Amazon Kinesis Analytics

Developer Guide

Amazon Kinesis Analytics: Developer Guide

Table of Contents

What Is Amazon Kinesis Analytics?

With Amazon Kinesis Analytics, you can process and analyze streaming data using standard SQL. The service enables you to quickly author and run powerful SQL code against streaming sources to perform time series analytics, feed real-time dashboards, and create real-time metrics.

To get started with Amazon Kinesis Analytics, you create a Amazon Kinesis Analytics application that continuously reads and processes streaming data. The service supports ingesting data from Amazon Kinesis Streams and Amazon Kinesis Firehose streaming sources. Then, you author your SQL code using the interactive editor and test it with live streaming data. You can also configure destinations where you want Amazon Kinesis Analytics to persist the results. Amazon Kinesis Analytics supports Amazon Kinesis Firehose (Amazon S3, Amazon Redshift, and Amazon Elasticsearch Service), and Amazon Kinesis Streams as destinations.

When Should I Use Amazon Kinesis Analytics?

Amazon Kinesis Analytics enables you to quickly author SQL code that continuously reads, processes, and stores data in near real time. Using standard SQL queries on the streaming data, you can construct applications that transform and gain insights into your data. Following are some of example scenarios for using Amazon Kinesis Analytics:

- **Generate time-series analytics** – You can calculate metrics over time windows, and then stream values to Amazon S3 or Amazon Redshift through an Amazon Kinesis Firehose delivery stream.
- **Feed real-time dashboards** – You can send aggregated and processed streaming data results downstream to feed real-time dashboards.
- **Create real-time metrics** – You can create custom metrics and triggers for use in real-time monitoring, notifications, and alarms.

Are You a First-time User of Amazon Kinesis Analytics?

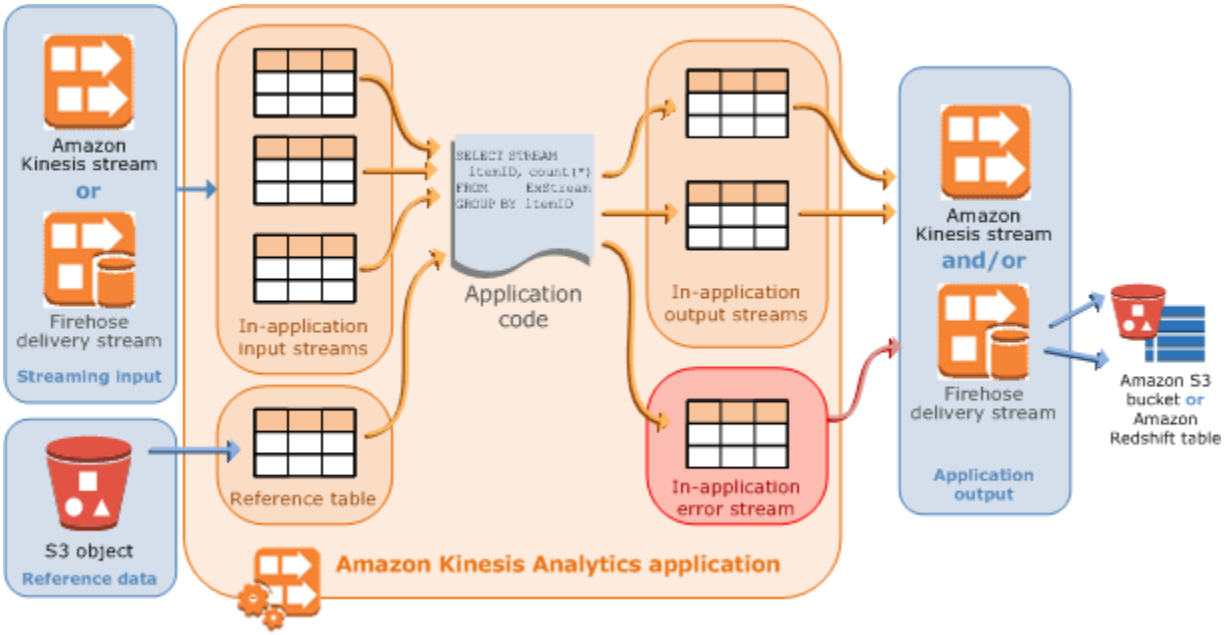
If you are a first-time user of Amazon Kinesis Analytics, we recommend that you read the following sections in order:

1. **Read the How It Works section of this guide.** This section introduces various Amazon Kinesis Analytics components that you work with to create an end-to-end experience. For more information, see Amazon Kinesis Analytics: How It Works (p. 3).
2. **Try the Getting Started Exercises.** For more information, see Getting Started (p. 17).
3. **Explore the streaming SQL concepts.** For more information, see Streaming SQL Concepts (p. 31).
4. **Try additional examples.** For more information, see Example Amazon Kinesis Analytics Applications (p. 43).

Amazon Kinesis Analytics: How It Works

An *application* is the primary resource in Amazon Kinesis Analytics that you can create in your account. You can create and manage applications using the console or the Amazon Kinesis Analytics API. Amazon Kinesis Analytics provides API operations to manage applications. For a list of API operations, see Actions (p. 101).

Amazon Kinesis Analytics applications continuously read and process streaming data in real-time. You write application code using SQL to process the incoming streaming data and produce output. Then, Amazon Kinesis Analytics writes the output to a configured destination. The following diagram illustrates a typical application architecture.



Each application has a name, description, version ID, and status. Amazon Kinesis Analytics assigns a version ID when you first create an application. This version ID is updated when you update any application configuration. For example, if you add an input configuration, add or delete a reference data source, or add or delete output configuration, or update application code, Amazon Kinesis Analytics updates the current application version ID. Amazon Kinesis Analytics also maintains timestamps when an application was created and last updated.

In addition to these basic properties, each application consists of the following:

- **Input** – The streaming source for your application. You can select either an Amazon Kinesis stream or a Firehose delivery stream as the streaming source. In the input configuration, you map the streaming source to an in-application input stream. The in-application stream is like a continuously updating table upon which you can perform the SELECT and INSERT SQL operations. In your application code you can create additional in-application streams to store intermediate query results.

  You can optionally partition a single streaming source in multiple in-application input streams to improve the throughput. For more information, see Limits (p. 83) and Configuring Application Input (p. 5).

  Amazon Kinesis Analytics provides a timestamp column in each application stream called Timestamps and the ROWTIME Column (p. 32). You can use this column in time-based windowed queries. For more information, see Windowed Queries (p. 35).

  You can optionally configure a reference data source to enrich your input data stream within the application. It results in an in-application reference table. You must store your reference data as an object in your S3 bucket. When the application starts, Amazon Kinesis Analytics reads the S3 object and creates an in-application table. For more information, see Configuring Application Input (p. 5).

- **Application code** – A series of SQL statements that process input and produce output. You can write SQL statements against in-application streams, reference tables, and you can write JOIN queries to combine data from both of these sources.

  In its simplest form, application code can be a single SQL statement that selects from a streaming input and inserts results into a streaming output. It can also be a series of SQL statements where output of one feeds into the input of the next SQL statement. Further, you can write application code to split an input stream into multiple streams and then apply additional queries to process these streams. For more information, see Application Code (p. 10).

- **Output** – In application code, query results go to in-application streams. In your application code, you can create one or more in-application streams to hold intermediate results. You can then optionally configure application output to persist data in the in-application streams, that hold your application output (also referred to as in-application output streams), to external destinations. External destinations can be a Firehose delivery stream or an Amazon Kinesis stream. Note the following about these destinations:

  - You can configure a Firehose delivery stream to write results to Amazon S3, Amazon Redshift, or Amazon ES.

- You can also write application output to a custom destination, instead of Amazon S3 or Amazon Redshift. To do that, you specify an Amazon Kinesis stream as the destination in your output configuration. Then, you configure AWS Lambda to poll the stream and invoke your Lambda function. Your Lambda function code receives stream data as input. In your Lambda function code, you can write the incoming data to your custom destination. For more information, see Using AWS Lambda with Amazon Kinesis Analytics.

For more information, see Configuring Application Output (p. 11).

In addition, note the following:

- Amazon Kinesis Analytics needs permissions to read records from a streaming source and write application output to the external destinations. You use IAM roles to grant these permissions.

- Amazon Kinesis Analytics automatically provides an in-application error stream for each application. If your application has issues while processing certain records, for example because of a type mismatch or late arrival, that record will be written to the error stream. You can configure application output to direct Amazon Kinesis Analytics to persist the error stream data to an external destination for further evaluation. For more information, see Error Handling (p. 12).

- Amazon Kinesis Analytics ensures that your application output records are written to the configured destination. It uses an "at least once" processing and delivery model, even in the event of an application interruption for various reasons. For more information, see Delivery Model for Persisting Application Output to External Destination (p. 12).

Topics

Configuring Application Input

Topics

Your Amazon Kinesis Analytics application can receive input from a single streaming source and, optionally, use one reference data source. For more information, see Amazon Kinesis Analytics: How It Works (p. 3). The sections in this topic describe the application input sources.

Configuring a Streaming Source

At the time you create an application, you specify a streaming source. You can also modify and input after you create the application. Amazon Kinesis Analytics supports the following streaming sources for your application:

- An Amazon Kinesis stream
- An Amazon Kinesis Firehose delivery stream

Amazon Kinesis Analytics continuously polls the streaming source for new data and ingests it in in-application streams according to the input configuration. Your application code can query the in-application stream. As part of input configuration you provide the following:

- **Streaming source** – You provide the Amazon Resource Name (ARN) of the stream and an IAM role that Amazon Kinesis Analytics can assume to access the stream on your behalf.

- **In-application stream name prefix** – When you start the application, Amazon Kinesis Analytics creates the specified in-application stream. In your application code, you access the in-application stream using this name.

  You can optionally map a streaming source to multiple in-application streams. For more information, see Limits (p. 83). In this case, Amazon Kinesis Analytics creates the specified number of in-application streams with names as follows: *prefix*_001, *prefix*_002, and *prefix*_003. By default, Amazon Kinesis Analytics maps the streaming source to one in-application stream called *prefix*_001.

  There is a limit on the rate that you can insert rows in an in-application stream. Therefore, Amazon Kinesis Analytics supports multiple such in-application streams to enable you to bring records into your application at a much faster rate. If you find your application is not keeping up with the data in the streaming source, you can add units of parallelism to improve performance.

- **Mapping schema** – You describe the record format (JSON, CSV) on the streaming source, and describe how each record on the stream maps to columns in the in-application stream that is created. This is where you provide column names and data types.

  Note
  Amazon Kinesis Analytics adds quotation marks around the identifiers (stream name and column names) when creating the input in-application stream. When querying this stream and the columns, you must specify them in quotation marks using the exact same casing (matching lowercase and uppercase letters exactly). For more information about identifiers, see Identifiers in the *Amazon Kinesis Analytics SQL Reference*.

You can create application and configure inputs in the Amazon Kinesis Analytics console. The console then makes the necessary API calls. You can configure application input when you create a new application API or add input configuration to an existing application. For more information, see CreateApplication (p. 108) and AddApplicationInput (p. 102). The following is the input configuration part of the `Createapplication` API request body:

```
"Inputs": [
      {
          "InputSchema": {
              "RecordColumns": [
                  {
                      "IsDropped": boolean,
                      "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"
        }
    ]
```

Configuring a Reference Source

You can also optionally add a reference data source to an existing application to enrich the data coming in from streaming sources. You must store reference data as an object in your S3 bucket. When the application starts, Amazon Kinesis Analytics reads the S3 object and creates an in-application reference table. Your application code can then join it with an in-application stream.

You store reference data in the S3 object using supported formats (CSV, JSON). For example, suppose your application performs analytics on stock orders. Assume the following record format on the streaming source:

```
Ticker, SalePrice, OrderId

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

In this case, you might then consider maintaining a reference data source to provide details for each stock ticker, such as company name:

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

Amazon Kinesis Analytics provides the following APIs to manage reference data sources.

- AddApplicationReferenceDataSource (p. 106)
- UpdateApplication (p. 128)

Note
Amazon Kinesis Analytics console does not support managing reference data sources for your applications. You can use the AWS CLI to add reference data source to your application. For an example, see Example: Adding Reference Data to an Amazon Kinesis Analytics Application (p. 59).

Note the following:

- If the application is running, Amazon Kinesis Analytics creates an in-application reference table, and then loads the reference data immediately.
- If the application is not running (for example, it's in the ready state), Amazon Kinesis Analytics only saves the updated input configuration. When the application starts running, Amazon Kinesis Analytics loads the reference data in your application as a table.

If you want to refresh the data after Amazon Kinesis Analytics creates the in-application reference table, perhaps because you updated the S3 object or you want to use different S3 object, you must explicitly call the UpdateApplication (p. 128). Amazon Kinesis Analytics does not refresh the in-application reference table automatically.

There is a limit on the size of the S3 object that you can create as a reference data source. For more information, see Limits (p. 83). If the object size exceeds the limit, Amazon Kinesis Analytics can't load the data. The application state appears as running, but the data is not being read.

When you add a reference data source, you provide the following information:

- **S3 bucket and object key name** – In addition to bucket name and object key, you also provide an IAM role that Amazon Kinesis Analytics can assume to read the object on your behalf.

- **In-application reference table name** – Amazon Kinesis Analytics creates this in-application table and populates it by reading the S3 object. This is the table name you specify in your application code.

- **Mapping schema** – You describe the record format (JSON, CSV), encoding of data stored in the S3 object. You also describe how each data element maps to columns in the in-application reference table.

The following shows the request body in the `AddApplicationReferenceDataSource` API request.

```
{
    "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"
    }
}
```

Using the Schema Discovery Feature and Related Editing

Providing an input schema that describes how records on the streaming input map to in-application stream can be cumbersome and error prone. You can use the DiscoverInputSchema (p. 120) API (called the *discovery API*) to infer a schema. Using random samples of records on the streaming source, the API can infer a schema (that is, column names, data types, and position of the data element in the incoming data).

> **Note**
> You can use the discovery API only to infer a schema for a streaming source. It is not supported for inferring schema for a reference data source.

The console uses the same discovery feature. For a specified streaming source, the console shows the inferred schema. Using the console, you can also update the schema, such as change column names, data types, etc. However, you need to make changes carefully to ensure that you do not create an invalid schema. For more information, see Error Handling (p. 12).

After you finalize a schema for your in-application stream, there are functions you can use to manipulate string and date time values. You can leverage these functions in your application code when working with rows in the resulting in-application stream. For more information, see Example: Manipulating Strings and Date Times (p. 44).

Schema Discovery Issues

What happens if Amazon Kinesis Analytics does not infer a schema for a given streaming source?

Amazon Kinesis Analytics will infer your schema for common formats, such as CSV and JSON, which are UTF-8 encoded. Amazon Kinesis Analytics supports any UTF-8 encoded records including raw text like application logs and records with custom column and row delimiter. You can define as schema manually using the schema editor in the console (or using the API) if Amazon Kinesis Analytics does not infer a schema.

If you your data does not follow a pattern which you can specify using the schema editor, you can define a schema as a single column of type VARCHAR(N), where N is the largest number of characters you expect your record to include. From there, you can use string and date time manipulation to structure your data after it is in an in-application stream. More information on how to do this is found in the String and Date Time manipulation section. For examples, see Example: Manipulating Strings and Date Times (p. 44).

Application Code

Application code is a series of SQL statements that process input and produce output. These SQL statements operate on in-application streams and reference tables. For more information, see Amazon Kinesis Analytics: How It Works (p. 3).

In relational databases, you work with tables, using INSERT statements to add records and the SELECT statement to query the data. In Amazon Kinesis Analytics, you work with streams. You can write a SQL statement to query these streams. The results of querying one in-application stream are always sent to another in-application stream. When performing complex analytics, you might create several in-application streams to hold the results of intermediate analytics. And then finally, you configure application output to persist results of the final analytics (from one or more in-application streams) to external destinations. In summary, the following is a typical pattern for writing application code:

- The SELECT statement is always used in the context of an INSERT statement. That is, when you select rows, you insert results into another in-application stream.
- The INSERT statement is always used in the context of a pump. That is, you use pumps to write to an in-application stream.

The following example application code reads records from one in-application (SOURCE_SQL_STREAM_001) stream and write it to another in-application stream (DESTINATION_SQL_STREAM). You can insert records to in-application streams using pumps, as shown following:

```
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";
```

The identifiers that you specify for stream names and column names follow standard SQL conventions. For example, if you put quotation marks around an identifier, it will make the identifier case-sensitive. If you don't, the identifier will default to uppercase. For more information about identifiers, see Identifiers in the *Amazon Kinesis Analytics SQL Reference*.

Your application code can consist of many SQL statements. For example:

- You can write SQL queries in a sequential manner where the result of one SQL statement feeds into the next SQL statement.
- You can also write SQL queries that run independent of each other. For example, you can write two SQL statements that query the same in-application stream, but send output into different in-applications streams. You can then query the newly created in-application streams independently.

You can create in-application streams to save intermediate results. You insert data in in-application streams using pumps. For more information, see In-Application Streams and Pumps (p. 31).

If you add a in-application reference table, you can write SQL to join data in in-application streams and reference tables. For more information, see Example: Adding Reference Data to an Amazon Kinesis Analytics Application (p. 59).

According to the application's output configuration, Amazon Kinesis Analytics writes data from specific in-application streams to the external destination according to the application's output configuration. Make sure that your application code writes to the in-application streams specified in the output configuration.

For more information, see the following topics:

- Streaming SQL Concepts (p. 31)
- Amazon Kinesis Analytics SQL Reference

Configuring Application Output

In your application code, you write the output of SQL statements to one or more in-application streams. You can optionally add output configuration to your application to persist everything written to an in-application stream to an external destination such as an Amazon Kinesis stream or a Firehose delivery stream.

There is a limit on the number of external destinations you can persist an application output. For more information, see Limits (p. 83).

> **Note**
> We recommend that you use one external destination to persist in-application error stream data so you can investigate the errors.

In each of these output configurations, you provide the following:

- **In-application stream name** – This is the stream that you want to persist to an external destination.
- **External destination** – You can persist data to an Amazon Kinesis stream or a Firehose delivery stream. You provide the Amazon Resource Name (ARN) of the stream and an IAM role that Amazon Kinesis Analytics can assume write to the stream on your behalf. You also describe the record format (JSON, CSV) to Amazon Kinesis Analytics to use when writing to the external destination.

Amazon Kinesis Analytics looks for the in-application stream that you specified in the output configuration (note that the stream name is case-sensitive and must match exactly). You should make sure that your application code creates this in-application stream.

You can configure the application output using the console. The console makes the API call to save the configuration. The following JSON fragment shows the `Outputs` section in the `CreateApplication` request body.

```
"Outputs": [
   {
       "how-it-works-outputchema": {
           "RecordFormatType": "string"
       },
       "KinesisFirehoseOutput": {
           "ResourceARN": "string",
```

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

If Amazon Kinesis Analytics service is not able to write to the streaming destination, the service continues to try indefinitely. This creates back pressure and your application will fall behind. And, if this is not resolved, your application will eventually stop processing new data.

Delivery Model for Persisting Application Output to External Destination

Amazon Kinesis Analytics uses an "at least once" delivery model for application output to the configured destinations. When an application is running, Amazon Kinesis Analytics takes internal checkpoints, which are points in time when output records were delivered to the destinations and there is no data loss. The service uses the checkpoints as needed to ensure your application output is delivered at least once to the configured destinations.

In a normal situation, your application processes incoming data continuously, and Amazon Kinesis Analytics writes the output to the configured destinations such as an Amazon Kinesis stream or a Firehose delivery stream.

However, your application can be interrupted, either by your choice or by some application configuration change that causes an interruption or failure, such as:

- You might choose to stop your application and restart it later.
- You delete the IAM role that Amazon Kinesis Analytics needs to write your application output to the configured destination. Without the IAM role, Amazon Kinesis Analytics does not have any permissions to write to the external destination on your behalf.
- Network outage or other internal service failures causing your application to stop running momentarily.

When your application starts working again, Amazon Kinesis Analytics ensures it continues to process and write output from a point before or equal to when the failure occurred, so that it does not miss delivering any of your application output to the configured destinations.

If you configured multiple destinations from same in-application stream, after the application recovers from failure, Amazon Kinesis Analytics resumes persisting output to the configured destinations from the last record that was delivered to the slowest destination. This might result in the same output record delivered more than once to other destinations. In this case you need to handle potential duplications in the destination externally.

Error Handling

Amazon Kinesis Analytics returns API or SQL errors directly to you. For more information about API operations, see Actions (p. 101). For more information about handling SQL errors, see Amazon Kinesis Analytics SQL Reference.

Amazon Kinesis Analytics reports runtime errors using an in-application error stream called `error_stream`.

Reporting Errors Using an In-Application Error Stream

Amazon Kinesis Analytics reports runtime errors to the in-application error stream called `error_stream`. For example:

- A record read from the streaming source does not conform to the input schema.
- Your application code specifies division by zero.
- The rows are out of order (for example, a record appears on the stream with a `ROWTIME` value that a user modified that causes a record to go out of order).

we recommend that you either handle these errors programmatically in your SQL code and/or persist the data on the error stream to an external destination such as a Firehose delivery stream that is configured to write data to an S3 bucket. This requires you add output configuration (see Configuring Application Output (p. 11)) to your application. For an example of how in-application error stream works, see Example: Explore the In-Application Error Stream (p. 76).

Granting Amazon Kinesis Analytics Permissions to Access Streaming Sources (Creating an IAM Role)

Amazon Kinesis Analytics needs permissions to read records from a streaming source that you specify in your application input configuration. Amazon Kinesis Analytics also needs permissions to write your application output to streams that you specify in your application output configuration.

You can grant these permissions by creating an IAM role that Amazon Kinesis Analytics can assume. Permissions that you grant to this role determine what Amazon Kinesis Analytics can do when the service assumes the role.

Note
The information in this section is useful if you want to create an IAM role yourself. When you create an application in the Amazon Kinesis Analytics console, the console can create an IAM role for you at that time. The console uses the following naming convention for IAM roles that it creates:

```
kinesis-analytics-ApplicationName
```

After the role is created, you can review the role and attached policies in the IAM console.

Each IAM role has two policies attached to it. In the trust policy, you specify who can assume the role. In the permissions policy (there can be one or more), you specify the permissions that you want to

grant to this role. The following sections describe these policies, which you can use when you create an IAM role.

Trust Policy

To grant Amazon Kinesis Analytics permissions to assume a role, you can attach the following trust policy to an IAM role:

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

Permissions Policy

If you are creating an IAM role to allow Amazon Kinesis Analytics to read from an application's streaming source, you must grant permissions for relevant read actions. Depending on your streaming source (for example, an Amazon Kinesis stream or a Firehose delivery stream), you can attach the following permissions policy.

Permissions Policy for Reading an Amazon Kinesis Stream

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputKinesis",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:aws-region:aws-account-
id:stream/inputStreamName"
            ]
        }
    ]
}
```

Permissions Policy for Reading a Firehose Delivery Stream

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

If you direct Amazon Kinesis Analytics to write output to external destinations in your application output configuration, you need to grant the following permission to the IAM role.

Permissions Policy for Writing to an Amazon Kinesis Stream

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

Permissions Policy for Writing to a Firehose Delivery Stream

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

```
}
```

Getting Started

This section provides topics to get you started using Amazon Kinesis Analytics. If you are new to Amazon Kinesis Analytics, we recommend that you review the concepts and terminology presented in Amazon Kinesis Analytics: How It Works (p. 3) before performing the steps in the Getting Started section.

