2018 年 7 月 3 日
预处理数据示例应用程序	REGEX_LOG_PARSE、REGEX_REPLACE 和 DateTime 运算符的其他代码示例。有关更多信息，请参阅 示例：转换数据。	2018 年 5 月 18 日
返回的行和 SQL 代码的大小增加	返回的行的大小限制增加为 512 KB，应用程序中 SQL 代码的大小限制增加为 100 KB。有关更多信息，请参阅 Limits。	2018 年 5 月 2 日
使用 Java 和 .NET 的 AWS Lambda 函数示例	创建 Lambda 函数以预处理记录或作为应用程序目标的代码示例。有关更多信息，请参阅 创建 Lambda 函数以进行预处理 和 为应用程序目标创建 &LAM; 函数：	2018 年 3 月 22 日
新的 HOTSPOTS 函数	查找和返回有关数据中相对密集的区域的信息。有关更多信息，请参阅 示例：检测流上的热点 (HOTSPOTS 函数)。	2018 年 3 月 19 日
Lambda 函数作为目标	将分析结果发送到作为目标的 Lambda 函数。有关更多信息，请参阅 使用 Lambda 函数作为输出。	2017 年 12 月 20 日

更改	说明	日期
新的 RANDOM_CUT_FOREST_WITH_EXPLANATION 函数	了解在数据流中哪些字段会产生异常评分。有关更多信息，请参阅 示例：检测数据异常和获取说明 (RANDOM_CUT_FOREST_WITH_EXPLANATION 函数)。	2017 年 11 月 2 日
静态数据上的架构发现	为 Amazon S3 存储桶中存储的静态数据运行架构查找。有关更多信息，请参阅 针对静态数据使用架构发现功能。	2017 年 10 月 6 日
Lambda 预处理功能	在分析之前，使用 AWS Lambda 预处理输入流中的记录。有关更多信息，请参阅 使用 Lambda 函数预处理数据。	2017 年 10 月 6 日
自动扩展应用程序	使用自动扩展来自动增加应用程序的数据吞吐量。有关更多信息，请参阅 自动扩展应用程序以提高吞吐量。	2017 年 9 月 13 日
多个应用程序内部输入流	通过多个应用程序内部流提高应用程序吞吐量。有关更多信息，请参阅 并行处理输入流以增加吞吐量。	2017 年 6 月 29 日
使用 Kinesis Data Analytics AWS Management Console 的指南	在 Kinesis Data Analytics 控制台中使用架构编辑器和 SQL 编辑器编辑推断架构和 SQL 代码。有关更多信息，请参阅 步骤 4：(可选) 使用控制台编辑架构和 SQL 代码。	2017 年 4 月 7 日

更改	说明	日期
公开发行版	Amazon Kinesis Data Analytics 开发人员指南公开发行版。	2016 年 8 月 11 日
预览版	Amazon Kinesis Data Analytics 开发人员指南预览版。	2016 年 1 月 29 日

AWS 术语表

有关最新的 AWS 术语，请参阅《AWS 词汇表参考》中的 AWS 词汇表。