Topics

Step 1: Set Up an AWS Account and Create an Administrator User

Before you use Amazon Kinesis Analytics for the first time, complete the following tasks:

1. Sign up for AWS (p. 17)
2. Create an IAM User (p. 18)

Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including Amazon Kinesis Analytics. You are charged only for the services that you use.

With Amazon Kinesis Analytics, you pay only for the resources you use. If you are a new AWS customer, you can get started with Amazon Kinesis Analytics for free. For more information, see AWS Free Usage Tier.

If you already have an AWS account, skip to the next task. If you don't have an AWS account, perform the steps in the following procedure to create one.

To create an AWS account

1. Open http://aws.amazon.com/, and then choose **Create an AWS Account**.

2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Note your AWS account ID because you'll need it for the next task.

Create an IAM User

Services in AWS, such as Amazon Kinesis Analytics, require that you provide credentials when you access them so that the service can determine whether you have permissions to access the resources owned by that service. The console requires your password. You can create access keys for your AWS account to access the AWS CLI or API. However, we don't recommend that you access AWS using the credentials for your AWS account. Instead, we recommend that you use AWS Identity and Access Management (IAM). Create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user that you created. You can then access AWS using a special URL and that IAM user's credentials.

If you signed up for AWS, but you haven't created an IAM user for yourself, you can create one using the IAM console.

The Getting Started exercises in this guide assume that you have a user (`adminuser`) with administrator privileges. Follow the procedure to create `adminuser` in your account.

To create an administrator user and sign in to the console

1. Create an administrator user called `adminuser` in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.
2. A user can sign in to the AWS Management Console using a special URL. For more information, How Users Sign In to Your Account in the *IAM User Guide*.

For more information about IAM, see the following:

- Identity and Access Management (IAM)
- Getting Started
- IAM User Guide

Next Step

Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 18)

Step 2: Set Up the AWS Command Line Interface (AWS CLI)

Follow the steps to download and configure the AWS Command Line Interface (AWS CLI).

Important
You don't need the AWS CLI to perform the steps in the Getting Started exercise. However, some of the exercises in this guide use the AWS CLI. You can skip this step and go to Step 3: Getting Started Exercise (Create an Amazon Kinesis Analytics Application) (p. 19), and then set up the AWS CLI later when you need it.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:

   - Getting Set Up with the AWS Command Line Interface
   - Configuring the AWS Command Line Interface

2. Add a named profile for the administrator user in the AWS CLI config file. You use this profile when executing the AWS CLI commands. For more information about named profiles, see Named Profiles in the *AWS Command Line Interface User Guide*.

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

   For a list of available AWS regions, see Regions and Endpoints in the *Amazon Web Services General Reference*.

3. Verify the setup by entering the following help command at the command prompt:

   ```
   aws help
   ```

Next Step

Step 3: Getting Started Exercise (Create an Amazon Kinesis Analytics Application) (p. 19)

Step 3: Getting Started Exercise (Create an Amazon Kinesis Analytics Application)

In this section, you create your first Amazon Kinesis Analytics application using the console.

> **Note**
> We suggest that you review the Amazon Kinesis Analytics: How It Works (p. 3) section before trying the Getting Started exercise.

For this Getting Started exercise, you can use the following console features:

- **Demo stream** – If you choose to use the demo stream, the console creates an Amazon Kinesis stream (`kinesis-analytics-demo-stream`) in your account. Then, the console runs a script that populates stock trade records on the stream. The sample data is stock prices by ticker symbol. You can use this stream as the streaming source for your application.

  > **Note**
  > The demo stream remains in your account. You can use it to test other examples in this guide. However, when you leave the console, the script that the console uses stops populating the data. When needed, the console provides the option to start populating the stream again.

- **Templates with example application code** – You use the template code that the console provides to perform simple analytics on the demo stream.

You use these features to quickly set up your first application as follows:

1. **Create an application** – You only need to provide a name. The console creates the application and the service sets the application state to `READY`.

2. **Configure input** – First you add a streaming source, the demo stream. You must create a demo stream in the console before you can use it. Then, the console takes a random sample of records on the demo stream and infers a schema for the in-application input stream that is created. The console names the in-application stream `SOURCE_SQL_STREAM_001`.

   The console uses the discovery API to infer the schema. If necessary, you can edit the inferred schema. For more information, see DiscoverInputSchema (p. 120). Amazon Kinesis Analytics uses this schema to create an in-application stream.

   When you start the application, Amazon Kinesis Analytics reads the demo stream continuously on your behalf and inserts rows in the `SOURCE_SQL_STREAM_001` in-application input stream.

3. **Specify application code** – You use a template (called **Continuous filter**) that provides the following code:

   ```
   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%';
   ```

   The application code queries the in-application stream `SOURCE_SQL_STREAM_001` and inserts the resulting rows in another in-application stream `DESTINATION_SQL_STREAM`, using pumps. For more information about this coding pattern, see Application Code (p. 10).

4. **Configuring output** – In this exercise, you don't configure any output. That is, you will not persist data in the in-application stream that your application creates to any external destination. Instead, you verify query results in the console. There are additional examples in this guide that show how to configure output. For example, see Example: Simple Alerts (p. 65).

   Important
   The exercise uses the US East (N. Virginia) Region (us-east-1) to set up the application. You can use any of the supported regions.

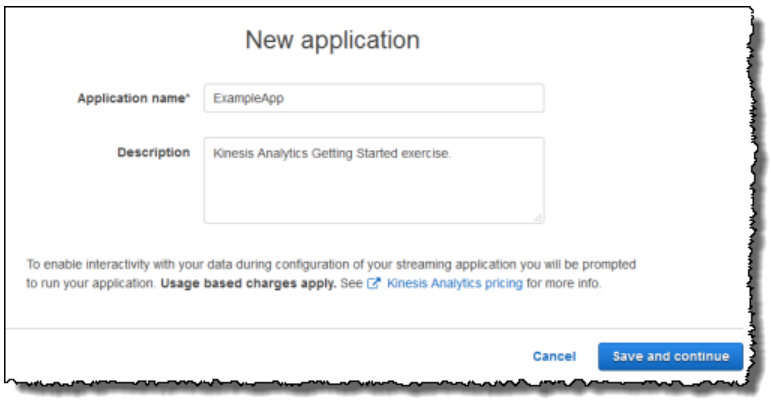
Next Step

Step 3.1: Create an Application (p. 20)

Step 3.1: Create an Application

In this section you create an Amazon Kinesis Analytics application. You configure application input in the next step.

1. Sign in to the AWS Management Console and open the Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create new application**.
3. On the **New application** page, type an application name, type a description, and then choose **Save and continue**.



This creates an Amazon Kinesis Analytics application with a status of READY. The console shows the application hub where you can configure input and output.

> **Note**
> To create an application, the CreateApplication (p. 108) operation requires only the application name. You can add input and output configuration after you create an application in the console.

In the next step, you configure input for the application. In the input configuration, you add a streaming data source to the application and discover a schema for an in-application input stream by sampling data on the streaming source.

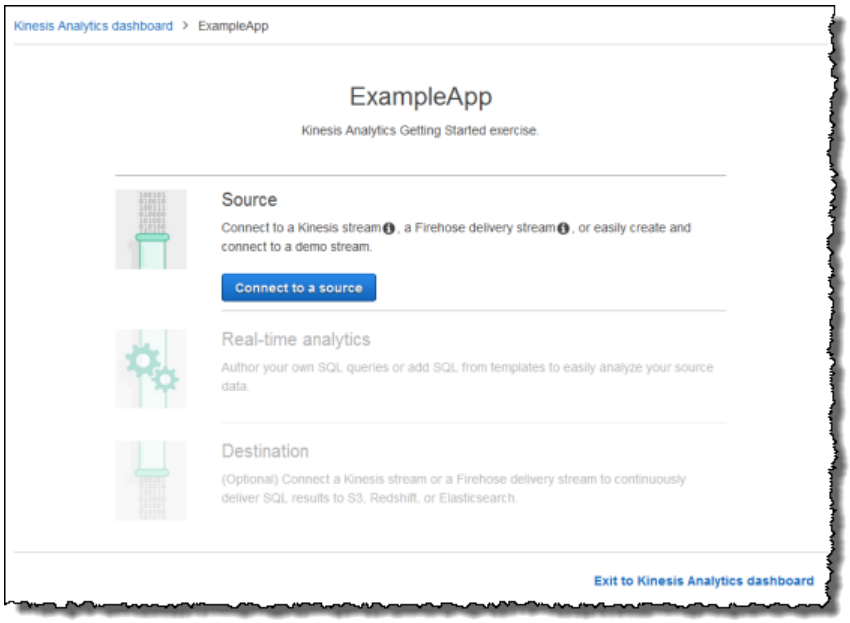
Next Step

Step 3.2: Configure Input (p. 21)

Step 3.2: Configure Input

Your application needs a streaming source. To help you get started, the console can create a demo stream (called `kinesis-analytics-demo-stream`). The console also runs a script that populates records in the stream. Add a streaming source to your application as follows:

1. On the application hub page in the console, choose **Connect to a source**.



2. On the page that appears, review the following:

   - **Source** section where you specify a streaming source for your application. You can select an existing stream source or create one. In this exercise, you create a new stream, the demo stream.

     By default the console names the in-application input stream that is created as `INPUT_SQL_STREAM_001`. For this exercise, keep this name as it appears.

     - **Stream reference name** – This shows the name of the in-application input stream, `SOURCE_SQL_STREAM_001`, that is created. You can change the name, but for this exercise use this name.

       In the input configuration, you map the demo stream to an in-application input stream that is created. When you start the application, Amazon Kinesis Analytics continuously reads the demo stream and insert rows in the in-application input stream. You query this in-application input stream in your application code.

     - **Permission to access the stream** – This is where you specify an IAM role. For more information, see Configuring a Streaming Source (p. 5). You have the option to choose an IAM role that exists in your account or create a new role. In this exercise, you create a new IAM role.

   After you provide all the information on this page, the console sends an update request (see UpdateApplication (p. 128)) to add the input configuration the application.

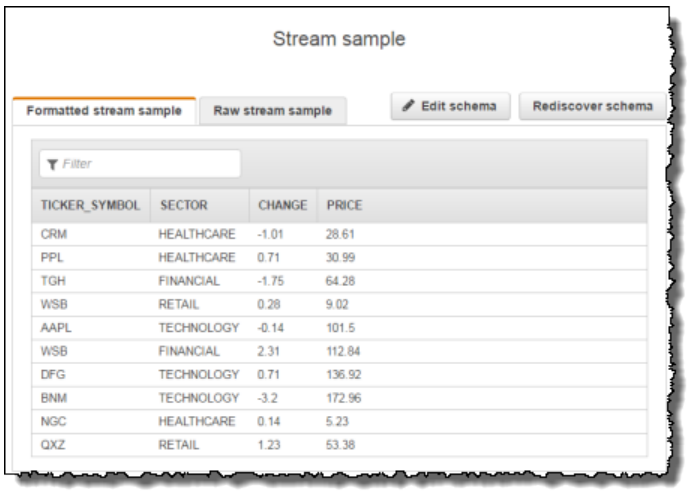
3. On the **Source** page, choose **Configure a new stream**.
4. Choose **Create demo stream**. The console does the following to configure the application input:

- Creates the Amazon Kinesis stream called `kinesis-analytics-demo-stream`.

- The console also runs a script that populates the stream with sample stock ticker data.

- Using the discovery API (see DiscoverInputSchema (p. 120)) infer a schema by reading sample records on the stream. This is the schema for the in-application input stream that is created. For more information, see Configuring Application Input (p. 5).
- Then, the console shows the inferred schema and the sample data it read from the streaming source to infer the schema.

The console displays the sample records on the streaming source.



Note the following:

- The **Raw stream sample** tab shows the raw stream records sampled by the discovery API (see DiscoverInputSchema (p. 120)) to infer the schema.
- The **Formatted stream sample** tab shows the tabular version of the data in the **Raw stream sample** tab.
- The **Edit schema** option allows you to edit the inferred schema. For this exercise, don't change the inferred schema.

  The **Rediscover schema** option allows you to request the console to run the discovery schema API again (see DiscoverInputSchema (p. 120)) and infer the schema.

5. Choose **Save and continue**.

   You now have an application with input configuration added to it. In the next step, you add SQL code to perform some analytics on the data in-application input stream.

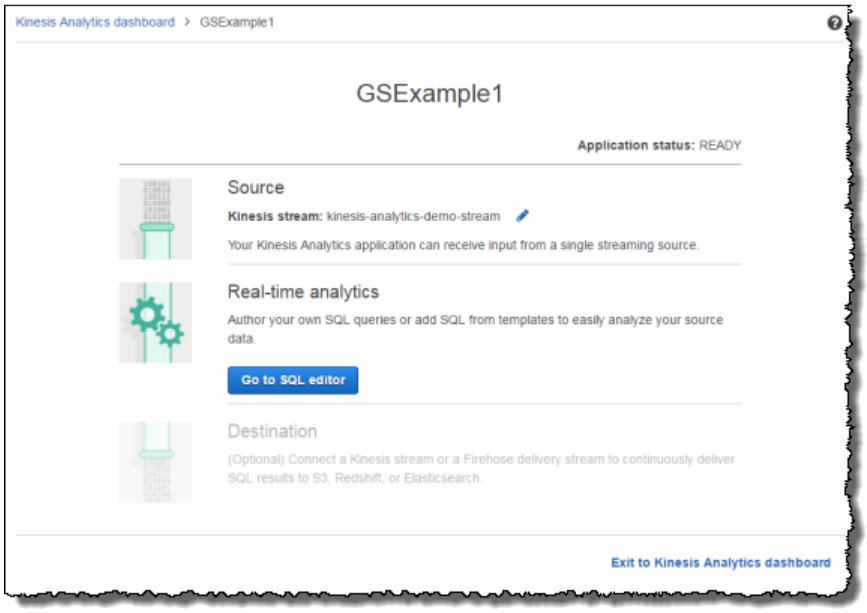
Next Step

Step 3.3: Add Real-Time Analytics (Add Application Code) (p. 24)

Step 3.3: Add Real-Time Analytics (Add Application Code)

You can write your own SQL queries against the in-application stream, but for this exercise you use one of the templates that provides sample code.

1. On the application hub page, choose **Go to SQL editor**.



2. In the **Would you like to start running "GSExample1"?** dialog box, choose **Yes, start application**.

   The console sends a request to start the application (see StartApplication (p. 125)), and then the SQL editor page appears.

3. The console opens the SQL editor page. Review the page, including the buttons (**Add SQL from templates**, **Save and run SQL**) and various tabs.
4. In the SQL editor, choose **Add SQL from templates**.
5. From the available template list, choose **Continuous filter**. Note that the sample code reads data from one in-application stream (the `WHERE` clause filers the rows) and inserts it in another in-application stream as follows:

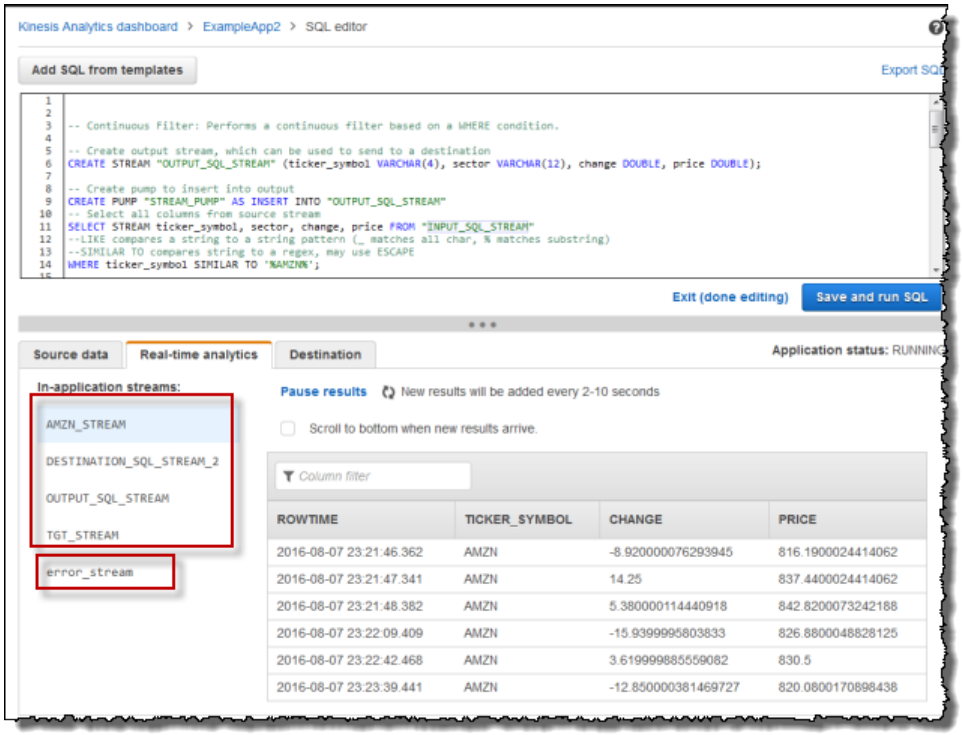
   - Creates the in-application stream `DESTINATION_SQL_STREAM`.
   - Creates a pump `STREAM_PUMP`, uses it to select rows from `SOURCE_SQL_STREAM_001` and insert them in the `DESTINATION_SQL_STREAM`.

6. Choose **Add this SQL to editor**.
7. Test the application code as follows:

   Remember, you already started the application (status is RUNNING). Therefore, Amazon Kinesis Analytics is already continuously reading from the streaming source and adding rows to the in-application stream `SOURCE_SQL_STREAM_001`.

a. In the SQL Editor, click **Save and run SQL**. The console first sends update request to save the application code. Then, the code continuously executes.

b. You can see the results in the **Real-time analytics** tab.



The SQL Editor has the following tabs:

- The **Source data** tab shows an in-application input stream that is mapped to the streaming source. Choose the in-application stream and you can see data coming in. Note the additional columns in the in-application input stream that were not specified in the input configuration. These include the following timestamp columns:

  - **ROWTIME** – Each row in an in-application stream has a special column called `ROWTIME`. It's the timestamp when Amazon Kinesis Analytics inserted the row in the first in-application stream (the in-application input stream that is mapped to the streaming source).

  - **Approximate_Arrival_Time** – Each Amazon Kinesis Analytics record includes a value called `Approximate_Arrival_Time`. It is the approximate arrival timestamp that is set when the streaming source successfully receives and stores the record. When Amazon Kinesis Analytics reads records from a streaming source, it fetches this column into the in-application input stream.

  These timestamp values are useful in windowed queries that are time-based. For more information, see Windowed Queries (p. 35).

- The **Real-time analytics** tab shows all the other in-application streams created by your application code. It also includes the error stream. Amazon Kinesis Analytics sends any rows it cannot process to the error stream. For more information, see Error Handling (p. 12).

  Choose the `DESTINATION_SQL_STREAM` to view the rows your application code inserted. Note again the additional columns that your application code did not create. These include the `ROWTIME` timestamp column. Amazon Kinesis Analytics simply copies these values from the source (`SOURCE_SQL_STREAM_001`).

- The **Destination** tab shows the external destination where Amazon Kinesis Analytics writes the query results. You have not configured any external destination for your application output yet.

Next Step

Step 3.4: (Optional) Update Application Code (p. 26)

Step 3.4: (Optional) Update Application Code

In this step, you explore how to update the application code.

1. Create another in-application stream as follows:

   - Create another in-application stream called `DESTINATION_SQL_STREAM_2`.
   - Create a pump, and then use it to insert rows in the newly created stream by selecting rows from the `DESTINATION_SQL_STREAM`.

   In the SQL Editor, append the following code to the existing application code:

   ```
   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";
   ```

   Save and run the code. Additional in-application streams appear on the **Real-time analytics** tab.

2. Create two in-application streams. Filter rows in the `SOURCE_SQL_STREAM_001` based on stock ticker, and then insert them in to these separate streams.

   Append the following SQL statements to your application code:

   ```
   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%';
```

Save and run the code. Notice additional in-application streams on the **Real-time analytics** tab.

Next Step

Step 3.5: (Optional) Configure Output (p. 27)

Step 3.5: (Optional) Configure Output

You now have your first working Amazon Kinesis Analytics application. In this exercise, you did the following:

- Created your first Amazon Kinesis Analytics application.

- Configured application input that identified the demo stream as the streaming source and mapped it to an in-application stream (`SOURCE_SQL_STREAM_001`) that is created. Amazon Kinesis Analytics continuously reads the demo stream and inserts records in the in-application stream.

- Your application code queried the `SOURCE_SQL_STREAM_001` and wrote output to another in-application stream called `DESTINATION_SQL_STREAM`.

Now you can optionally configure application output to write the application output to an external destination. That is, configure the application output to write records in the `DESTINATION_SQL_STREAM` to an external destination. For this exercise, we make this an optional step. You configure the destination in the next exercise.

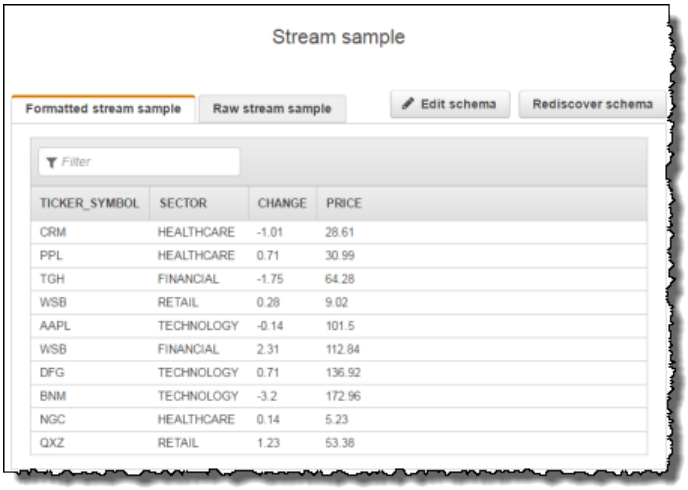
Next Step

Step 4: Console Feature Summary (p. 27).

Step 4: Console Feature Summary

This section summarizes some of the useful console features you used in the Getting Started exercise. These are used in several examples in this guides.

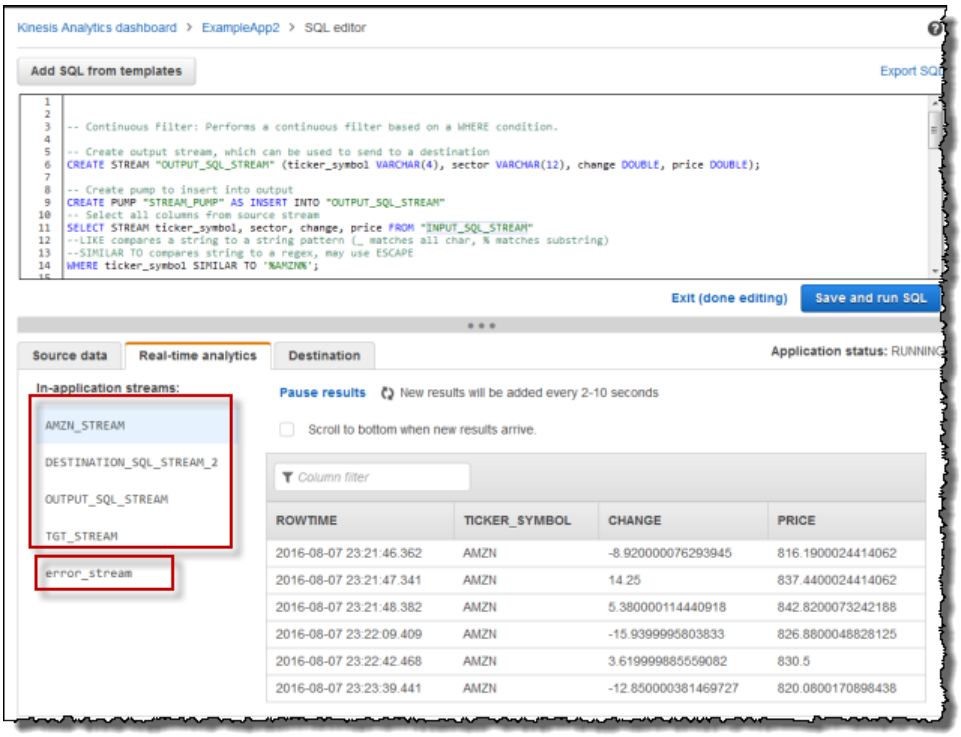
- **Demo stream** – An Amazon Kinesis Analytics application requires a streaming source. Several examples of SQL code in this guide use a demo stream that the console can create in your account. It is an Amazon Kinesis stream called `kinesis-analytics-demo-stream`. The console also runs a script that continuously add sample data (simulated stock trade records) on the stream as shown:



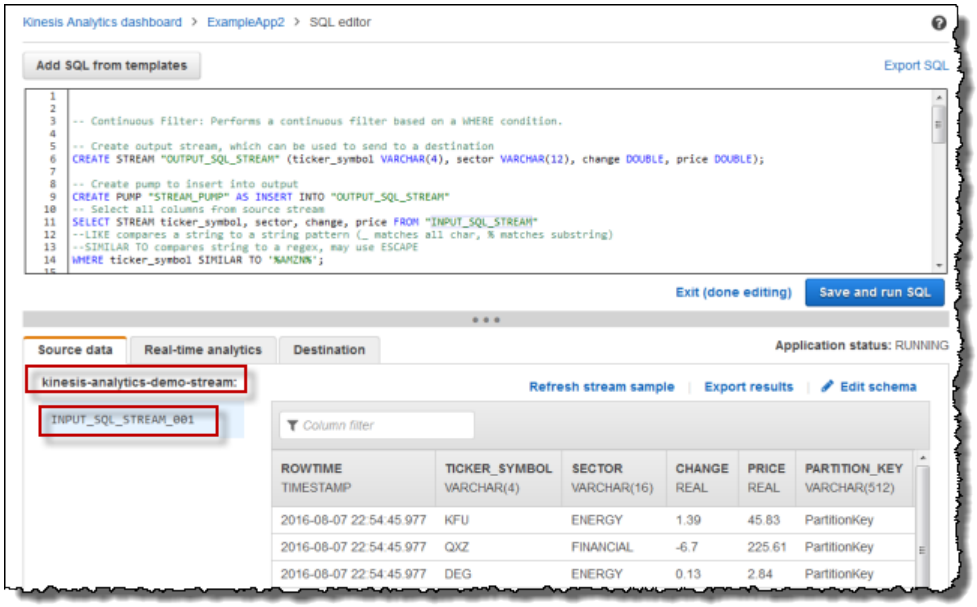
Note
The demo stream remains in your account. You can use it to test other examples in this guide. However, when you leave the console, the script that the console uses stops populating the data. When needed, the console provides the option to start populating the stream again.

- **Templates** – In the SQL editor, you can either author your own code yourself or choose **Add SQL from templates** to start with a template that provide example code. The example applications in this guide use some of these templates. For more information, see Example Amazon Kinesis Analytics Applications (p. 43).

- **Various Tabs in SQL editor** – Note the tabs in the SQL editor.



- **Source** tab – Identifies the streaming source and in-application input stream to which it maps (as the application input configuration).



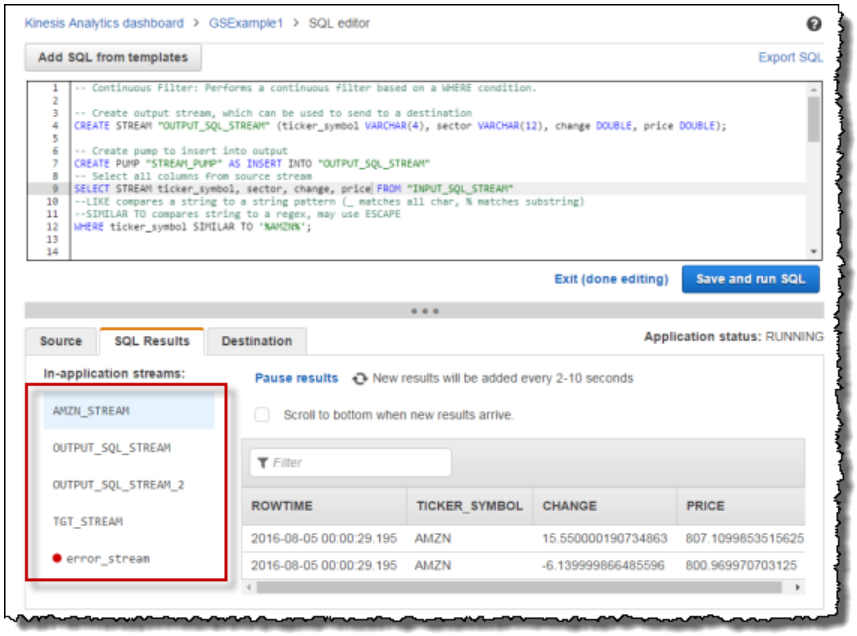
Note that Amazon Kinesis Analytics provides the following timestamp columns (you don't need to provide explicit mapping in your input configuration).

- **ROWTIME** – Each row in an in-application stream has a special column called `ROWTIME`. It is the timestamp when Amazon Kinesis Analytics inserted the row in the first in-application stream.

- **Approximate_Arrival_Time** – Records on your streaming source include the `Approximate_Arrival_Timestamp` column. It is the approximate arrival timestamp that is set when the streaming source successfully receives and stores the record. Amazon Kinesis Analytics fetches this column into the in-application input stream as `Approximate_Arrival_Time`. Amazon Kinesis Analytics provides this column only in the in-application input stream that is mapped to the streaming source.

These timestamp values are useful in windowed queries that are time-based. For more information, see Windowed Queries (p. 35).

- **Real-time analytics** tab – Shows all the in-application streams that your application code creates. This also includes the error stream (`error_stream`) that Amazon Kinesis Analytics provides for all applications.



- **Destination** tab – Enables you to configure application output, to persist in-application streams to external destinations. You can configure output to persist data in any of the in-application streams to external destinations. For more information, see Configuring Application Output (p. 11).

For additional examples, see Example Amazon Kinesis Analytics Applications (p. 43).

Streaming SQL Concepts

Amazon Kinesis Analytics implements the ANSI 2008 SQL standard with extensions. These extensions enable you to process streaming data. The following topics cover key streaming SQL concepts.

Topics

In-Application Streams and Pumps

When you configure application input, you map a streaming source to an in-application stream that is created. Data continuously flows from the streaming source into the in-application stream. An in-application stream works like a table that you can query using SQL statements, but it's called a stream because it represents continuous data flow.

> **Note**
> Do not confuse in-application streams with the Amazon Kinesis streams and Firehose delivery streams. In-application streams exist only in the context of an Amazon Kinesis Analytics application. Amazon Kinesis streams and Firehose delivery streams exist independent of your application, and you can configure them as a streaming source in your application input configuration or as a destination in output configuration.

You can also create additional in-application streams as needed to store intermediate query results. Creating an in-application stream is a two-step process. First, you create an in-application stream, and then you pump data into it. For example, suppose the input configuration of your application creates an in-application stream called `INPUTSTREAM`. In the following example, you create another stream (`TEMPSTREAM`), and then you pump data from `INPUTSTREAM` into it.

1. Create an in-application stream (`TEMPSTREAM`) with three columns, as shown following:

```
CREATE OR REPLACE STREAM "TEMPSTREAM" (
   "column1" BIGINT NOT NULL,
   "column2" INTEGER,
   "column3" VARCHAR(64));
```

The column names are specified in quotes, making them case-sensitive. For more information, see Identifiers in the Amazon Kinesis Analytics SQL Reference.

2. Insert data into the stream using a pump. A pump is a continuous insert query running that inserts data from one in-application stream to another in-application stream. The following statement creates a pump (`SAMPLEPUMP`) and inserts data into the `TEMPSTREAM` by selecting records from another stream (`INPUTSTREAM`).

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

You can have multiple writers insert into an in-application stream, and there can be multiple readers selected from the stream. You can think of an in-application stream as implementing a publish/subscribe messaging paradigm in which the data row, including time of creation and time of receipt, can be processed, interpreted, and forwarded by a cascade of streaming SQL statements, without having to be stored in a traditional RDBMS.

After an in-application stream is created, you can perform normal SQL queries.

> **Note**
> When querying streams, most SQL statements are bound using a row-based or time-based window. For more information, see Windowed Queries (p. 35).

You can also join streams. For examples of joining streams, see Streaming Data Operations: Stream Joins (p. 41).

Timestamps and the ROWTIME Column

In-application streams include a special column called `ROWTIME`. It stores a timestamp when Amazon Kinesis Analytics inserts a row in the first in-application stream. `ROWTIME` reflects the timestamp at which Amazon Kinesis Analytics inserted a record into the first in-application stream after reading from the streaming source. This `ROWTIME` value is then maintained throughout your application.

> **Note**
> When you pump records from one in-application stream into another, you don't need to explicitly copy the `ROWTIME` column, Amazon Kinesis Analytics copies this column for you.

Amazon Kinesis Analytics guarantees that the `ROWTIME` values are monotonically increased. You use this timestamp in time-based windowed queries. For more information, see Windowed Queries (p. 35).

You can access the ROWTIME column in your SELECT statement like any other columns in your in-application stream. For example:

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

Understanding Various Times in Streaming Analytics

In addition to `ROWTIME`, there are other types of times in real-time streaming applications. These are:

- **Event time** – The timestamp when the event occurred. This is also sometimes called the *client-side time*. It is often desirable to use this time in analytics because it is the time when an event occurred. However, many event sources, such as mobile phones and web clients, do not have reliable clocks, which can lead to inaccurate times. In addition, connectivity issues can lead to records appearing on a stream not in the same order the events occurred.

- **Ingest time** – The timestamp of when record was added to the streaming source. Amazon Kinesis Streams includes a field called `ApproximateArrivalTimeStamp` in every record that provides this timestamp. This is also sometimes referred to as the *server-side time*. This ingest time is often the close approximation of event time. If there is any kind of delay in the record ingestion to the stream, this can lead to inaccuracies, which are typically rare. Also, the ingest time is rarely out of order, but it can occur due to the distributed nature of streaming data. Therefore, Ingest time is a mostly accurate and in-order reflection of the event time.

- **Processing time** – The timestamp when Amazon Kinesis Analytics inserts a row in the first in-application stream. Amazon Kinesis Analytics provides this timestamp in the `ROWTIME` column that exists in each in-application stream. The processing time is always monotonically increasing, but it will not be accurate if your application falls behind (if an application falls behind, the processing time will not accurately reflect the event time). This `ROWTIME` is very accurate in relation to the wall clock, but it might not be the time when the event actually occurred.

As you can see from the preceding discussion, using each of these times in windowed queries that are time-based has advantages and disadvantages. We recommend you choose one or more of these times, and a strategy to deal with the relevant disadvantages based on your use case scenario.

> **Note**
> If you are using row-based windows, time is not an issue and you can ignore this section.

We recommend a two-window strategy that uses two time-based, both `ROWTIME` and one of the other times (ingest or event time).

- Use `ROWTIME` as the first window, which controls how frequently the query emits the results, as shown in the following example. It is not used as a logical time.
- Use one of the other times that is the logical time you want to associated with your analytics. This time represents when the event occurred. In the following example, the analytics goal is to group the records and return count by ticker.

The advantage of this strategy is that it can use a time that represents when the event occurred, and it can gracefully handle when your application falls behind or when events arrive out of order. If the

application falls behind when bringing records into the in-application stream, they are still grouped by the logical time in the second window. The query uses `ROWTIME` to guarantee the order of processing. Any records that are late (ingest timestamp shows earlier value compared to the `ROWTIME` value) are processed successfully too.

Consider the following query against the demo stream used in the Getting Started Exercise. The query uses the `GROUP BY` clause and emits ticker count in a one-minute tumbling window.

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

--CREATE OR REPLACE PUMP data into output
CREATE OR REPLACE PUMP "myOutputPUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      -- select the ingest time used in the GROUP BY clause
      SELECT STREAM FLOOR("SOURCE_SQL_STREAM_001".Approximate_Arrival_Time TO
 MINUTE) AS ingest_time,
                    Ticker_Symbol,
                    COUNT(*) AS symbol_count
      FROM "SOURCE_SQL_STREAM_001"
      GROUP BY Ticker_Symbol,
                -- use process time as a trigger, which can be different time
 window as the aggregate
                FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE),
                -- aggregate records based upon ingest time
                FLOOR("SOURCE_SQL_STREAM_001".Approximate_Arrival_Time TO
 MINUTE);
```

In `GROUP BY`, you first group the records based on `ROWTIME` in a one-minute window and then by `Approximate_Arrival_Time`.

Note that the timestamp values in the result are rounded to nearest minute. The first group result emitted by the query shows records in the first minute. The second group of results emitted shows records in the next minutes based on `ROWTIME`. The last record indicates that the application was late in bringing the record in the in-application stream (it shows a late `ROWTIME` value compared to the ingest timestamp).

```
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.
```

You can combine the results for a final accurate count per minute by pushing the results to a downstream database. For example, you can configure application output to persist the results to a Firehose delivery stream that can write to an Amazon Redshift table. After results are in an

Amazon Redshift table, you can query the Amazon Redshift table to compute the total count group by `Ticker_Symbol`. In the case of `ABC`, the total is accurate (6+1) even though a record arrived late.

Continuous Queries

A query over a stream executes continuously over streaming data. This continuous execution enables scenarios, such as the ability for applications to continuously query a stream and generate alerts.

In the Getting Started exercise, you have an in-application stream called `SOURCE_SQL_STREAM_001` that continuously receives stock prices from a demo stream (an Amazon Kinesis stream). Following is the schema:

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

Suppose you are interested in stock price changes greater than 15%. You can use the following query in your application code. This query runs continuously and emits records when a stock price change greater than 1% is detected.

```
SELECT STREAM TICKER_SYMBOL, PRICE
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  (ABS((CHANGE / (PRICE-CHANGE)) * 100)) > 1
```

Use the following procedure to set up an Amazon Kinesis Analytics application and test this query.

To test the query

1. Set up an application by following the Getting Started Exercise.
2. Replace the `SELECT` statement in the application code with the preceding `SELECT` query. The resulting application code is shown following:

```
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)) > 1;
```

Windowed Queries

SQL queries in your application code execute continuously over in-application streams. And, an in-application stream represents unbounded data that is flowing continuously through your application. Therefore, to get result sets from this continuously updating input, you often bound queries using a window defined in terms of time or rows. These are also called *windowed SQL*.

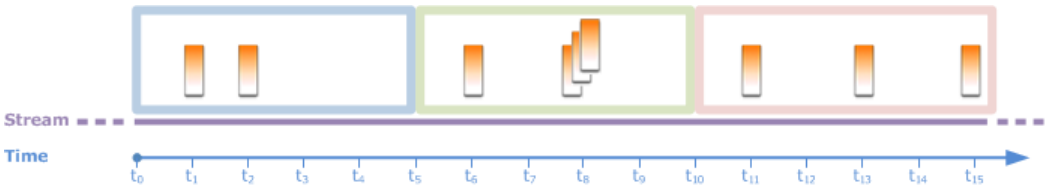
For a time-based windowed query, you specify the window size in terms of time (for example, a one-minute window). This requires a timestamp column in your in-application stream that is monotonically increasing (timestamp for a new row is greater than or equal to previous row). Amazon Kinesis Analytics provides such a timestamp column called `ROWTIME` for each in-application stream. You can use this column when specifying time-based queries. For your application, you might choose some other timestamp option. For more information, see Timestamps and the ROWTIME Column (p. 32).

For a row-based windowed query, you specify window size in terms of the number of rows.

You can specify a query to process records in a tumbling window or sliding window manner, depending on your application needs. For more information, see the following topics:

Tumbling Windows (Aggregations Using GROUP BY)

When a windowed query processes each window in a non-overlapping manner, the window is referred to as a *tumbling window*. In this case, each record on an in-application stream belongs to a specific window, and it's processed only once (when the query processes the window to which the record belongs).



For example, an aggregation query using a `GROUP BY` clause processes rows in a tumbling window. The demo stream in the Getting Started Exercise receives stock price data that is mapped to the in-application stream `SOURCE_SQL_STREAM_001` in your application, which has the following schema:

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

In your application code, suppose you want to find aggregate (min, max) prices for each ticker over a one-minute window. You can use the following query:

```
SELECT STREAM ROWTIME,
              Ticker_Symbol,
              MIN(Price) AS Price,
              MAX(Price) AS Price
FROM     "SOURCE_SQL_STREAM_001"
GROUP BY Ticker_Symbol,
         FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE);
```

This is an example of a windowed query that is time-based, the query groups records by `ROWTIME` values. For a per-minute basis reporting, the `FLOOR` function rounds down the `ROWTIME` values to the nearest minute.

This query is an example of a non-overlapping (tumbling) window. The `GROUP BY` clause groups records in a one-minute window and each record belongs to a specific window (no overlapping). The

query emits one output record per minute, providing the min/max ticker price recorded at the specific minute. This type of query is useful for generating periodic reports (in this example, each minute) from the input data stream.

To test the query

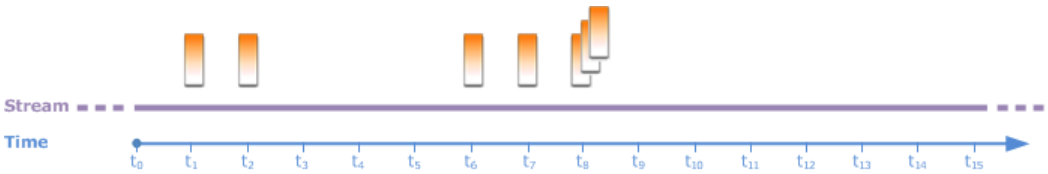
1. Set up an application by following the Getting Started Exercise.
2. Replace the `SELECT` statement in the application code by the preceding `SELECT` query. The resulting application code is shown following:

```
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,
             FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE);
```

Sliding Windows

Instead of grouping records using `GROUP BY`, you can define a window (time- or row-based). For example, you can do this by adding an explicit `WINDOW` clause. In this case, as the window slides with time, Amazon Kinesis Analytics emits an output when new records appear on the stream, by processing rows in the window. Note that windows can overlap in this type of processing, a record can be part of multiple windows and processed with the window. The following example illustrates the sliding window.

Consider a simple query that counts records on the stream. We assume a five-second window. In the following example stream, new records arriving at time t_1, t_2, t_6, t_7, and three records at time t_8 seconds.



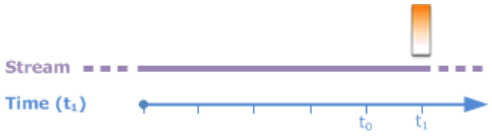
Keep the following in mind:

- We assume a five-second window. The five-second window slides continuously with the time.
- For every row that enters a window, an output row is emitted by the sliding window. Soon after the application starts, initially you see the query emit output for every new record that appears on the stream, even though it is not a five-second window yet. For example, the query emits output when a record appears in the first second and second second. Later, the query processes records in the five-second window.
- The windows slide with time and if an old record on the stream falls out of the window, the query will not emit any output unless there is also a new record on the stream in that five-second window.

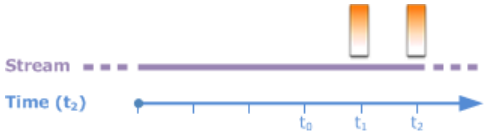
Suppose the query starts executing at t_0.

1. At the time t_0, the query starts. The query will not emit output (count value) because there are no records at this time.

2. At time t_1, a new record appears on the stream, and the query emit count value 1.

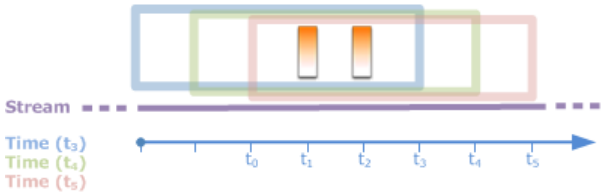


3. At time t_2, another record appears, and the query emits count 2.

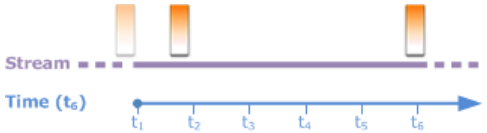


4. The five-second window slides with time.
   - At t_3, the sliding window t_3 to t_0.
   - At t4 (sliding window t_4 to t_0), and
   - At t_5 the sliding window t_5-t_0.

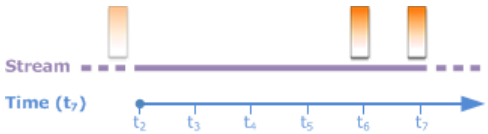
   At all of these times, the five-second window has the same records—there are no new records. Therefore, the query doesn't emit any output.



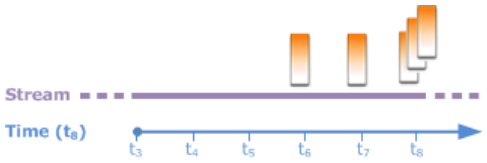
5. At time t_6, the five-second window is (t_6 to t_1), the query detects one new record at t_6 so it emits output 2. The record at t_1 is no longer in the window and it will not count.



6. At time t_7, the five-second window is t_7 to t_2, the query detects one new record at t_7 so it emits output 2. The record at t_2 is no longer in the five-second window and, therefore, is not counted.



7. At time t_8, the five-second window is t_8 to t_3, the query detects three new records, and therefore emits record count 5.



In summary, the window is a fixed size and slides with time. The query emits output when new records appear.

The following are example queries that use the `WINDOW` clause to define windows and perform aggregates. Because the queries don't specify `GROUP BY`, the query uses the sliding window approach to process records on the stream.

Example 1: Process a Stream Using a One-Minute Sliding Window

For example, consider the demo stream in the Getting Started exercise that populates the in-application stream, `SOURCE_SQL_STREAM_001`. The following is the schema:

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

Suppose you want your application to compute aggregates using a sliding one-minute window. That is, for each new record that appears on the stream, you want the application to emit an output by applying aggregates on records in the preceding one-minute window.

You can use the following time-based windowed query. The query uses the `WINDOW` clause to define the one-minute range interval. The `PARTITION BY` in the `WINDOW` clause groups records by ticker values within the sliding window.

```
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);
```

To test the query

1. Set up an application by following the Getting Started Exercise.
2. Replace the `SELECT` statement in the application code with the preceding `SELECT` query. The resulting application code is:

```
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);
```

Example 2: Query Applying Aggregates on a Sliding Window

The following query against the demo stream returns the average of the percent change in the price of each ticker in a ten-second window.

```
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);
```

To test the query

1. Set up an application by following the Getting Started Exercise.
2. Replace the `SELECT` statement in the application code with the preceding `SELECT` query. The resulting application code is:

```
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);
```

Example 3: Query Data from Multiple Sliding Windows on the Same Stream

You can write queries to emit output in which each column value is calculated using different sliding windows defined over the same stream.

In this example, the query emits output (that is, ticker, price, a2, and a10) for ticker symbols whose two-row moving average crosses the ten-row moving average. Note that the `a2` and `a10` column values are derived from two-row and ten-row sliding windows

```
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);
```

To test this query against the demo stream, follow the test procedure described in Example 1 (p. 39). Use the following application code that creates another in-application stream `DESTINATION_SQL_STREAM`.

Streaming Data Operations: Stream Joins

You can have multiple in-application streams in your application. You can write `JOIN` queries to correlate data arriving on these streams. For example, suppose you have the following in-application streams:

- **OrderStream** – Receives stock orders being placed.

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

- **TradeStream** – Receives resulting stock trades for those orders.

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

The following are `JOIN` query examples that correlate data on these streams.

Example 1: Report Orders Where There Are Trades within One Minute of the Order Being Placed

In this example, your query joins both the `OrderStream` and `TradeStream`. However, because we want only trades placed one minute after the orders, the query defines the one-minute window over the `TradeStream`. For information about windowed queries, see Sliding Windows (p. 37).

```
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 FOLLOWING) AS t
ON   o.orderId = t.orderId;
```

You can define the windows explicitly using the `WINDOW` clause and writing the preceding query as follows:

```
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 FOLLOWING)
```

When you include this query in your application code, the application code runs continuously. For each arriving record on the `OrderStream`, the application emits an output if there are trades within the one-minute window following the order being placed.

The join in the preceding query is an inner join where the query emits records in `OrderStream` for which there is a matching record in `TradeStream` (and vice versa). Using an outer join you can create another interesting scenario. Suppose you want stock orders for which there are no trades within one minute of stock order being placed, and trades reported within the same window but for some other orders. This is example of an *outer join*.

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

Example Amazon Kinesis Analytics Applications

This section provides examples of working with Amazon Kinesis Analytics. Some of these examples also provide step-by-step instructions for you to create an Amazon Kinesis Analytics application and test the setup.

Before you explore these walkthroughs, we recommend that you first review Amazon Kinesis Analytics: How It Works (p. 3) and Getting Started (p. 17).

Topics

Examples: Preprocessing Streams

There are times when your application code needs to preprocess the incoming records before performing any analytics. This can happen for various reasons, such as records not conforming the supported record formats that can result into unnormalized columns in in-application input streams. This section provides examples of how to use the available string functions to normalize data, how to extract information that you need from string columns, and so on. The section also points to date time functions that you might find useful.

Topics

Example: Manipulating Strings and Date Times

String Manipulation

Amazon Kinesis Analytics supports formats such as JSON and CSV for records on a streaming source. For details, see RecordFormat (p. 165). These records then map to rows in in-application stream as per the input configuration. For details, see Configuring Application Input (p. 5). The input configuration specifies how record fields in the streaming source map to columns in in-application stream.

This mapping works when records on the streaming source follow the supported formats, that results in an in-application stream with normalized data.

But, what if data on your streaming source does not conform to supported standards? For example, what if your streaming source contain data such as clickstream data, IoT sensors, and application logs? Consider these examples:

- **Streaming source contains application logs** – The application logs follow the standard Apache log format, and are written to the stream using JSON format.

```
{
   "Log":"192.168.254.30 - John [24/May/2004:22:01:02 -0700] "GET /icons/
apache_pb.gif HTTP/1.1" 304 0"
}
```

  For more information about the standard Apache log format, see Log Files on the Apache website.

- **Streaming source contains semi-structured data** – The following example shows two records. The `Col_E_Unstrucutured` field value is a series of comma-separated values.

```
{ "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"}
```

  There are five columns, the first four have string type values and the last column contains comma-separated values.

- Records on your streaming source contain URLs and you need a portion of the URL domain name for analytics.

```
{ "referrer" : "http://www.amazon.com"}
{ "referrer" : "http://www.stackoverflow.com" }
```

In such cases, the following two-step process generally works for creating in-application streams that contain normalized data:

1. Configure application input to map the unstructured field to a column of the `VARCHAR(N)` type in the in-application input stream that is created.
2. In your application code, use string functions to split this single column into multiple columns and then save the rows in another in-application stream. This in-application stream that your application code creates will have normalized data. You can then perform analytics on this in-application stream.

Amazon Kinesis Analytics provides string operations, standard SQL functions, and extensions to the SQL standard for working with string columns, including the following:

- **String operators** – Operators such as `LIKE` and `SIMILAR` are useful in comparing strings. For more information, see String Operators in the *Amazon Kinesis Analytics SQL Reference*.

- **SQL functions** – The following functions are useful when manipulating individual strings. For more information, see Scalar Functions in the *Amazon Kinesis Analytics SQL Reference*.
  - CHAR_LENGTH – Provides the length of a string.
  - LOWER/UPPER – Converts a string to lowercase or uppercase.
  - OVERLAY – Replace a portion of the first string argument (the original string) with the second string argument (the replacement string).
  - SUBSTRING – Extracts a portion of a source string starting at a specific position.
  - POSITION – Searches for a string within another string.

- **SQL Extensions** – These are useful for working with unstructured strings such as logs and URIs.
  - REGEX_LOG_PARSE – Parses a string based on default Java Regular Expression patterns.
  - FAST_REGEX_LOG_PARSER – Works similar to the regex parser, but takes several shortcuts to ensure faster results. For example, the fast regex parser stops at the first match it finds (known as *lazy semantics*).
  - W3C_Log_Parse – A function for quickly formatting Apache logs.
  - FIXED_COLUMN_LOG_PARSE – Parses fixed-width fields and automatically converts them to the given SQL types.
  - VARIABLE_COLUMN_LOG_PARSE – Splits an input string into fields separated by a delimiter character or a delimiter string.

For examples using these function, see the following topics:

- Example: String Manipulation (W3C_LOG_PARSE Function) (p. 45)
- Example: String Manipulation (VARIABLE_COLUMN_LOG_PARSE Function) (p. 48)
- Example: String Manipulation (SUBSTRING Function) (p. 50)

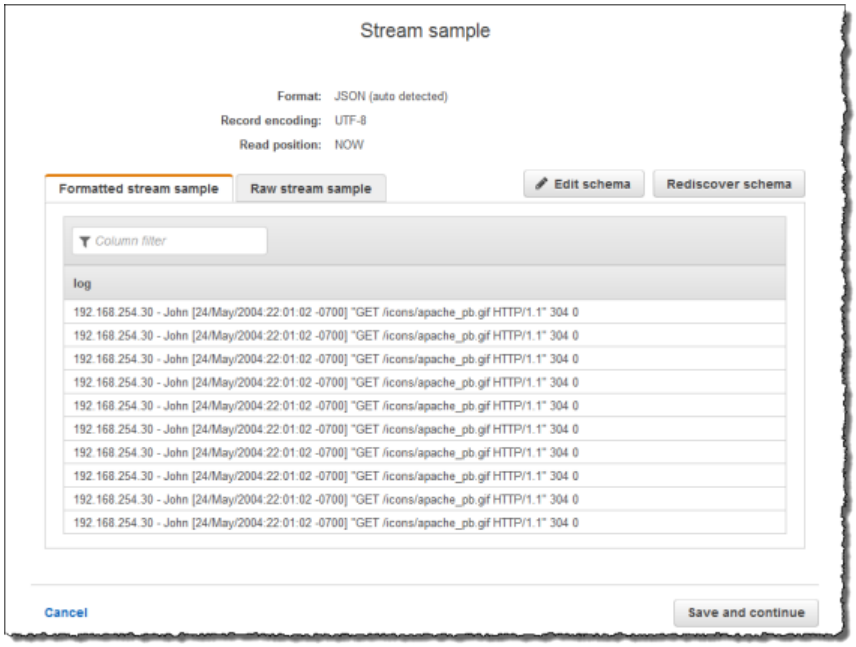
Example: String Manipulation (W3C_LOG_PARSE Function)

In this example, you write log records to an Amazon Kinesis stream. Example logs are shown following:

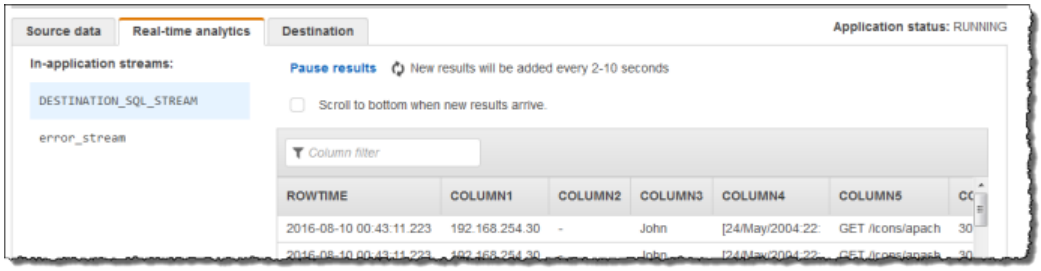
```
{"Log":"192.168.254.30 - John [24/May/2004:22:01:02 -0700] "GET /icons/
apache_pba.gif HTTP/1.1" 304 0"}
{"Log":"192.168.254.30 - John [24/May/2004:22:01:03 -0700] "GET /icons/
apache_pbb.gif HTTP/1.1" 304 0"}
```

```
{"Log":"192.168.254.30 - John [24/May/2004:22:01:04 -0700] "GET /icons/
apache_pbc.gif HTTP/1.1" 304 0"}
...
```

You then create an Amazon Kinesis Analytics application in the console, with the Amazon Kinesis stream as the streaming source. The discovery process reads sample records on the streaming source and infers an in-application schema with one column (log), as shown following:



Then, you use the application code with the `W3C_LOG_PARSE` function to parse the log, and create another in-application stream with various log fields in separate columns, as shown following:



Step 1: Create an Amazon Kinesis Stream

Create an Amazon Kinesis stream and populate log records as follows:

1. Sign in to the AWS Management Console and open the Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Kinesis Stream** and then create a stream with one shard.
3. Run the following Python code to populate sample log records. The Python code is simple, it continuously writes same log record to the stream.

```
import json
```

```
from boto import kinesis
import random

kinesis = kinesis.connect_to_region("us-east-1")
def getHighHeartRate():
    data = {}
    data['log'] = '192.168.254.30 - John [24/May/2004:22:01:02 -0700]
 "GET /icons/apache_pb.gif HTTP/1.1" 304 0'
    return data

while True:
        data = json.dumps(getHighHeartRate())
        print data
        kinesis.put_record("stream-name", data, "partitionkey")
```

Step 2: Create the Amazon Kinesis Analytics Application

Create an Amazon Kinesis Analytics application as follows:

1. Sign in to the AWS Management Console and open the Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create new application**, and specify an application name.
3. On the application hub, connect to the source.
4. On the **Source** page, do the following:
   - Select the stream that you created in the preceding section.
   - Choose the create IAM role option.
   - Wait for console to show the inferred schema and samples records used to infer the schema for the in-application stream created. Note that the inferred schema has only one column.
   - Choose **Save and continue**.
5. On the application hub, choose **Go to SQL editor**. To start the application, choose **yes** in the dialog box that appears.
6. In the SQL editor, write application code and verify the results as follows:
   - Copy the following application code and paste it into the editor.

```
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);
```

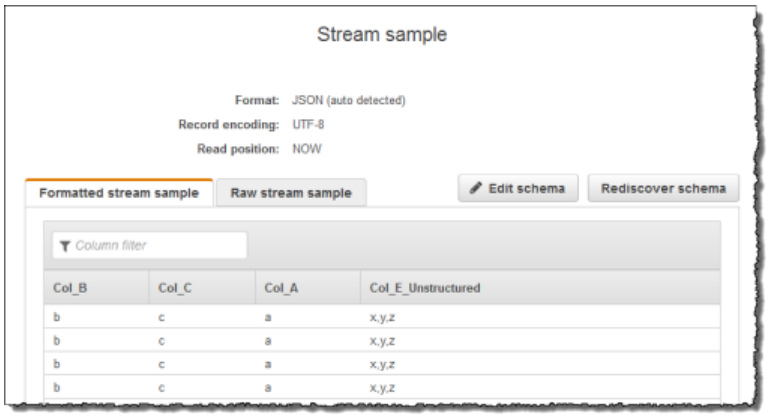
- Choose **Save and run SQL**. On the **Real-time analytics** tab you can see all of the in-application streams that the application created and verify the data.

Example: String Manipulation (VARIABLE_COLUMN_LOG_PARSE Function)

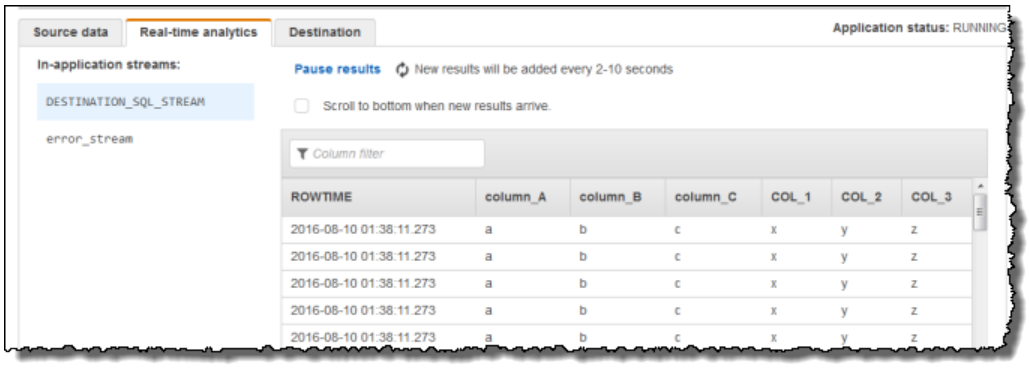
In this example, you write semi-structured records to an Amazon Kinesis stream. The example records are as follows:

```
{ "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"}
```

You then create an Amazon Kinesis Analytics application in the console, with the Amazon Kinesis stream as the streaming source. The discovery process reads sample records on the streaming source and infer an in-application schema with one column (log), as shown following:



Then, you use the application code with the `VARIABLE_COLUMN_LOG_PARSE` function to parse the comma-separated values, and insert normalized rows in another in-application stream, as shown following:



Step 1: Create an Amazon Kinesis Stream

Create an Amazon Kinesis stream and populate log records as follows:

1. Sign in to the AWS Management Console and open the Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Kinesis Stream** and then create a stream with one shard.
3. Run the following Python code to populate sample log records. The Python code is simple, it continuously writes same log record to the stream.

```
import json
from boto import kinesis
import random

kinesis = kinesis.connect_to_region("us-east-1")
def getHighHeartRate():
    data = {}
    data['Col_A'] = 'a'
    data['Col_B'] = 'b'
    data['Col_C'] = 'c'
    data['Col_E_Unstructured'] = 'x,y,z'
    return data

while True:
        data = json.dumps(getHighHeartRate())
        print data
        kinesis.put_record("teststreamforkinesisanalyticsapps", data,
 "partitionkey")
```

Step 2: Create the Amazon Kinesis Analytics Application

Create an Amazon Kinesis Analytics application as follows:

1. Sign in to the AWS Management Console and open the Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create new application**, and specify an application name.
3. On the application hub, connect to the source.
4. On the **Source** page, do the following:

   - Select the stream you created in the preceding section.

- Choose the create IAM role option.
- Wait for console to show the inferred schema and samples records used to infer the schema for the in-application stream created. Note that the inferred schema has only one column.
- Choose **Save and continue**.

5. On the application hub, choose **Go to SQL editor**. To start the application, choose **yes** in the dialog box that appears.
6. In the SQL editor, write application code and verify results:
   - Copy the following application code and paste it into the editor.

```
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;
```

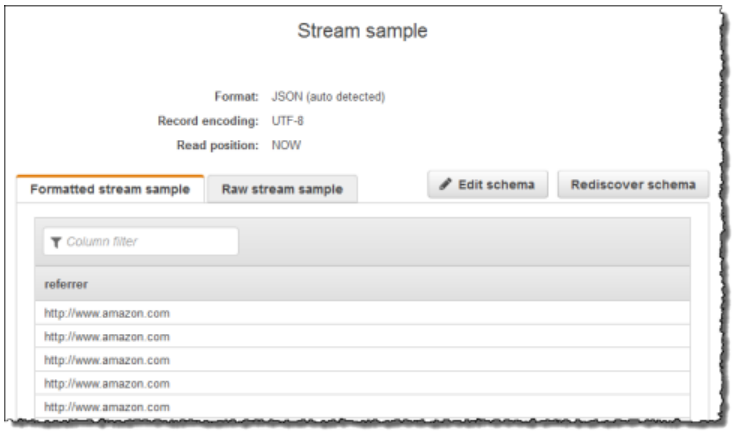
   - Choose **Save and run SQL**. On the **Real-time analytics** tab you can see all of the in-application streams that the application created and verify the data.

Example: String Manipulation (SUBSTRING Function)

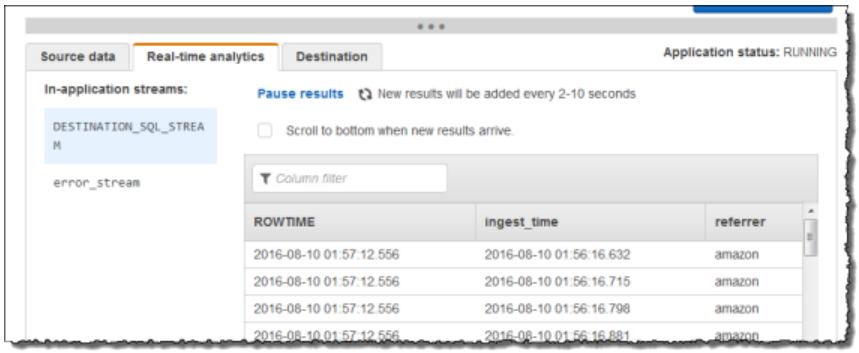
In this example, you write the following records to your an Amazon Kinesis stream.

```
{ "referrer" : "http://www.stackoverflow.com" }
{ "referrer" : "http://www.amazon.com"}
{ "referrer" : "http://www.amazon.com"}
...
```

You then create an Amazon Kinesis Analytics application in the console, with the Amazon Kinesis stream as the streaming source. The discovery process reads sample records on the streaming source and infers an in-application schema with one column (log) as shown.



Then, you use the application code with the `SUBSTRING` function to parse URL string to retrieve company name, and insert resulting data in another in-application stream, as shown following:



Step 1: Create an Amazon Kinesis Stream

Create an Amazon Kinesis stream and populate log records as follows:

1. Sign in to the AWS Management Console and open the Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Kinesis Stream**, and then create a stream with one shard.
3. Run the following Python code to populate sample log records. The Python code is simple, it continuously writes same log record to the stream.

```
import json
from boto import kinesis
import random

kinesis = kinesis.connect_to_region("us-east-1")
def getReferrer():
    data = {}
    data['referrer'] = 'http://www.amazon.com'
    return data

while True:
        data = json.dumps(getReferrer())
        print data
```

```
        kinesis.put_record("teststreamforkinesisanalyticsapps", data,
"partitionkey")
```

Step 2: Create the Amazon Kinesis Analytics Application

Create an Amazon Kinesis Analytics application as follows:

1. Sign in to the AWS Management Console and open the Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create new application**, and specify an application name.
3. On the application hub, connect to the source.
4. On the **Source** page, do the following:
   - Select the stream you created in the preceding section.
   - Choose the create IAM role option.
   - Wait for console to show the inferred schema and samples records used to infer the schema for the in-application stream created. Note that the inferred schema has only one column.
   - Choose **Save and continue**.
5. On the application hub, choose **Go to SQL editor**. To start the application, choose **yes** in the dialog box that appears.
6. In the SQL editor, write application code and verify the results as follows:
   - Copy the following application code and paste it into the editor.

```
-- 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";
```

   - Choose **Save and run SQL**. On the **Real-time analytics** tab you can see all of the in-application streams that the application created and verify the data.

Date Time Manipulation

Amazon Kinesis Analytics supports converting columns to timestamps. For example, you might want to use your own timestamp as part of a `GROUP BY` clause as another time-based window, in addition to the `ROWTIME` column. Amazon Kinesis Analytics provides operations and SQL functions for working with date and time fields.

- **Date and time operators** – You can perform arithmetic operations on dates, times, and interval data types. For more information, see Date, Timestamp, and Interval Operators in the *Amazon Kinesis Analytics SQL Reference*.

- **SQL Functions** – These include the following:
  - `EXTRACT( )` – Extracts one field from a date, time, timestamp, or interval expression.
  - `CURRENT_TIME` – Returns the time when the query executes (UTC).
  - `CURRENT_DATE` – Returns the date when the query executes (UTC).
  - `CURRENT_TIMESTAMP` – Returns the timestamp when the query executes (UTC).
  - `LOCALTIME` – Returns the current time when the query executes as defined by the environment on which Amazon Kinesis Analytics is running (UTC).
  - `LOCALTIMESTAMP` – Returns the current timestamp as defined by the environment on which Amazon Kinesis Analytics is running (UTC).

- **SQL Extensions** – These include the following:
  - `CURRENT_ROW_TIMESTAMP` – Returns a new timestamp for each row in the stream.
  - `TSDIFF` – Returns the difference of two timestamps in milliseconds.
  - `CHAR_TO_DATE` – Converts a string to a date.
  - `CHAR_TO_TIME` – Converts a string to time.
  - `CHAR_TO_TIMESTAMP` – Converts a string to a timestamp.
  - `DATE_TO_CHAR` – Converts a date to a string.
  - `TIME_TO_CHAR` – Converts a time to a string.
  - `TIMESTAMP_TO_CHAR` – Converts a timestamp to a string.

Most of the preceding SQL functions use a format to convert the columns. The format is flexible. For example, you can specify the format `yyyy-MM-dd hh:mm:ss` to convert an input string `2009-09-16 03:15:24` into a timestamp. For more information, Char To Timestamp(Sys) in the *Amazon Kinesis Analytics SQL Reference*.

Example: Streaming Source With Multiple Record Types

Topics

A common requirement in Extract, Transform and Load (ETL) applications is to process multiple record types on a streaming source. You can create Amazon Kinesis Analytics application to process these kinds of streaming sources. You do the following:

- First, you map the streaming source to an in-application input stream, similar to all other Amazon Kinesis Analytics applications.
- Then, in your application code you write SQL statements to retrieve rows of specific types from the in-application input stream, and insert them in separate in-application streams (you can create additional in-application streams in your application code).

In this exercise, you have a streaming source that receives records of two types (`Order` and `Trade` types). These are stock orders and corresponding trades. For each order, there can be zero or more trades. Example records of each type are shown following:

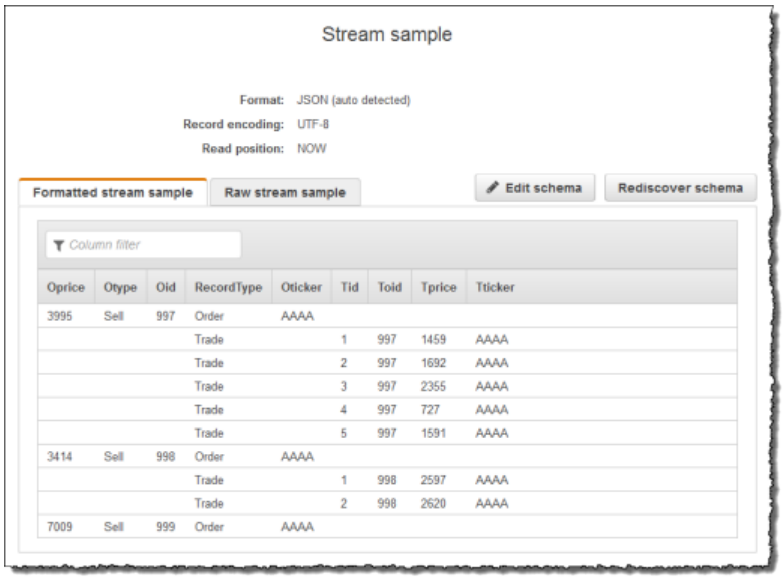
Order record

```
{"RecordType": "Order", "Oprice": 9047, "Otype": "Sell", "Oid": 3811,
 "Oticker": "AAAA"}
```

Trade record

```
{"RecordType": "Trade", "Tid": 1, "Toid": 3812, "Tprice": 2089, "Tticker":
 "BBBB"}
```

When you create an application using the console, the console displays the following inferred schema for the in-application input stream created. By default, console names this in-application stream as `SOURCE_SQL_STREAM_001`.



When you save the configuration, Amazon Kinesis Analytics continuously reads data from the streaming source and inserts rows in the in-application stream. You can now perform analytics on data in the in-application stream.

In this example, the application code you first create two additional in-application streams, `Order_Stream`, `Trade_Stream`. You then filter the rows from `SOURCE_SQL_STREAM_001` stream based on record type and insert them in the newly created streams using pumps. For information about this coding pattern, see Application Code (p. 10).

- Filter order and trade rows into separate in-application streams
  - Filter the order records in the `SOURCE_SQL_STREAM_001` and save the orders in the `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';
```

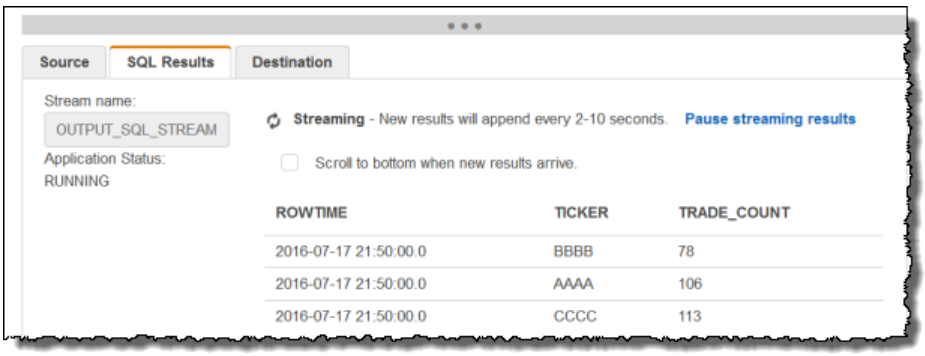
- Filter the trade records in the `SOURCE_SQL_STREAM_001` and save the orders in the `Trade_Stream`.

```
--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';
```

- Now you can perform additional analytics on these streams. In this example, you count number of trades by ticker in a one-minute tumbling window and save results to yet another stream, `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);
```

You see the result, as shown following:



Next Step

Step 1: Prepare (p. 56)

Step 1: Prepare

In this section, you create an Amazon Kinesis stream, and then populate order and trade records on the stream. This is your streaming source for the application you create in the next step.

Step 1.1: Create a Streaming Source

You can create an Amazon Kinesis stream using the console or the AWS CLI. The example assumes `OrdersAndTradesStream` as the stream name.

- Using the console – Sign in to the AWS Management Console and open the Amazon Kinesis console at https://console.aws.amazon.com/kinesis. Choose **Kinesis Stream**, and then create a stream with one shard.
- Using the AWS CLI – Use the following Amazon Kinesis `create-stream` CLI command to create the stream:

```
$ aws kinesis create-stream \
--stream-name OrdersAndTradesStream \
--shard-count 1 \
--region us-east-1 \
--profile adminuser
```

Step 1.2: Populate the Streaming Source

Run the following Python script to populate sample records on the `OrdersAndTradesStream`. If you created the stream with different name, update the Python code appropriately.

1. Install Python and `pip`.

   For information about installing Python, see the Python website.

   You can install dependencies using pip. For information about installing pip, see Installing on the pip website.
2. Run the following Python code. The `put-record` command in the code writes the JSON records to the stream.

```
import testdata
import json
from boto import kinesis
import random

kinesis = kinesis.connect_to_region("us-east-1")

def getOrderData(orderId, ticker):
    data = {}
    data['RecordType'] = "Order"
    data['Oid'] = orderId
    data['Oticker'] = ticker
    data['Oprice'] = random.randint(500, 10000)
    data['Otype'] = "Sell"
    return data
```

```
def getTradeData(orderId, tradeId, ticker, tradePrice):
    data = {}
    data['RecordType'] = "Trade"
    data['Tid'] = tradeId
    data['Toid'] = orderId
    data['Tticker'] = ticker
    data['Tprice'] = tradePrice
    return data

x = 1
while True:
    #rnd = random.random()
    rnd = random.randint(1,3)
    if rnd == 1:
        ticker = "AAAA"
    elif rnd == 2:
        ticker = "BBBB"
    else:
        ticker = "CCCC"
    data = json.dumps(getOrderData(x, ticker))
    kinesis.put_record("OrdersAndTradesStream", data, "partitionkey")
    print data
    tId = 1
    for y in range (0, random.randint(0,6)):
        tradeId = tId
        tradePrice = random.randint(0, 3000)
        data2 = json.dumps(getTradeData(x, tradeId, ticker, tradePrice));
        kinesis.put_record("OrdersAndTradesStream", data2, "partitionkey")
        print data2
        tId+=1

    x+=1
```

Next Step

Step 2: Create an Application (p. 57)

Step 2: Create an Application

In this section, you create an Amazon Kinesis Analytics application. You then update the application by adding input configuration that maps the streaming source you created in the preceding section to an in-application input stream.

1. Sign in to the AWS Management Console and open the Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create new application**. We assume application name is **`ProcessMultipleRecordTypes`**
3. On the application hub, connect to the source.
4. On the **Source** page,
    a. Select the stream you created in the preceding section.
    b. Choose the create IAM role option.
    c. Wait for console to show the inferred schema and samples records used to infer the schema for the in-application stream created.
    d. Choose **Save and continue**.

5. On the application hub, choose **Go to SQL editor**. To start the application, reply "yes" in the dialog box that appears.
6. In the SQL editor, write application code and verify results:

   a. Copy the following application code and paste it into the editor.

```
--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. Choose **Save and run SQL**. Choose the **Real-time analytics** tab to see all of the in-application streams that the application created and verify data.

Next Step

You can configure application output to persist results to an external destination, such as another Amazon Kinesis stream or a Firehose delivery stream.

Example: Adding Reference Data to an Amazon Kinesis Analytics Application

Topics

In this exercise, you add reference data to an existing Amazon Kinesis Analytics application. For information about reference data, see the following topics:

- Amazon Kinesis Analytics: How It Works (p. 3)
- Configuring Application Input (p. 5)

In this exercise you add reference data to the application you created in the getting started exercise. The reference data provides company name for each ticker symbol. For example,

```
Ticker, Company
AMZN,Amazon
ASD, SomeCompanyA
MMB, SomeCompanyB
WAS,  SomeCompanyC
```

First complete the Getting Started Exercise. Then you do the following to set up and add reference data to your application.

1. Prepare
   - Store preceding reference data as an object in your S3 bucket.
   - Create an IAM role, that Amazon Kinesis Analytics can assume to read the S3 object on your behalf.
2. Add the reference data source to your application. Amazon Kinesis Analytics reads the S3 object and create an in-application reference table that you can query in your application code.
3. Test. In your application code you will write a join query to join the in-application stream with the in-application reference table, to get company name for each ticker symbol.

> **Note**
> Amazon Kinesis Analytics console does not support managing reference data sources for your applications. In this exercise, you use the AWS CLI to add reference data source to your application. If you haven't already done so, set up the AWS CLI.

Step 1: Prepare

In this section, you store sample reference data as an object in your S3 bucket. You also create an IAM role that Amazon Kinesis Analytics can assume to read the object on your behalf.

Prepare: Store Reference Data as S3 Object

Store sample reference data as S3 object.

1. Open a text editor, type the following data, and save the file as `TickerReference.csv`.

```
Ticker, Company
AMZN,Amazon
ASD, SomeCompanyA
MMB, SomeCompanyB
WAS,  SomeCompanyC
```

2. Upload the `TickerReference.csv` file to your S3 bucket. For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.

Prepare: Create an IAM Role

Create an IAM role. Follow the procedure to create an IAM role that Amazon Kinesis Analytics can assume and read the S3 object.

1. Create an IAM role called **`KinesisAnalytics-ReadS3Object`**. In the IAM console, you specify the following when you create a role:

   - Choose **AWS Lambda** on the **Select Role Type**. After creating the role, you will change the trust policy to allow Amazon Kinesis Analytics to assume the role (not AWS Lambda).
   - Do not attach any policy on the **Attach Policy** page.

   For instructions, see Creating a Role for an AWS Service (AWS Management Console) in the *IAM User Guide*.

2. Update the IAM role policies.

   a. In the IAM console, select the role you created.

   b. On the **Trust Relationships** tab, update the trust policy to allow Amazon Kinesis Analytics permissions to assume the role. The trust policy is shown following:

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

   c. On the **Permissions** tab, attach an AWS managed policy called **AmazonS3ReadOnlyAccess**. This grants the role permissions to read an S3 object. The policy is shown following for your information:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

Step 2: Add Reference Data Source to the Application Configuration

In this section you add reference data source to your application configuration. You will need the following information:

- Your Amazon Kinesis Analytics application name and current application version ID
- S3 bucket name and object key name
- IAM role ARN

Now, you now use the AWS CLI to complete the step:

1. Run the `describe-application` to get the application description, as shown following:

```
$  aws kinesisanalytics describe-application  \
--region us-east-1 \
--application-name application-name
```

2. Note the current application version ID.

   Each time you make changes to your application, the current version is updated. So you need to make sure you have the current application version ID.

3. Use the following JSON to add the reference data source:

```
{
   "TableName":"CompanyName",
   "S3ReferenceDataSource":{
      "BucketARN":"arn:aws:s3:::bucket-name",
      "FileKey":"TickerReference.csv",
      "ReferenceRoleARN":"arn:aws:iam::aws-account-id:role/IAM-role-name"
   },
   "ReferenceSchema":{
      "RecordFormat":{
         "RecordFormatType":"CSV",
         "MappingParameters":{
            "CSVMappingParameters":{
               "RecordRowDelimiter":"\n",
               "RecordColumnDelimiter":","
```

```
            }
         }
      },
      "RecordEncoding":"UTF-8",
      "RecordColumns":[
         {
            "Name":"Ticker",
            "SqlType":"VARCHAR(64)"
         },
         {
            "Name":"Company",
            "SqlType":"VARCHAR(64)"
         }
      ]
   }
}
```

Run the `add-application-reference-data-source` command using the preceding reference data configuration information. You need to provide your bucket name, object key name, IAM role name, and AWS account ID.

```
$  aws kinesisanalytics add-application-reference-data-source  \
--endpoint https://kinesis-stream-analytics-aws-region.amazon.com \
--region us-east-1 \
--application-name DemoStreamBasedGettingStarted \
--debug \
--reference-data-source
 '{"TableName":"CompanyName","S3ReferenceDataSource":
{"BucketARN":"arn:aws:s3:::bucket-name","FileKey":"TickerReference.csv",
"ReferenceRoleARN":"arn:aws:iam::aws-account-id:role/IAM-
role-name"},"ReferenceSchema":{ "RecordFormat":
{"RecordFormatType":"CSV", "MappingParameters":{"CSVMappingParameters":
{"RecordRowDelimiter":"\n","RecordColumnDelimiter":","} }},"RecordEncoding":"string","Re
[{"Name":"Ticker","SqlType":"VARCHAR(64)"},
{ "Name":"Company","SqlType":"VARCHAR(64)"}]}}' \
--current-application-version-id 10
```

4. Verify that the reference data was added to the application by getting the application description using the `describe-application` operation.

Step 3: Test: Query the In-Application Reference Table

You can now query the in-application reference table, `CompanyName`. You can use the reference information to enrich your application by joining the ticker price data with the reference table, and then the result shows the company name.

1. Replace your application code by the following. The query joins the in-application input stream with the in-application reference table. The application code writes the results to another in-application stream, 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 c
   ON "SOURCE_SQL_STREAM_001".ticker_symbol = c.Ticker;
```

2. Verify that the application output appears in the **SQLResults** tab. Make sure some of the rows show company names (your sample reference data does not have all company names).

Examples: Basic Analytics

This section provides examples of Amazon Kinesis Analytics applications that perform basic analytics. The examples provide step-by-step instructions to set up an Amazon Kinesis Analytics application.

Topics

Example: Most Frequently Occurring Values (the TOP_K_ITEMS_TUMBLING Function)

In this exercise, you set up an Amazon Kinesis Analytics application to find the top ten most frequently traded stocks in a one-minute window.

For this exercise, you use the demo stream, which provides continuous flow of simulated stock trade records and finds the top ten most frequently traded stocks in a one-minute window. For more information about the demo stream, see Step 4: Console Feature Summary (p. 27).

Use the following application code:

```
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 *
      FROM TABLE(TOP_K_ITEMS_TUMBLING(
                        CURSOR(SELECT STREAM * FROM
 "SOURCE_SQL_STREAM_001"),
                        'column1', -- name of column in single quote.
                        10,          -- number of top items.
                        60           -- tumbling window size in seconds
                        )
                  );
```

The code uses the `TOP_K_ITEMS_TUMBLING` function to find the most frequently traded stock. Note that, for efficiency, the function approximates the most frequently occurring values. For more information about the function, see TOP_K_ITEMS_TUMBLING Function in the *Amazon Kinesis Analytics SQL Reference*.

In the console, this application code is available as a template (**Approximate Top-K items**), which you use to quickly create the application. You need to update this template code by replacing '*column1*' with `'TICKER_SYMBOL'` to estimate the most frequently occurring values, in a one-minute tumbling window.

You can use the following procedure to test this template using the demo stream.

To create an application

1. Complete the Getting Started exercise. For instructions, see Step 3: Getting Started Exercise (Create an Amazon Kinesis Analytics Application) (p. 19).
2. Replace the application code in the SQL editor with the **Approximate Top-K items** template as follows in the SQL editor:
   a. Delete the existing sample code.
   b. Choose **Add SQL from templates** and then select the **TopKItems** template.
   c. Update the template code by replacing the column name from `COLUMN1` to `'TICKER_SYMBOL'` (with single quotes around). Also, change the number of items from 10 to 3, so that you get the top three most frequently traded stocks in each one-minute window.
3. Save and run SQL. Review results in the **Real-time analytics** tab in the SQL editor.

   Because the window size is one minute, you need to wait to see the results. The `DESTINATION_SQL_STREAM` displays three columns (`ROWTIME`, `ITEM`, and `ITEM_COUNT`). The query emits results every one minute.

Example: Counting Distinct Values (the COUNT_DISTINCT_ITEMS_TUMBLING function)

In this exercise, you set up a Amazon Kinesis Analytics application to count distinct values in a one-minute tumbling window.

For the exercise, you use the demo stream, which provides continuous flow of simulated stock trade records and finds distinct stocks traded in a one-minute window. For information about the demo stream, see Step 4: Console Feature Summary (p. 27).

Use the following application code:

```
CREATE OR REPLACE STREAM DESTINATION_SQL_STREAM (
               NUMBER_OF_DISTINCT_ITEMS BIGINT);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM *
      FROM TABLE(COUNT_DISTINCT_ITEMS_TUMBLING(
                       CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"),
                       'column1', -- name of column in single quotes
                        60          -- tumbling window size in seconds
  )
);
```

The code uses the `COUNT_DISTINCT_ITEMS_TUMBLING` function to approximate the number of distinct values. For more information about the function, see COUNT_DISTINCT_ITEMS_TUMBLING Function in the *Amazon Kinesis Analytics SQL Reference*.

In the console, this application code is available as a template (**Approximate distinct count**), which you use to quickly create the application. You need to update this template code by replacing '*columnn1*' with `'TICKER_SYMBOL'` to estimate the number of distinct stocks traded, in a one-minute tumbling window.

You can use the following procedure to test this template using the demo stream.

To create an application

1. Complete the getting started exercise. For instructions, see Step 3: Getting Started Exercise (Create an Amazon Kinesis Analytics Application) (p. 19).
2. Now you replace the application code in the SQL editor by the **Approximate distinct count** template as follows. In SQL editor, do the following:
   a. Delete the existing sample code.
   b. Choose **Add SQL from templates** and then select the **Approximate distinct count** template.
   c. Update the template code by replacing the column name from *columnn1* to 'TICKER_SYMBOL' (with single quotes around).
3. Save and run SQL. Review results in the **Real-time analytics** tab in the SQL editor.

   Because the window size is one minute, you need to wait to see the results. The `DESTINATION_SQL_STREAM` shows two columns (`ROWTIME` and `NUMBER_OF_DISTINCT_ITEMS`). The query emits results every one minute.

Example: Simple Alerts

In this application, the query runs continuously on the in-application stream created over the demo stream. For more information, see Continuous Queries (p. 35). If any rows show stock price change is greater than 1%, those rows are inserted in another in-application stream. In the exercise, you can configure the application output persist the results to an external destination. You can then further investigate results. For example, you can use an AWS Lambda function to process records and send you alerts.

To create a simple alerts application

1. Create the Amazon Kinesis Analytics application as described in the Getting Started Exercise.
2. In the SQL editor, replace the application code with the following:

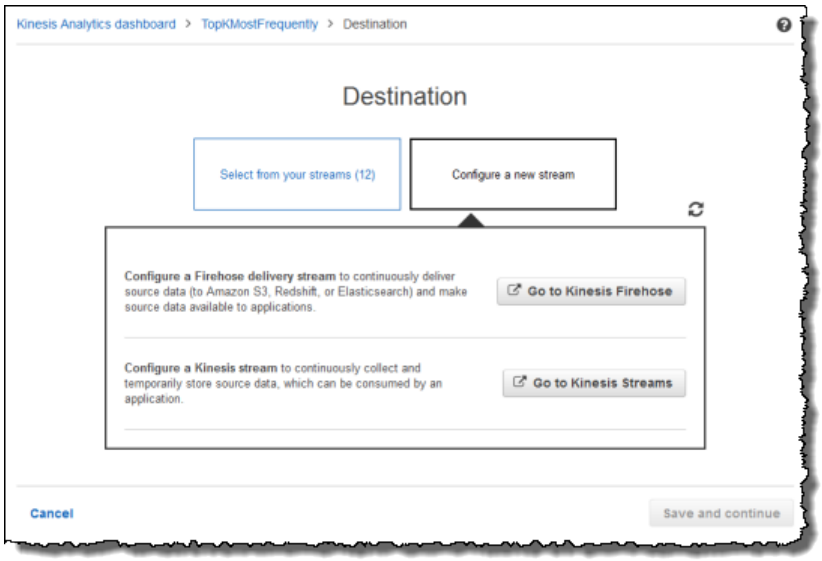
```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
           (ticker_symbol VARCHAR(4),
            sector        VARCHAR(12),
            change        DOUBLE,
            price         DOUBLE);

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;
```

The SELECT statement in the application code filters rows in the `SOURCE_SQL_STREAM_001` for stock price changes greater than 1%, and inserts those rows to another in-application stream `DESTINATION_SQL_STREAM` using a pump. For more information about the coding pattern that explains using pumps to insert rows in in-application streams, see Application Code (p. 10).

3. Click **Save and run SQL**.
4. Add a destination. You can either choose the **Destination** in the SQL Editor, or choose **Add a destination** on the application hub.

   a. In SQL editor, choose the **Destination** tab and then choose **Add a destination**.

      On the **Add a destination** page, choose **Configure a new stream**.



   b. Choose **Go to Kinesis Streams**.

   c. In the Amazon Kinesis Streams console, create a new Amazon Kinesis stream (for example, `gs-destination`) with 1 shard. Wait until the stream status is **ACTIVE**.

   d. Return to the Amazon Kinesis Analytics console. On the **Destination** page, choose the stream that you created.

      If the stream does not show, refresh the page.

      Now you have an external destination, where Amazon Kinesis Analytics persists any records your application writes to the in-application stream `DESTINATION_SQL_STREAM`.

   e. Choose **Save and continue**.

   Now you have an external destination, a Amazon Kinesis stream, where Amazon Kinesis Analytics persists your application output in the `DESTINATION_SQL_STREAM` in-application stream.

5. Configure AWS Lambda to monitor the Amazon Kinesis stream you created and invoke a Lambda function.

   For instructions, see Example: Integrating Amazon Kinesis Analytics with AWS Lambda (p. 73).

Examples: Advanced Analytics

This section provides additional examples of Amazon Kinesis Analytics applications. This includes using the `RANDOM_CUT_FOREST` function to assign anomaly scores to your stream data. You can then evaluate the anomaly scores to determine if the data is anomalous and perhaps take additional action. In addition, how section provides examples of using different types of times in analytics.

Topics

Example: Detecting Data Anomalies on a Stream (the RANDOM_CUT_FOREST Function)

Amazon Kinesis Analytics provides a function (`RANDOM_CUT_FOREST`) that can assign an anomaly score to each record based on values in the numeric columns. For more information, see RANDOM_CUT_FOREST Function in the *Amazon Kinesis Analytics SQL Reference*. In this exercise, you write application code to assign anomaly score to records on your application's streaming source. You do the following to set up the application:

1. **Set up a streaming source** – You set up a Amazon Kinesis stream and write sample `heartRate` data as shown following:

   ```
   {"heartRate": 60, "rateType":"NORMAL"}
   ...
   {"heartRate": 180, "rateType":"HIGH"}
   ```

   The walkthrough provides a Python script for you to populate the stream. The `heartRate` values are randomly generated, with 99% of the records having `heartRate` values between 60 and 100, and only 1% of `heartRate` values between 150 and 200. Thus, records with `heartRate` values between 150 and 200 are anomalies.
2. **Configure input** – Using the console, create an Amazon Kinesis Analytics application, and configure application input by mapping the streaming source to an in-application stream (`SOURCE_SQL_STREAM_001`). When the application starts, Amazon Kinesis Analytics continuously reads the streaming source and inserts records into the in-application stream.
3. **Specify application code** – Use the following application code:

   ```
   --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")));

   -- 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;
```

The code reads rows in the SOURCE_SQL_STREAM_001, assigns an anomaly score, and writes the resulting rows to another in-application stream (TEMP_STREAM). The application code then sorts the records in the TEMP_STREAM and saves the results to another in-application stream (DESTINATION_SQL_STREAM). Note that you use pumps to insert rows in in-application streams. For more information, see In-Application Streams and Pumps (p. 31).

4. **Configure output** – You configure the application output to persist data in the DESTINATION_SQL_STREAM to an external destination, which is another Amazon Kinesis stream. Reviewing the anomaly scores assigned to each record and determining what score indicates an anomaly (and you need to be alerted) is external to the application. You can use a Lambda function to process these anomaly scores and configure alerts.

The exercise uses the US East (N. Virginia) (us-east-1) AWS Region to create these streams and your application. If you use any other region, you need to update the code accordingly.

Next Step

Step 1: Prepare (p. 68)

Step 1: Prepare

Before you create an Amazon Kinesis Analytics application for this exercise, you create two Amazon Kinesis streams. You configure one of the streams as the streaming source for your application, and another stream as destination where Amazon Kinesis Analytics persists your application output.

Step 1.1: Create Two Amazon Kinesis Streams

In this section, you create two Amazon Kinesis streams (ExampleInputStream and ExampleOutputStream).

1. You can create these streams using the console or the AWS CLI.

   - Sign in to the AWS Management Console and open the Analytics console at https://console.aws.amazon.com/kinesisanalytics.
   - Choose **Kinesis Stream**, and then create a stream with one shard.
   - Use the following Amazon Kinesis create-stream CLI command to create the first stream (ExampleInputStream).

   ```
   $ aws kinesis create-stream \
   --stream-name ExampleInputStream \
   --shard-count 1 \
   --region us-east-1 \
   --profile adminuser
   ```

2. Run the same command, changing the stream name to ExampleOutputStream, to create the second stream that the application will use to write output.

Step 1.2: Write Sample Records to the Input Stream

In this step, you run Python code to continuously generate sample records and write to the ExampleInputStream stream.

```
{"heartRate": 60, "rateType":"NORMAL"}
```

```
...
{"heartRate": 180, "rateType":"HIGH"}
```

The code writes these records to the `ExampleInputStream` stream.

1. Install Python and `pip`.

   For information about installing Python, see the Python website.

   You can install dependencies using pip. For information about installing pip, see Installation on the pip website.

2. Run the following Python code. The `put-record` command in the code writes the JSON records to the stream.

```
import json
from boto import kinesis
import random

kinesis = kinesis.connect_to_region("us-east-1")
# generate normal heart rate with probability .99
def getNormalHeartRate():
    data = {}
    data['heartRate'] = random.randint(60, 100)
    data['rateType'] = "NORMAL"
    return data
# generate high heart rate with probability .01 (very few)
def getHighHeartRate():
    data = {}
    data['heartRate'] = random.randint(150, 200)
    data['rateType'] = "HIGH"
    return data

while True:
    rnd = random.random()
    if (rnd < 0.01):
        data = json.dumps(getHighHeartRate())
        print data
        kinesis.put_record("ExampleInputStream", data, "partitionkey")
    else:
        data = json.dumps(getNormalHeartRate())
        print data
        kinesis.put_record("ExampleInputStream", data, "partitionkey")
```

Next Step

Step 2: Create an Application (p. 69)

Step 2: Create an Application

In this section, you create an Amazon Kinesis Analytics application as follows:

- Configure the application input to use the Amazon Kinesis stream you created in the preceding section as the streaming source.
- Use the **Anomaly Detection** template in the console.

To create an application

1. Follow steps 1, 2, and 3 in Getting Started exercise (see Step 3.1: Create an Application (p. 20)) to create an application. Note the following:

   - In the source configuration, do the following:

     - Specify the streaming source you created in the preceding section.
     - After the console infers the schema, edit the schema and set the `heartRate` column type to INTEGER.

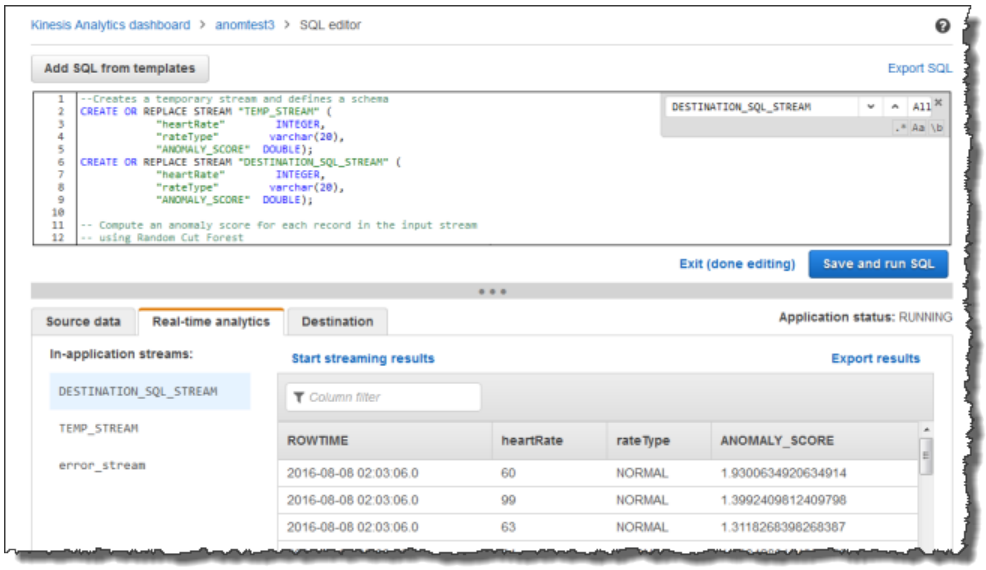
       Most of the heart rate values are normal and the discovery process will most likely assign TINYINT type to this column. But very small percentage of values that show high heart rate. If these high values don't fit in the TINYINT type, Amazon Kinesis Analytics sends these rows to error stream. Update the data type to INTEGER so that it can accommodate all of the generated heart rate data.

   - Use the **Anomaly Detection** template in the console. You then update the template code to provide appropriate column name.

2. Update the application code by providing column names. The resulting application code is shown following (you can paste this code into the SQL editor):

```
--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")));

-- 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. Run the SQL code and review results:



Next Step

Step 3: Configure Application Output (p. 71)

Step 3: Configure Application Output

At this time, you have application code reading heart rate data from a streaming source and assigning an anomaly score to each. You can now send the application result from the in-application stream to an external destination, another Amazon Kinesis stream (`OutputStreamTestingAnomalyScores`). You can then analyze the anomaly scores and determine which heart rate is anomalous. You can extend this application further to generate alerts. Follow these steps to configure application output:

1. In the SQL editor, choose either **Destination** or **Add a destination** in the application dashboard.
2. On the **Add a destination** page, choose **Select from your streams**, and then choose the `OutputStreamTestingAnomalyScores` stream you created in the preceding section.

   Now you have an external destination, where Amazon Kinesis Analytics persists any records your application writes to the in-application stream `DESTINATION_SQL_STREAM`.
3. You can optionally configure AWS Lambda to monitor the `OutputStreamTestingAnomalyScores` stream and send you alerts. For instructions, see Example: Integrating Amazon Kinesis Analytics with AWS Lambda (p. 73). If not, you can review the records that Amazon Kinesis Analytics writes to the external destination, the Amazon Kinesis stream `OutputStreamTestingAnomalyScores`, as described in the next step.

Next Step

Step 4: Verify Output (p. 71)

Step 4: Verify Output

In this step, you use the following AWS CLI commands to read records in the destination stream written by the application:

1. Run the `get-shard-iterator` command to get a pointer to data on the output stream.

```
aws kinesis get-shard-iterator \
--shard-id shardId-000000000000 \
--shard-iterator-type TRIM_HORIZON \
--stream-name OutputStreamTestingAnomalyScores \
--region us-east-1 \
--profile adminuser
```

   You get a response with a shard iterator value, as shown in the following example response:

```
  {
      "ShardIterator":
      "shard-iterator-value"   }
```

   Copy the shard iterator value.

2. Run the CLI `get-records` command.

```
aws kinesis get-records \
--shard-iterator shared-iterator-value \
--region us-east-1 \
--profile adminuser
```

   The command returns a page of records and another shard iterator that you can use in the subsequent `get-records` command to fetch the next set of records.

Example: Using Different Types of Times in Streaming Analytics

For information about different types of times and an example query, see Timestamps and the ROWTIME Column (p. 32). You can try the example query in that section against the demo stream you created in the Getting Started Exercise.

Examples: Post Processing In-Application Stream

In your Amazon Kinesis Analytics application, you can create in-application streams to store intermediate results of analytics. Post processing refers to persisting the results stored in-application streams to external destinations for further analysis.

In your application configuration, you can configure output to persist data in your in-application streams to external destinations, such as an Amazon Kinesis stream or an Amazon Kinesis Firehose delivery stream, for further analysis.

For example, if application output is persisted to an Amazon Kinesis stream, you can configure AWS Lambda to poll the stream and invoke a Lambda function to process records on the stream.

Topics

Example: Integrating Amazon Kinesis Analytics with AWS Lambda

Integrating Amazon Kinesis Analytics applications with AWS Lambda enable additional scenarios. If you persist your application output an Amazon Kinesis stream, you can have AWS Lambda poll the stream and invoke a Lambda function. Your Lambda function can then process records that arrive on the stream, for example write those records to a destination of your choice.

The example Amazon Kinesis Analytics application in the following sections persist output to an Amazon Kinesis stream:

- Example: Simple Alerts (p. 65)
- Example: Detecting Data Anomalies on a Stream (the RANDOM_CUT_FOREST Function) (p. 67)

You can further enhance these examples using AWS Lambda to publish alerts. For illustration, this section shows how to create a Lambda function and configure AWS Lambda so you get email notifications when records arrive at the Amazon Kinesis Analytics stream.

You configure AWS Lambda as follows:

- Configure Lambda to poll the Amazon Kinesis stream and invoke your Lambda function when new records are detected. The Lambda function receives these new records as the *event* parameter.

- Write a Lambda function to process the events. In this example, the Lambda function publishes a message to an Amazon Simple Notification Service (Amazon SNS) topic.

  For testing, you subscribe to the topic using email protocol. Amazon SNS then notifies you whenever the Lambda function publishes a message (an alert) to the Amazon SNS topic.

- Add event source mapping in AWS Lambda to associate the Lambda function with your Amazon Kinesis stream.

> **Note**
> The instructions in this exercise use the US East (N. Virginia) Region, (`us-east-1`).

About AWS Lambda

If you are new to AWS Lambda, we recommend that you read the overview topic What IS AWS Lambda? in the *AWS Lambda Developer Guide*. The Using AWS Lambda with Amazon Kinesis chapter also provides an AWS Lambda and Amazon Kinesis Analytics integration example that you might find useful. However, that example uses the AWS CLI. In this exercise you use the AWS Lambda console to quickly create a Lambda function and map it to the destination stream of your application.

Topics

- Step 1: Create an Amazon Kinesis Analytics Application (p. 74)
- Step 2: Create an Amazon SNS Topic (p. 74)
- Step 3: Create a Lambda Function (p. 74)
- Step 4: Verify Results (p. 75)

Step 1: Create an Amazon Kinesis Analytics Application

In this section you set up an Amazon Kinesis Analytics application as follows:

1. First set up the example application that assigns anomaly score to heart rate data on a stream. For instructions, see Example: Detecting Data Anomalies on a Stream (the RANDOM_CUT_FOREST Function) (p. 67).
2. You now update part of the application code that writes rows to the DESTINATION_SQL_STREAM stream. Now you want application to write only rows with higher anomaly score to the DESTINATION_SQL_STREAM.

```
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM * FROM "TEMP_STREAM"
      WHERE "ANOMALY_SCORE" > 3.0;
```

Here we choose, 3.0 anomaly score, you can tweak this value as needed. The idea is to have the application write high heart rate records to the output.

Step 2: Create an Amazon SNS Topic

Create an Amazon SNS topic and subscribe to it using the email as the protocol. Your Lambda function will post messages to the topic and you will get email notifications. For instructions, see Getting Started with Amazon Simple Notification Service in the *Amazon Simple Notification Service Developer Guide*.

Step 3: Create a Lambda Function

In this step, you do two things—create a Lambda function and then map your application destination stream as the event source for your Lambda function.

If you are new to AWS Lambda, we recommend that you first review AWS Lambda: How It Works in the *AWS Lambda Developer Guide*.

In the AWS Lambda console at https://console.aws.amazon.com/lambda/, choose **Create Function** and then follow these steps:

1. On the **Step 1: Select blueprint** page, select the **kinesis-process-record-python** blueprint. This blueprint closely resembles the scenario in this exercise.
2. On the **Step2: Configure event sources** page, specify the following values:

   - **Event source type** – Kinesis
   - **Kinesis stream** – Select the Amazon Kinesis stream from the that is the configured destination for your Amazon Kinesis Analytics application.
   - **Batch size** – 1

3. On the **Step 3: Configure function** page, specify following values:

   Name – ProcessAnomalies

   Runtime – Python 2.7.

   Replace the sample code by the following:

```
import base64
```

```
import json
import boto3

snsClient = boto3.client('sns')
print('Loading function')

def lambda_handler(event, context):
    for record in event['Records']:
        # Kinesis data is base64 encoded so decode here
        # payload = json.loads(base64.b64decode(record['kinesis']
['data']))
        payload = base64.b64decode(record['kinesis']['data'])
        print payload
        response = snsClient.publish(
        TopicArn='SNS-topic-ARN',
        Message='Anomaly detected ...  ' + payload,
        Subject='Anomaly detected',
        MessageStructure='string',
        MessageAttributes={
        'String': {
            'DataType': 'String',
            'StringValue': 'New records have been processed.'
            }
        }
    )
    return 'Successfully processed {}
 records.'.format(len(event['Records']))
```

Note
You need to update the code by providing the `TopicArn`.

Role – Choose **Kinesis execution role**. On the detail page that appears, choose **View Policy Document**, and then choose **edit**. Add permission for the **sns:Publish** action. This allows the Lambda function to publish the anomaly event to the specific Amazon SNS topic.

Timeout –60 seconds

Leave the default values for the other fields.

4. Choose **Create function** to create the Lambda function.
5. On the **Event sources** tab for the Lambda function, verify that the specific event source is **enabled**.

   You now have a Lambda function created and it is mapped to the destination stream of your application. AWS Lambda now begins polling the destination stream, and invokes your Lambda function when records appear on the stream.

Step 4: Verify Results

If all is well, you have the following occurring in your application flow:

- Sample script is writing data to your application's streaming source.
- Your application is processing records on the streaming source (assigning anomaly score to each record based on the hear rate), and writing records with anomaly scores to in-application output stream.
- Amazon Kinesis Analytics is writing records from the in-application output stream to the output destination (an Amazon Kinesis stream) configured for your application.

- AWS Lambda is polling your destination stream and invoking your Lambda function. Your Lambda function will process each record, and publish a message to your Amazon SNS topic.
- Amazon SNS is sending email notifications to you.

If you don't get Amazon SNS email notifications, you can check the logs in the CloudWatch log for your application. The logs provide information that can help you debug the problem. For example, your Lambda function might be posting messages to the Amazon SNS topic, but you have not subscribed to the topic (or you subscribed to the topic, but did not confirm the subscription). The log provides useful information that will help you fix the problem.

Examples: Other Amazon Kinesis Analytics Applications

This section provides examples that help you explore Amazon Kinesis Analytics concepts. This includes, examples in which you introduce runtime errors that cause your application send rows to in-application stream, explore console support for editing schemas that the console infers for in-application input stream, by sampling data on the streaming source.

Topics

Example: Explore the In-Application Error Stream

Amazon Kinesis Analytics provides an in-application error stream for each application you create. Any rows that your application cannot process are sent to this error stream. You might consider persisting the error stream data to an external destination so that you can investigate.

In this exercise, you introduce errors in input configuration by editing the schema inferred by the discovery process, and verify rows sent to the error stream.

You perform this exercise in the console.

Introduce Parse Error

In this exercise, you introduce a parse error.

1. Create an application. For instructions, see Step 3.1: Create an Application (p. 20).
2. On the newly created application hub, choose **Connect to a source**.
3. On the **Source** page, select the demo stream (`kinesis-anlaytics-demo-stream`).

   If you followed the Getting Started exercise, you have a demo stream in your account.
4. Amazon Kinesis Analytics takes sample from the demo stream to infer a schema for the in-application input stream it creates. The console show the inferred schema and sample data in the **Formatted stream sample** tab.
5. Now you edit the schema and modify column type to introduce the parse error. Choose **Edit schema**.
6. Change the `TICKER_SYMBOL` column type from `VARCHAR(4)` to `INTEGER`.

   Now that column type of the in-application schema that is created is invalid, Amazon Kinesis Analytics will not be able to bring in data in the in-application stream, instead Analytics will send the rows to error stream.

7. Choose **Save schema**.
8. Choose **Refresh schema samples**.

   Notice that there are no rows in the **Formatted stream** sample. However, the **Error stream** tab shows data with an error message. The **Error stream** tab shows data sent to the in-application error stream.

   Because you changed the column data type, Amazon Kinesis Analytics was not able to bring the data in the in-application input stream, and instead it sent the data to the error stream.

Divide by Zero Error

In this exercise you update application code to introduce a runtime error (division by zero), and notice that Amazon Kinesis Analytics sends the resulting rows to the in-application error stream, not to the in-application error stream where the results are supposed to be written.

1. Follow the Getting Started exercise to create an application. For instructions, see Step 3: Getting Started Exercise (Create an Amazon Kinesis Analytics Application) (p. 19).

   Verify the results on the **Real-time analytics** tab as follows:

   Sour
2. Update the `SELECT` statement in the application code to introduce divide by zero. For example:

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

3. Run the application. Because of the division by zero runtime error occurs, instead of writing results to the `DESTINATION_SQL_STREAM` Amazon Kinesis Analytics sends rows to the in-application error stream. On the **Real-time analytics** tab, choose the error-stream and then you can see the rows in the in-application error stream.

Monitoring Amazon Kinesis Analytics

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Kinesis Analytics and your Amazon Kinesis Analytics application. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. Before you start monitoring Amazon Kinesis Analytics, however, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

The next step is to establish a baseline for normal Amazon Kinesis Analytics performance in your environment, by measuring performance at various times and under different load conditions. As you monitor Amazon Kinesis Analytics, store historical monitoring data so that you can compare it with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

With Amazon Kinesis Analytics you monitor the application. The application processes data streams (input or output), both of which include *identifiers* which you can use to narrow your search on CloudWatch logs. For information about how Amazon Kinesis Analytics processes data streams, see Amazon Kinesis Analytics: How It Works (p. 3).

The most important metric is the `millisBehindLatest`, which indicates how far behind an application is reading from the streaming source. In a typical case, the millis behind should be at or near zero. It is common for brief spikes to appear, which appears as an increase in `millisBehindLatest`.

We recommend that you set up a CloudWatch alarm that triggers when the application is behind by more than an hour reading the streaming source. For some use cases that require very close to real-time processing, such as emitting processed data to a live application, you might choose to set the alarm at a lower value, such as five minutes.

For a list of metrics Amazon Kinesis Analytics supports, see Amazon Kinesis Analytics Metrics and Dimensions (p. 80).

Topics

Monitoring Tools

AWS provides various tools that you can use to monitor Amazon Kinesis Analytics. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated Monitoring Tools

You can use the following automated monitoring tools to watch Amazon Kinesis Analytics and report when something is wrong:

- **Amazon CloudWatch Alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see Monitoring with Amazon CloudWatch (p. 80).
- **Amazon CloudWatch Logs** – Monitor, store, and access your log files from AWS CloudTrail or other sources. For more information, see Monitoring Log Files in the *Amazon CloudWatch User Guide*.
- **Amazon CloudWatch Events** – Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see Using Events in the *Amazon CloudWatch User Guide*.
- **AWS CloudTrail Log Monitoring** – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see Working with CloudTrail Log Files in the *AWS CloudTrail User Guide*.

Manual Monitoring Tools

Another important part of monitoring Amazon Kinesis Analytics involves manually monitoring those items that the CloudWatch alarms don't cover. The Amazon Kinesis Analytics, CloudWatch, Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment.

- CloudWatch home page shows:
  - Current alarms and status
  - Graphs of alarms and resources
  - Service health status

  In addition, you can use CloudWatch to do the following:
  - Create customized dashboards to monitor the services you care about
  - Graph metric data to troubleshoot issues and discover trends
  - Search and browse all your AWS resource metrics

  - Create and edit alarms to be notified of problems
- AWS Trusted Advisor can help you monitor your AWS resources to improve performance, reliability, security, and cost effectiveness. Four Trusted Advisor checks are available to all users; more than 50 checks are available to users with a Business or Enterprise support plan. For more information, see AWS Trusted Advisor.

Monitoring with Amazon CloudWatch

You can monitor Amazon Kinesis Analytics applications using CloudWatch, which collects and processes raw data from Amazon Kinesis Analytics into readable, near real-time metrics. These statistics are retained for a period of two weeks, so that you can access historical information and gain a better perspective on how your web application or service is performing. By default, Amazon Kinesis Analytics metric data is automatically sent to CloudWatch. For more information, see What Are Amazon CloudWatch, Amazon CloudWatch Events, and Amazon CloudWatch Logs? in the *Amazon CloudWatch User Guide*.

Topics

Amazon Kinesis Analytics Metrics and Dimensions

When your Amazon Kinesis Analytics application processes data streams, Amazon Kinesis Analytics sends the following metrics and dimensions to CloudWatch. You can use the following procedures to view the metrics for Amazon Kinesis Analytics.

To view metrics using the CloudWatch console

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. In the **CloudWatch Metrics by Category** pane, under the metrics category for Amazon Kinesis Analytics, select a metrics category, and then in the upper pane, scroll down to view the full list of metrics.

To view metrics using the AWS CLI

- At a command prompt, use the following command:

```
aws cloudwatch list-metrics --namespace "AWS/KinesisAnalytics" --
region region
```

Amazon Kinesis Analytics metrics are collected at the following levels:

- Application-level
- Per input stream
- Per output stream

Dimensions and Metrics

The metrics and dimensions that Amazon Kinesis Analytics sends to Amazon CloudWatch are listed below.

Metrics

Amazon Kinesis Analytics sends the following metrics to CloudWatch:

Metric	Description
`Bytes`	The number of bytes read (per input stream) or written (per output stream). Levels: Per input stream and per output stream
`MillisBehindLatest`	Indicates how far behind from the current time an application is reading from the streaming source. Levels: Application-level
`Records`	The number of records read (per input stream) or written (per output stream). Levels: Per input stream and per output stream

Dimensions for Metrics

Amazon Kinesis Analytics provide metrics for the following dimensions:

Dimension	Description
`Flow`	Per input stream: Input Per output stream: Output
`Id`	Per input stream: Input Id Per output stream: Output Id

Creating CloudWatch Alarms to Monitor Amazon Kinesis Analytics

You can create a CloudWatch alarm that sends an Amazon SNS message when the alarm changes state. An alarm watches a single metric over a time period you specify, and performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or Auto Scaling policy. Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods.

Set alarms using the CloudWatch console

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Create Alarm**. This launches the **Create Alarm Wizard**.

3. Choose **Kinesis Analytics Metrics** and scroll through the Amazon Kinesis Analytics metrics to locate the metric you want to place an alarm on. To display just the Amazon Kinesis Analytics metrics in this dialog box, search on the file system id of your file system. Select the metric to create an alarm on and choose **Next**.
4. Fill in the **Name**, **Description**, **Whenever** values for the metric.
5. If you want CloudWatch to send you an email when the alarm state is reached, in the **Whenever this alarm:** field, choose **State is ALARM**. In the **Send notification to:** field, choose an existing SNS topic. If you select **Create topic**, you can set the name and email addresses for a new email subscription list. This list is saved and appears in the field for future alarms.

   Note
   If you use **Create topic** to create a new Amazon SNS topic, the email addresses must be verified before they receive notifications. Emails are only sent when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, they do not receive a notification.
6. At this point, the **Alarm Preview** area gives you a chance to preview the alarm you're about to create. Choose **Create Alarm**.

To set an alarm using the CloudWatch CLI

- Call `mon-put-metric-alarm`. For more information, see *Amazon CloudWatch CLI Reference*.

To set an alarm using the CloudWatch API

- Call `http://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_PutMetricAlarm.html`. For more information, see *Amazon CloudWatch API Reference*

Limits

When working with Amazon Kinesis Analytics, note the following limits:

- Size of a row in an in-application stream is limited to 50 KB.

- The service is available in specific AWS Regions. For more information, see Amazon Kinesis Analytics in the *AWS General Reference*.

- You can create up to five Amazon Kinesis Analytics applications per AWS Region in your account. You can create a case to request for additional applications via the service limit increase form. For more information, see AWS Support Center.

- Maximum amount of source parallelism is ten. That is, in your application input configuration, you can request the mapping of a streaming source to up to ten in-application streams.

- Number of Amazon Kinesis Analytics processing units (KPU) is limited to eight.

  With Amazon Kinesis Analytics, you pay only for what you use. You are charged an hourly-rate based on the average number of Kinesis Processing Units (or KPUs) used to run your stream processing application. A single KPU provides you with 1 vCPU and 4 GB of memory.

- Each application can have one streaming source and up to one reference data source.

- Amazon Kinesis Analytics supports four total different destinations for configuring application output; however the per-application limit is three destinations. We recommend that you use one of these destinations to persist data on the in-application error stream.

- The S3 object that stores reference data can be up to 1 GB in size.

- If you change the reference data stored in the S3 bucket after you upload reference data to an in-application table, you need use the UpdateApplication (p. 128) operation (using the API or AWS CLI) to refresh the data in the in-application table. Currently, the console does not support refreshing reference data in your application.

- Amazon Kinesis Analytics does not currently support data generated by the Amazon Kinesis Producer Library (KPL).

Best Practices

This section describes best practices when working with Amazon Kinesis Analytics applications.

Topics

Managing Applications

When managing Amazon Kinesis Analytics applications, follow these best practices:

- **Set up CloudWatch alarms** – Using the CloudWatch metrics that Amazon Kinesis Analytics provides, you can monitor the following:

  - Input bytes and input records (number of bytes and records entering the application)
  - Output bytes, output record
  - `MillisBehindLatest` (tracks how far behind the application is in reading from the streaming source)

  We recommend that you set up at least two CloudWatch alarms on the following metrics for your in-production applications:

  - Alarm on `MillisBehindLatest` – For most cases, we recommend that you set this alarm to trigger when your application is one hour behind the latest data, for an average of one minute. For applications with lower end-to-end processing needs, you can tune this to a lower tolerance. The alarm can help you ensure that your application is reading the latest data.

- Limit the number of production applications reading from the same Amazon Kinesis stream to two applications to avoid getting the `ReadProvisionedThroughputException` exception.

  Note
  In this case, the term *application* refers to any application that can read from the streaming source. Only an Amazon Kinesis Analytics application can read from a Firehose delivery stream. However, many applications can read from an Amazon Kinesis stream, such as an

Amazon Kinesis Analytics application or AWS Lambda. The recommended application limit refers to all applications that you configure to read from a streaming source.

Amazon Kinesis Analytics reads a streaming source approximately once per second per application. However, an application that falls behind might read data at a faster rate to catch up. To allow adequate throughput for applications to catch up, you limit the number of applications reading the same data source.

- Limit the number of production applications reading from the same Firehose delivery stream to one application.

  A Firehose delivery stream can write to destinations such as Amazon S3, Amazon Redshift, and it can also be a streaming source for your Amazon Kinesis Analytics application. Therefore, we recommend you do not configure more than one Amazon Kinesis Analytics application per Firehose delivery stream to make sure the delivery stream can also deliver to other destinations.

Defining Input Schema

When configuring application input in the console, you first specify a streaming source. The console then uses the discovery API (see DiscoverInputSchema (p. 120)) to infer a schema by sampling records on the streaming source. The schema, among other things, defines names and data types of the columns in the resulting in-application stream. The console displays the schema. We recommend you do the following with this inferred schema:

- Adequately test the inferred schema. The discovery process uses only a sample of records on the streaming source to infer a schema. If your streaming source has many record types, there is a possibility that the discovery API missed sampling one or more record types, which can result in a schema that does not accurately reflect data on the streaming source.

  When your application starts, these missed record types might result in parsing errors. Amazon Kinesis Analytics sends these records to the in-application error stream. To reduce these parsing errors, we recommend that you test the inferred schema interactively in the console, and monitor the in-application stream for missed records.

- The Amazon Kinesis Analytics API does not support specifying the `NOT NULL` constraint on columns in the input configuration. If you want `NOT NULL` constraints on columns in your in-application stream, you should create these in-application streams using your application code. You can then copy data from one in-application stream into another, and then the constraint will be enforced.

  Any attempt to insert rows with `NULL` values when a value is required results in an error, and Amazon Kinesis Analytics sends these errors to the in-application error stream.

- Relax data types inferred by the discovery process. The discovery process recommends columns and data types based on a random sampling of records on the streaming source. We recommend

that you review these carefully and consider relaxing these data types to cover all of the possible cases of records in your input. This ensures fewer parsing errors across the application while it is running. For example, if inferred schema has a `SMALLINT` as column type, perhaps consider changing it to `INTEGER`.

- Use SQL functions in your application code to handle any unstructured data or columns. You may have unstructured data or columns, such as log data, in your input. For examples, see Example: Manipulating Strings and Date Times (p. 44). One approach to handling this type of data is to define the schema with only one column of type `VARCHAR(N)`, where `N` is the largest possible row that you would expect to see in your stream. In your application code you can then read the incoming records, use the `String` and `Date Time` functions to parse and schematize the raw data.

- Make sure that you handle streaming source data that contains nesting more than two levels deep completely. When source data is JSON, you can have nesting. The discovery API will infer a schema that flattens one level of nesting. For two levels of nesting, the discovery API will also attempt to flatten these. Beyond two levels of nesting, there is limited support for flattening. In order to handle nesting completely, you have to manually modify the inferred schema to suite your needs. Use either of the following strategies to do this:

  - Use the JSON row path to selectively pull out only the required key value pairs for your application. A JSON row path provides a pointer to the specific key value pair you would like to bring in your application. This can be done for any level of nesting.

  - Use the JSON row path to selectively pull out complex JSON objects and then use string manipulation functions in your application code to pull the specific data that you need.

Connecting to Outputs

We recommend that every application have at least two outputs. use the first destination to insert the results of your SQL queries. Use the second destination to insert the entire error stream and send it to an S3 bucket through a Amazon Kinesis Firehose delivery stream.

Authoring Application Code

We recommend the following:

- In your SQL statement, we recommend that you do not specify time-based window that is longer than one hour for the following reasons:

  - If an application needs to be restarted, either because you updated the application or for Amazon Kinesis Analytics internal reasons, all data included in the window must be read again from the streaming data source. This will take time before Amazon Kinesis Analytics can emit output for that window.
  - Amazon Kinesis Analytics must maintain everything related to the application's state, including relevant data, for the duration. This will consume significant Amazon Kinesis Analytics processing units.

- During development, keep window size small in your SQL statements so that you can see the results faster. When you deploy the application to your production environment, you can set the window size as appropriate.
- Instead of a single complex SQL statement, you might consider breaking it into multiple statements, in each step saving results in intermediate in-application streams. This might help you debug faster.
- When using tumbling windows, we recommend that you use two windows, one for processing time and one for your logical time (ingest time or event time). For more information, see Timestamps and the ROWTIME Column (p. 32).

Authentication and Access Control for Amazon Kinesis Analytics

Access to Amazon Kinesis Analytics requires credentials. Those credentials must have permissions to access AWS resources, such as an Amazon Kinesis Analytics application or an Amazon Elastic Compute Cloud (Amazon EC2) instance. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon Kinesis Analytics to help secure access to your resources.

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your *root credentials* and they provide complete access to all of your AWS resources.

  Important
  For security reasons, we recommend that you use the root credentials only to create an *administrator user*, which is an *IAM user* with full permissions to your AWS account. Then, you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the *IAM User Guide*.
- **IAM user** – An IAM user is simply an identity within your AWS account that has specific custom permissions (for example, permissions to create an application in Amazon Kinesis Analytics). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the

several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use the AWS tools, you must sign the request yourself. Amazon Kinesis Analytics supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

- **IAM role** – An IAM role is another IAM identity you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use preexisting user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

  - **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the *IAM User Guide*.

  - **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data stored in the bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

  - **Applications running on Amazon EC2** – Instead of storing access keys within the EC2 instance for use by applications running on the instance and making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An instance profile contains the role and enables programs running on the EC2 instance to get temporary credentials. For more information, see Using Roles for Applications on Amazon EC2 in the *IAM User Guide*.

Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon Kinesis Analytics resources. For example, you must have permissions to create an Amazon Kinesis Analytics application.

The following sections describe how to manage permissions for Amazon Kinesis Analytics. We recommend that you read the overview first.

Overview of Managing Access Permissions to Your Amazon Kinesis Analytics Resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and some services (such as AWS Lambda) also support attaching permissions policies to resources.

> **Note**
> An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

Amazon Kinesis Analytics Resources and Operations

In Amazon Kinesis Analytics, the primary resource is *an application*. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to.

These resources have unique Amazon Resource Names (ARNs) associated with them, as shown in the following table.

Resource Type	ARN Format
Application	`arn:aws:kinesisanalytics:`*`region`*`:`*`account-id`*`:application/`*`application-name`*

Amazon Kinesis Analytics provides a set of operations to work with Amazon Kinesis Analytics resources. For a list of available operations, see Amazon Kinesis Analytics Actions (p. 101).

Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the principal entity (that is, the root account, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create an application, your AWS account is the owner of the resource (in Amazon Kinesis Analytics, the resource is an application).
- If you create an IAM user in your AWS account and grant permissions to create an application to that user, the user can create an application. However, your AWS account, to which the user belongs, owns the application resource.

- If you create an IAM role in your AWS account with permissions to create an application, anyone who can assume the role can create an application. Your AWS account, to which the user belongs, owns the application resource.

Managing Access to Resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

> **Note**
> This section discusses using IAM in the context of Amazon Kinesis Analytics. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What Is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based* policies (IAM polices) and policies attached to a resource are referred to as *resource-based* policies. Amazon Kinesis Analytics supports only identity-based policies (IAM policies).

Topics

Identity-Based Policies (IAM Policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to create an Amazon Kinesis Analytics resource, such as an application, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in account A can create a role to grant cross-account permissions to another AWS account (for example, account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in account A.
  2. Account A administrator attaches a trust policy to the role identifying account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in account B. Doing this allows users in account B to create or access resources in account A. The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

  For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

The following is an example policy that grants permission for the `kinesisanalytics:CreateApplication` action, which is required to create an Amazon Kinesis Analytics application.

Note that:

> **Note**
> This is an introductory example policy. When you attach the policy to the user, the user will be able to create an application using the AWS CLI or AWS SDK. But the user will need more

permissions to configure input and output. In addition, the user will need more permissions when using the console. The later sections provide more information.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1473028104000",
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:CreateApplication"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

For more information about using identity-based policies with Amazon Kinesis Analytics, see Using Identity-Based Policies (IAM Policies) for Amazon Kinesis Analytics (p. 94). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-Based Policies

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Amazon Kinesis Analytics doesn't support resource-based policies.

Specifying Policy Elements: Actions, Effects, and Principals

For each Amazon Kinesis Analytics resource, the service defines a set of API operations. To grant permissions for these API operations, Amazon Kinesis Analytics defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation. For more information about resources and API operations, see Amazon Kinesis Analytics Resources and Operations (p. 91) and Amazon Kinesis Analytics Actions (p. 101).

The following are the most basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. For more information, see Amazon Kinesis Analytics Resources and Operations (p. 91).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, you can use `create` to allow users to create an application.
- **Effect** – You specify the effect, either allow or deny, when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only). Amazon Kinesis Analytics doesn't support resource-based policies.

To learn more about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a list showing all of the Amazon Kinesis Analytics API operations and the resources that they apply to, see Amazon Kinesis Analytics API Permissions: Actions, Permissions, and Resources Reference (p. 99).

Specifying Conditions in a Policy

When you grant permissions, you can use the access policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition in the *IAM User Guide*.

To express conditions, you use predefined condition keys. There are no condition keys specific to Amazon Kinesis Analytics. However, there are AWS-wide condition keys that you can use as appropriate. For a complete list of AWS-wide keys, see Available Keys for Conditions in the *IAM User Guide*.

Using Identity-Based Policies (IAM Policies) for Amazon Kinesis Analytics

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on Amazon Kinesis Analytics resources.

> **Important**
> We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your Amazon Kinesis Analytics resources. For more information, see Overview of Managing Access Permissions to Your Amazon Kinesis Analytics Resources (p. 91).

Topics

- Permissions Required to Use the Amazon Kinesis Analytics Console (p. 95)
- AWS Managed (Predefined) Policies for Amazon Kinesis Analytics (p. 95)
- Customer Managed Policy Examples (p. 96)

The following shows an example of a permissions policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1473028104000",
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:CreateApplication"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
```

```
}
```

The policy has one statement:

- The first statement grants permissions for one Amazon Kinesis Analytics action (`kinesisanalytics:CreateApplication`) on a resource using the Amazon Resource Name (ARN) for the application. The ARN in this case specifies a wildcard character (*) to indicate the permission is granted for any resource.

For a table showing all of the Amazon Kinesis Analytics API operations and the resources that they apply to, see Amazon Kinesis Analytics API Permissions: Actions, Permissions, and Resources Reference (p. 99).

Permissions Required to Use the Amazon Kinesis Analytics Console

For a user to work with Amazon Kinesis Analytics console, you need to grant the requisite permissions. For example, if you want to grant a user permission to create an application, you need to grant permissions that will show the user the streaming sources in the account so that the user can configure input and output in the console.

We recommend the following:

- Use the AWS-managed policies to grant user permissions. For available policies, see AWS Managed (Predefined) Policies for Amazon Kinesis Analytics (p. 95).
- Create custom policies. In this case, we recommend that you review the example provided in this section. For more information, see Customer Managed Policy Examples (p. 96).

AWS Managed (Predefined) Policies for Amazon Kinesis Analytics

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see AWS Managed Policies in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon Kinesis Analytics:

- **AmazonKinesisAnalyticsReadOnly** – Grants permissions for Amazon Kinesis Analytics actions that enable user to list Amazon Kinesis Analytics applications and review input/output configuration. It also grants permission that allow user to view a list of Amazon Kinesis streams and Firehose delivery streams. The application is running, the user can view source data and real-time analytics results in the console.

- **AmazonKinesisAnalyticsFullAccess** – Grants permissions for all Amazon Kinesis Analytics actions and all other permissions that allows a user to create and manage Amazon Kinesis Analytics applications. However, note the following:

- These permissions are not sufficient if the user wants to create a new IAM role in the console (these permissions allow the user to select an existing role). If you want the user to be able to create an IAM role in the console, add the `IAMFullAccess` AWS managed policy.

- A user must have permission for the `iam:PassRole` action to specify an IAM role when configuring Amazon Kinesis Analytics application. This AWS managed policy grants permission for the `iam:PassRole` action to the user only on the IAM roles that start with the prefix `service-role/kinesis-analytics`.

  If the user wants to configure the Amazon Kinesis Analytics application with a role that does not have this prefix, you first need to explicitly grant the user permission for the `iam:PassRole` action on the specific role.

  Note
  You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

You can also create your own custom IAM policies to allow permissions for Amazon Kinesis Analytics actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

Customer Managed Policy Examples

The examples in this section provide a group of sample policies that you can attach to a user. If you are new to creating policies, we recommend that you first create an IAM user in your account and attach the policies to the user in sequence, as outlined in the steps in this section. You can then use the console to verify the effects of each policy as you attach the policy to the user.

Initially, the user doesn't have permissions and the user won't be able to do anything in the console. As you attach policies to the user, you can verify that the user can perform various actions in the console.

We recommend that you use two browser windows: one to create the user and grant permissions, and the other to sign in to the AWS Management Console using the user's credentials and verify permissions as you grant them to the user.

For examples that show how to create an IAM role that you can use as an execution role for your Amazon Kinesis Analytics application, see Creating IAM Roles in the *IAM User Guide*.

Example Steps

Step 1: Create an IAM User

First, you need to create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user that you created. You can then access AWS using a special URL and that IAM user's credentials.

For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.

Step 2: Allow the User Permissions for Actions that Are Not Specific to Amazon Kinesis Analytics

Let us first grant a user permission for all actions that aren't specific to Amazon Kinesis Analytics that the user will need when working with Amazon Kinesis Analytics applications. These include permissions working with streams (Amazon Kinesis Streams actions, Amazon Kinesis Firehose actions), and permissions for CloudWatch actions. Attach the following policy to the user.

You need to update the policy by providing an IAM role name for which you want to grant the `iam:PassRole` permission, or specify a wildcard character (*) indicating all IAM roles. This is not a secure practice; however you might not have a specific IAM role created during this testing.

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

Step 3: Allow the User to View a List of Applications and View Details

The following policy grants a user the following permissions:

- Permission for the `kinesisanalytics:ListApplications` action so the user can view a list of applications. Note that this is a service-level API call, and therefore you specify "*" as the Resource value.
- Permission for the `kinesisanalytics:DescribeApplication` action so that you can get information about any of the applications.

Add this policy to the user.

```
{
    "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/*"
        }
    ]
}
```

Verify these permissions by signing into the Amazon Kinesis Analytics console using the IAM user credentials.

Step 4: Allow the User to Start a Specific Application

If you want the user to be able to start one of the existing Amazon Kinesis Analytics applications, you attach the following policy to the user, which provides the permission for `kinesisanalytics:StartApplication` action. You will need to update the policy by providing your account id, region and application name.

```
{
    "Version": "2012-10-17",
    "Statement": [
                {
```

```
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:StartApplication"
            ],
            "Resource": "arn:aws:kinesisanalytics:aws-region:aws-account-
id:application/application-name"
        }
    ]
}
```

Step 5: Allow the User to Create an Amazon Kinesis Analytics Application

Now suppose you want the user to create an Amazon Kinesis Analytics application. You can then attach the following policy to the user. You will need to update the policy and provide a region, your account ID and either a specific application name that you want the user to create or a "*" so the user can specify any application name (and thus the user can create multiple applications).

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

Amazon Kinesis Analytics API Permissions: Actions, Permissions, and Resources Reference

When you are setting up Access Control (p. 90) and writing a permissions policy that you can attach to an IAM identity (identity-based policies), you can use the following list as a reference. The list includes each Amazon Kinesis Analytics API operation, the corresponding actions for which you can grant permissions to perform the action, and the AWS resource for which you can grant the

permissions. You specify the actions in the policy's `Action` field, and you specify the resource value in the policy's `Resource` field.

You can use AWS-wide condition keys in your Amazon Kinesis Analytics policies to express conditions. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*.

> **Note**
> To specify an action, use the `kinesisanalytics` prefix followed by the API operation name (for example, `kinesisanalytics:AddApplicationInput`).

Amazon Kinesis Analytics API and Required Permissions for Actions

API Operation:
Required Permissions (API Action):

Resources:

Amazon Kinesis Analytics API and Required Permissions for Actions

Amazon RDS API and Required Permissions for Actions

API Operation: AddApplicationInput (p. 102)
Action: `kinesisanalytics:AddApplicationInput`

Resources:

```
arn:aws:kinesisanalytics: region:accountId:application/application-name
```

API Reference

You can use the AWS CLI to explore the Amazon Kinesis Analytics API. This guide provides Getting Started (p. 17) exercises that use the AWS CLI.

Topics

Actions

The following actions are supported:

AddApplicationInput

Adds a streaming source to your Amazon Kinesis application. For conceptual information, see Configuring Application Input.

You can add a streaming source either when you create an application or you can use this operation to add a streaming source after you create an application. For more information, see CreateApplication (p. 108).

Any configuration update, including adding a streaming source using this operation, results in a new version of the application. You can use the DescribeApplication (p. 117) operation to find the current application version.

This operation requires permissions to perform the `kinesisanalytics:AddApplicationInput` action.

Request Syntax

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "Input": {
      "InputParallelism": {
         "Count": number
      },
      "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"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 102)
Name of your existing Amazon Kinesis Analytics application to which you want to add the streaming source.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

CurrentApplicationVersionId (p. 102)
Current version of your Amazon Kinesis Analytics application. You can use the DescribeApplication (p. 117) operation to find the current application version.

Type: Long

Valid Range: Minimum value of 1. Maximum value of 999999999.

Required: Yes

Input (p. 102)
When you configure the application input, you specify the streaming source, the in-application stream name that is created, and the mapping between the two. For more information, see Configuring Application Input.

Type: Input (p. 138) object

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

ConcurrentModificationException
Exception thrown as a result of concurrent modification to an application. For example, two individuals attempting to edit the same application at the same time.

HTTP Status Code: 400

InvalidArgumentException
Specified input parameter value is invalid.

HTTP Status Code: 400

ResourceInUseException
Application is not available for this operation.

HTTP Status Code: 400

ResourceNotFoundException
Specified application can't be found.

HTTP Status Code: 400

AddApplicationOutput

Adds an external destination to your Amazon Kinesis Analytics application.

If you want Amazon Kinesis Analytics to deliver data from an in-application stream within your application to an external destination (such as an Amazon Kinesis stream or a Firehose delivery stream), you add the relevant configuration to your application using this operation. You can configure one or more outputs for your application. Each output configuration maps an in-application stream and an external destination.

You can use one of the output configurations to deliver data from your in-application error stream to an external destination so that you can analyze the errors. For conceptual information, see Understanding Application Output (Destination).

Note that any configuration update, including adding a streaming source using this operation, results in a new version of the application. You can use the DescribeApplication (p. 117) operation to find the current application version.

For the limits on the number of application inputs and outputs you can configure, see Limits.

This operation requires permissions to perform the `kinesisanalytics:AddApplicationOutput` action.

Request Syntax

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "Output": {
      "DestinationSchema": {
         "RecordFormatType": "string"
      },
      "KinesisFirehoseOutput": {
         "ResourceARN": "string",
         "RoleARN": "string"
      },
      "KinesisStreamsOutput": {
         "ResourceARN": "string",
         "RoleARN": "string"
      },
      "Name": "string"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 104)

Name of the application to which you want to add the output configuration.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

CurrentApplicationVersionId (p. 104)

Version of the application to which you want add the output configuration. You can use the DescribeApplication (p. 117) operation to get the current application version. If the version specified is not the current version, the `ConcurrentModificationException` is returned.

Type: Long

Valid Range: Minimum value of 1. Maximum value of 999999999.

Required: Yes

Output (p. 104)

An array of objects, each describing one output configuration. In the output configuration, you specify the name of an in-application stream, a destination (that is, an Amazon Kinesis stream or an Amazon Kinesis Firehose delivery stream), and record the formation to use when writing to the destination.

Type: Output (p. 161) object

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

ConcurrentModificationException

Exception thrown as a result of concurrent modification to an application. For example, two individuals attempting to edit the same application at the same time.

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

HTTP Status Code: 400

AddApplicationReferenceDataSource

Adds a reference data source to an existing application.

Amazon Kinesis Analytics reads reference data (that is, an Amazon S3 object) and creates an in-application table within your application. In the request, you provide the source (S3 bucket name and object key name), name of the in-application table to create, and the necessary mapping information that describes how data in Amazon S3 object maps to columns in the resulting in-application table.

For conceptual information, see Configuring Application Input. For the limits on data sources you can add to your application, see Limits.

This operation requires permissions to perform the `kinesisanalytics:AddApplicationOutput` action.

Request Syntax

```
{
   "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"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 106)

Name of an existing application.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

CurrentApplicationVersionId (p. 106)

Version of the application for which you are adding the reference data source. You can use the DescribeApplication (p. 117) operation to get the current application version. If the version specified is not the current version, the `ConcurrentModificationException` is returned.

Type: Long

Valid Range: Minimum value of 1. Maximum value of 999999999.

Required: Yes

ReferenceDataSource (p. 106)

The reference data source can be an object in your Amazon S3 bucket. Amazon Kinesis Analytics reads the object and copies the data into the in-application table that is created. You provide an S3 bucket, object key name, and the resulting in-application table that is created. You must also provide an IAM role with the necessary permissions that Amazon Kinesis Analytics can assume to read the object from your S3 bucket on your behalf.

Type: ReferenceDataSource (p. 166) object

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

ConcurrentModificationException

Exception thrown as a result of concurrent modification to an application. For example, two individuals attempting to edit the same application at the same time.

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

HTTP Status Code: 400

CreateApplication

Creates an Amazon Kinesis Analytics application. You can configure each application with one streaming source as input, application code to process the input, and up to five streaming destinations where you want Amazon Kinesis Analytics to write the output data from your application. For an overview, see How it Works.

In the input configuration, you map the streaming source to an in-application stream, which you can think of as a constantly updating table. In the mapping, you must provide a schema for the in-application stream and map each data column in the in-application stream to a data element in the streaming source.

Your application code is one or more SQL statements that read input data, transform it, and generate output. Your application code can create one or more SQL artifacts like SQL streams or pumps.

In the output configuration, you can configure the application to write data from in-application streams created in your applications to up to five streaming destinations.

To read data from your source stream or write data to destination streams, Amazon Kinesis Analytics needs your permissions. You grant these permissions by creating IAM roles. This operation requires permissions to perform the `kinesisanalytics:CreateApplication` action.

For introductory exercises to create an Amazon Kinesis Analytics application, see Getting Started.

Request Syntax

```
{
   "ApplicationCode": "string",
   "ApplicationDescription": "string",
   "ApplicationName": "string",
   "Inputs": [
      {
         "InputParallelism": {
            "Count": number
         },
         "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"
         },
         "Name": "string"
      }
   ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationCode (p. 108)

One or more SQL statements that read input data, transform it, and generate output. For example, you can write a SQL statement that reads data from one in-application stream, generates a running average of the number of advertisement clicks by vendor, and insert resulting rows in another in-application stream using pumps. For more inforamtion about the typical pattern, see Application Code.

You can provide such series of SQL statements, where output of one statement can be used as the input for the next statement. You store intermediate results by creating in-application streams and pumps.

Note that the application code must create the streams with names specified in the `Outputs`. For example, if your `Outputs` defines output streams named `ExampleOutputStream1` and `ExampleOutputStream2`, then your application code must create these streams.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 51200.

Required: No

ApplicationDescription (p. 108)

Summary description of the application.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 1024.

Required: No

ApplicationName (p. 108)

Name of your Amazon Kinesis Analytics application (for example, `sample-app`).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

Inputs (p. 108)

Use this parameter to configure the application input.

You can configure your application to receive input from a single streaming source. In this configuration, you map this streaming source to an in-application stream that is created. Your application code can then query the in-application stream like a table (you can think of it as a constantly updating table).

For the streaming source, you provide its Amazon Resource Name (ARN) and format of data on the stream (for example, JSON, CSV, etc). You also must provide an IAM role that Amazon Kinesis Analytics can assume to read this stream on your behalf.

To create the in-application stream, you need to specify a schema to transform your data into a schematized version used in SQL. In the schema, you provide the necessary mapping of the data elements in the streaming source to record columns in the in-app stream.

Type: array of Input (p. 138) objects

Required: No

Outputs (p. 108)

You can configure application output to write data from any of the in-application streams to up to five destinations.

These destinations can be Amazon Kinesis streams, Amazon Kinesis Firehose delivery streams, or both.

In the configuration, you specify the in-application stream name, the destination stream Amazon Resource Name (ARN), and the format to use when writing data. You must also provide an IAM role that Amazon Kinesis Analytics can assume to write to the destination stream on your behalf.

In the output configuration, you also provide the output stream Amazon Resource Name (ARN) and the format of data in the stream (for example, JSON, CSV). You also must provide an IAM role that Amazon Kinesis Analytics can assume to write to this stream on your behalf.

Type: array of Output (p. 161) objects

Required: No

Response Syntax

```
{
   "ApplicationSummary": {
      "ApplicationARN": "string",
      "ApplicationName": "string",
      "ApplicationStatus": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ApplicationSummary (p. 110)

In response to your `CreateApplication` request, Amazon Kinesis Analytics returns a response with a summary of the application it created, including the application Amazon Resource Name (ARN), name, and status.

Type: ApplicationSummary (p. 134) object

Errors

CodeValidationException

User-provided application code (query) is invalid. This can be a simple syntax error.

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

HTTP Status Code: 400

LimitExceededException

Exceeded the number of applications allowed.

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

HTTP Status Code: 400

DeleteApplication

Deletes the specified application. Amazon Kinesis Analytics halts application execution and deletes the application, including any application artifacts (such as in-application streams, reference table, and application code).

This operation requires permissions to perform the `kinesisanalytics:DeleteApplication` action.

Request Syntax

```
{
   "ApplicationName": "string",
   "CreateTimestamp": number
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 112)
Name of the Amazon Kinesis Analytics application to delete.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

CreateTimestamp (p. 112)
You can use the `DescribeApplication` operation to get this value.

Type: Timestamp

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

ConcurrentModificationException
Exception thrown as a result of concurrent modification to an application. For example, two individuals attempting to edit the same application at the same time.

HTTP Status Code: 400

ResourceInUseException
Application is not available for this operation.

HTTP Status Code: 400

ResourceNotFoundException
Specified application can't be found.

HTTP Status Code: 400

DeleteApplicationOutput

Deletes output destination configuration from your application configuration. Amazon Kinesis Analytics will no longer write data from the corresponding in-application stream to the external output destination.

This operation requires permissions to perform the `kinesisanalytics:DeleteApplicationOutput` action.

Request Syntax

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "OutputId": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 113)
Amazon Kinesis Analytics application name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

CurrentApplicationVersionId (p. 113)
Amazon Kinesis Analytics application version. You can use the DescribeApplication (p. 117) operation to get the current application version. If the version specified is not the current version, the `ConcurrentModificationException` is returned.

Type: Long

Valid Range: Minimum value of 1. Maximum value of 999999999.

Required: Yes

OutputId (p. 113)
The ID of the configuration to delete. Each output configuration that is added to the application, either when the application is created or later using the AddApplicationOutput (p. 104) operation, has a unique ID. You need to provide the ID to uniquely identify the output configuration that you want to delete from the application configuration. You can use the DescribeApplication (p. 117) operation to get the specific `OutputId`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

ConcurrentModificationException
Exception thrown as a result of concurrent modification to an application. For example, two individuals attempting to edit the same application at the same time.

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

HTTP Status Code: 400

DeleteApplicationReferenceDataSource

Deletes a reference data source configuration from the specified application configuration.

If the application is running, Amazon Kinesis Analytics immediately removes the in-application table that you created using the AddApplicationReferenceDataSource (p. 106) operation.

This operation requires permissions to perform the `kinesisanalytics.DeleteApplicationReferenceDataSource` action.

Request Syntax

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "ReferenceId": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 115)
Name of an existing application.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

CurrentApplicationVersionId (p. 115)
Version of the application. You can use the DescribeApplication (p. 117) operation to get the current application version. If the version specified is not the current version, the `ConcurrentModificationException` is returned.

Type: Long

Valid Range: Minimum value of 1. Maximum value of 999999999.

Required: Yes

ReferenceId (p. 115)
ID of the reference data source. When you add a reference data source to your application using the AddApplicationReferenceDataSource (p. 106), Amazon Kinesis Analytics assigns an ID. You can use the DescribeApplication (p. 117) operation to get the reference ID.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

ConcurrentModificationException
Exception thrown as a result of concurrent modification to an application. For example, two individuals attempting to edit the same application at the same time.

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

HTTP Status Code: 400

DescribeApplication

Returns information about a specific Amazon Kinesis Analytics application.

If you want to retrieve a list of all applications in your account, use the ListApplications (p. 123) operation.

This operation requires permissions to perform the `kinesisanalytics:DescribeApplication` action. You can use `DescribeApplication` to get the current application versionId, which you need to call other operations such as `Update`.

Request Syntax

```
{
   "ApplicationName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 117)

Name of the application.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

Response Syntax

```
{
   "ApplicationDetail": {
      "ApplicationARN": "string",
      "ApplicationCode": "string",
      "ApplicationDescription": "string",
      "ApplicationName": "string",
      "ApplicationStatus": "string",
      "ApplicationVersionId": number,
      "CreateTimestamp": number,
      "InputDescriptions": [
         {
            "InAppStreamNames": [ "string" ],
            "InputId": "string",
            "InputParallelism": {
               "Count": number
            },
            "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"
          },
          "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"
         }
      ]
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ApplicationDetail (p. 117)

Provides a description of the application, such as the application Amazon Resource Name (ARN), status, latest version, and input and output configuration details.

Type: ApplicationDetail (p. 132) object

Errors

ResourceNotFoundException

Specified application can't be found.

HTTP Status Code: 400

DiscoverInputSchema

Infers a schema by evaluating sample records on the specified streaming source (Amazon Kinesis stream or Amazon Kinesis Firehose delivery stream). In the response, the operation returns the inferred schema and also the sample records that the operation used to infer the schema.

You can use the inferred schema when configuring a streaming source for your application. For conceptual information, see Configuring Application Input. Note that when you create an application using the Amazon Kinesis Analytics console, the console uses this operation to infer a schema and show it in the console user interface.

This operation requires permissions to perform the `kinesisanalytics:DiscoverInputSchema` action.

Request Syntax

```
{
   "InputStartingPositionConfiguration": {
      "InputStartingPosition": "string"
   },
   "ResourceARN": "string",
   "RoleARN": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

InputStartingPositionConfiguration (p. 120)

Point at which you want Amazon Kinesis Analytics to start reading records from the specified streaming source discovery purposes.

Type: InputStartingPositionConfiguration (p. 145) object

Required: Yes

ResourceARN (p. 120)

Amazon Resource Name (ARN) of the streaming source.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: Yes

RoleARN (p. 120)

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream on your behalf.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: Yes

Response Syntax

```
{
   "InputSchema": {
      "RecordColumns": [
```

```
            {
               "Mapping": "string",
               "Name": "string",
               "SqlType": "string"
            }
         ],
         "RecordEncoding": "string",
         "RecordFormat": {
            "MappingParameters": {
               "CSVMappingParameters": {
                  "RecordColumnDelimiter": "string",
                  "RecordRowDelimiter": "string"
               },
               "JSONMappingParameters": {
                  "RecordRowPath": "string"
               }
            },
            "RecordFormatType": "string"
         }
      },
      "ParsedInputRecords": [
         [ "string" ]
      ],
      "RawInputRecords": [ "string" ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

InputSchema (p. 120)

Schema inferred from the streaming source. It identifies the format of the data in the streaming source and how each data element maps to corresponding columns in the in-application stream that you can create.

Type: SourceSchema (p. 172) object

ParsedInputRecords (p. 120)

An array of elements, where each element corresponds to a row in a stream record (a stream record can have more than one row).

Type: array of array of Stringss

RawInputRecords (p. 120)

Raw stream data that was sampled to infer the schema.

Type: array of Strings

Errors

InvalidArgumentException

Specified input parameter value is invalid.

HTTP Status Code: 400

ResourceProvisionedThroughputExceededException

Discovery failed to get a record from the streaming source because of the Amazon Kinesis Streams ProvisionedThroughputExceededException. For more information, see GetRecords in the Amazon Kinesis Streams API Reference.

HTTP Status Code: 400

UnableToDetectSchemaException

Data format is not valid, Amazon Kinesis Analytics is not able to detect schema for the given streaming source.

HTTP Status Code: 400

ListApplications

Returns a list of Amazon Kinesis Analytics applications in your account. For each application, the response includes the application name, Amazon Resource Name (ARN), and status. If the response returns the `HasMoreApplications` value as true, you can send another request by adding the `ExclusiveStartApplicationName` in the request body, and set the value of this to the last application name from the previous response.

If you want detailed information about a specific application, use DescribeApplication (p. 117).

This operation requires permissions to perform the `kinesisanalytics:ListApplications` action.

Request Syntax

```
{
   "ExclusiveStartApplicationName": "string",
   "Limit": number
}
```

Request Parameters

The request accepts the following data in JSON format.

ExclusiveStartApplicationName (p. 123)

Name of the application to start the list with. When using pagination to retrieve the list, you don't need to specify this parameter in the first request. However, in subsequent requests, you add the last application name from the previous response to get the next page of applications.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: No

Limit (p. 123)

Maximum number of applications to list.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 50.

Required: No

Response Syntax

```
{
   "ApplicationSummaries": [
      {
         "ApplicationARN": "string",
         "ApplicationName": "string",
         "ApplicationStatus": "string"
      }
   ],
   "HasMoreApplications": boolean
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ApplicationSummaries (p. 123)

List of `ApplicationSummary` objects.

Type: array of ApplicationSummary (p. 134) objects

HasMoreApplications (p. 123)

Returns true if there are more applications to retrieve.

Type: Boolean

StartApplication

Starts the specified Amazon Kinesis Analytics application. After creating an application, you must exclusively call this operation to start your application.

After the application starts, it begins consuming the input data, processes it, and writes the output to the configured destination.

The application status must be READY for you to start an application. You can get the application status in the console or using the DescribeApplication (p. 117) operation.

After you start the application, you can stop the application from processing the input by calling the StopApplication (p. 127) operation.

This operation requires permissions to perform the `kinesisanalytics:StartApplication` action.

Request Syntax

```
{
   "ApplicationName": "string",
   "InputConfigurations": [
      {
         "Id": "string",
         "InputStartingPositionConfiguration": {
            "InputStartingPosition": "string"
         }
      }
   ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 125)

Name of the application.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

InputConfigurations (p. 125)

Identifies the specific input, by ID, that the application starts consuming. Amazon Kinesis Analytics starts reading the streaming source associated with the input. You can also specify where in the streaming source you want Amazon Kinesis Analytics to start reading.

Type: array of InputConfiguration (p. 139) objects

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

InvalidApplicationConfigurationException

User-provided application configuration is not valid.

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

HTTP Status Code: 400

StopApplication

Stops the application from processing input data. You can stop an application only if it is in the running state. You can use the DescribeApplication (p. 117) operation to find the application state. After the application is stopped, Amazon Kinesis Analytics stops reading data from the input, the application stops processing data, and there is no output written to the destination.

This operation requires permissions to perform the `kinesisanalytics:StopApplication` action.

Request Syntax

```
{
   "ApplicationName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 127)
Name of the running application to stop.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 128.
Pattern: `[a-zA-Z0-9_.-]+`
Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

ResourceInUseException
Application is not available for this operation.
HTTP Status Code: 400

ResourceNotFoundException
Specified application can't be found.
HTTP Status Code: 400

UpdateApplication

Updates an existing Amazon Kinesis Analytics application. Using this API, you can update application code, input configuration, and output configuration.

Note that Amazon Kinesis Analytics updates the `CurrentApplicationVersionId` each time you update your application.

This operation requires permission for the `kinesisanalytics:UpdateApplication` action.

Request Syntax

```
{
   "ApplicationName": "string",
   "ApplicationUpdate": {
      "ApplicationCodeUpdate": "string",
      "InputUpdates": [
         {
            "InputId": "string",
            "InputParallelismUpdate": {
               "CountUpdate": number
            },
            "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"
            },
            "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
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName (p. 128)

Name of the Amazon Kinesis Analytics application to update.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

ApplicationUpdate (p. 128)

Describes application updates.

Type: ApplicationUpdate (p. 135) object

Required: Yes

CurrentApplicationVersionId (p. 128)

The current application version ID. You can use the DescribeApplication (p. 117) operation to get this value.

Type: Long

Valid Range: Minimum value of 1. Maximum value of 999999999.

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

CodeValidationException

User-provided application code (query) is invalid. This can be a simple syntax error.

HTTP Status Code: 400

ConcurrentModificationException

Exception thrown as a result of concurrent modification to an application. For example, two individuals attempting to edit the same application at the same time.

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

HTTP Status Code: 400

Data Types

The following data types are supported:

- ApplicationDetail (p. 132)
- ApplicationSummary (p. 134)
- ApplicationUpdate (p. 135)
- CSVMappingParameters (p. 136)
- DestinationSchema (p. 137)
- Input (p. 138)
- InputConfiguration (p. 139)
- InputDescription (p. 140)
- InputParallelism (p. 142)
- InputParallelismUpdate (p. 143)
- InputSchemaUpdate (p. 144)

ApplicationDetail

Provides a description of the application, including the application Amazon Resource Name (ARN), status, latest version, and input and output configuration.

Contents

ApplicationARN
ARN of the application.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`
Required: Yes

ApplicationCode
Returns the application code that you provided to perform data analysis on any of the in-application streams in your application.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 51200.
Required: No

ApplicationDescription
Description of the application.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 1024.
Required: No

ApplicationName
Name of the application.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 128.
Pattern: `[a-zA-Z0-9_.-]+`
Required: Yes

ApplicationStatus
Status of the application.
Type: String
Valid Values: `DELETING | STARTING | STOPPING | READY | RUNNING | UPDATING`
Required: Yes

ApplicationVersionId
Provides the current application version.
Type: Long
Valid Range: Minimum value of 1. Maximum value of 999999999.
Required: Yes

CreateTimestamp
Timestamp when the application version was created.
Type: Timestamp
Required: No

InputDescriptions
Describes the application input configuration. For more information, see Configuring Application Input.
Type: array of InputDescription (p. 140) objects
Required: No

LastUpdateTimestamp

Timestamp when the application was last updated.

Type: Timestamp

Required: No

OutputDescriptions

Describes the application output configuration. For more information, see Configuring Application Output.

Type: array of OutputDescription (p. 162) objects

Required: No

ReferenceDataSourceDescriptions

Describes reference data sources configured for the application. For more information, see Configuring Application Input.

Type: array of ReferenceDataSourceDescription (p. 167) objects

Required: No

ApplicationSummary

Provides application summary information, including the application Amazon Resource Name (ARN), name, and status.

Contents

ApplicationARN

ARN of the application.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: Yes

ApplicationName

Name of the application.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

ApplicationStatus

Status of the application.

Type: String

Valid Values: `DELETING | STARTING | STOPPING | READY | RUNNING | UPDATING`

Required: Yes

ApplicationUpdate

Describes updates to apply to an existing Amazon Kinesis Analytics application.

Contents

ApplicationCodeUpdate
Describes application code updates.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 51200.
Required: No

InputUpdates
Describes application input configuration updates.
Type: array of InputUpdate (p. 146) objects
Required: No

OutputUpdates
Describes application output configuration updates.
Type: array of OutputUpdate (p. 163) objects
Required: No

ReferenceDataSourceUpdates
Describes application reference data source updates.
Type: array of ReferenceDataSourceUpdate (p. 168) objects
Required: No

CSVMappingParameters

Provides additional mapping information when the record format uses delimiters, such as CSV. For example, the following sample records use CSV format, where the records use the *'\n'* as the row delimiter and a comma (",") as the column delimiter:

```
"name1", "address1"
"name2, "address2"
```

Contents

RecordColumnDelimiter

Column delimiter. For example, in a CSV format, a comma (",") is the typical column delimiter.

Type: String

Required: Yes

RecordRowDelimiter

Row delimiter. For example, in a CSV format, *'\n'* is the typical row delimiter.

Type: String

Required: Yes

DestinationSchema

Describes the data format when records are written to the destination. For more information, see Configuring Application Output.

Contents

RecordFormatType

Specifies the format of the records on the output stream.

Type: String

Valid Values: `JSON` | `CSV`

Required: No

Input

When you configure the application input, you specify the streaming source, the in-application stream name that is created, and the mapping between the two. For more information, see Configuring Application Input.

Contents

InputParallelism

Describes the number of in-application streams to create.

Data from your source will be routed to these in-application input streams.

(see Configuring Application Input.

Type: InputParallelism (p. 142) object

Required: No

InputSchema

Describes the format of the data in the streaming source, and how each data element maps to corresponding columns in the in-application stream that is being created.

Also used to describe the format of the reference data source.

Type: SourceSchema (p. 172) object

Required: No

KinesisFirehoseInput

If the streaming source is an Amazon Kinesis Firehose delivery stream, identifies the Firehose delivery stream's ARN and an IAM role that enables Amazon Kinesis Analytics to access the stream on your behalf.

Type: KinesisFirehoseInput (p. 148) object

Required: No

KinesisStreamsInput

If the streaming source is an Amazon Kinesis stream, identifies the stream's Amazon Resource Name (ARN) and an IAM role that enables Amazon Kinesis Analytics to access the stream on your behalf.

Type: KinesisStreamsInput (p. 154) object

Required: No

NamePrefix

Name prefix to use when creating in-application stream. Suppose you specify a prefix "MyInApplicationStream". Amazon Kinesis Analytics will then create one or more (as per the `InputParallelism` count you specified) in-application streams with names "MyInApplicationStream_001", "MyInApplicationStream_002" and so on.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `[a-zA-Z][a-zA-Z0-9_]+`

Required: Yes

InputConfiguration

When you start your application, you provide this configuration, which identifies the input source and the point in the input source at which you want the application to start processing records.

Contents

Id

Input source ID. You can get this ID by calling the DescribeApplication (p. 117) operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

InputStartingPositionConfiguration

Point at which you want the application to start processing records from the streaming source.

Type: InputStartingPositionConfiguration (p. 145) object

Required: Yes

InputDescription

Describes the application input configuration. For more information, see Configuring Application Input.

Contents

InAppStreamNames
Returns the in-application stream names that are mapped to the stream source.
Type: array of Strings
Length Constraints: Minimum length of 1. Maximum length of 32.
Pattern: `[a-zA-Z][a-zA-Z0-9_]+`
Required: No

InputId
Input ID associated with the application input. This is the ID that Amazon Kinesis Analytics assigns to each input configuration you add to your application.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 50.
Pattern: `[a-zA-Z0-9_.-]+`
Required: No

InputParallelism
Describes the configured parallelism (number of in-application streams mapped to the streaming source).
Type: InputParallelism (p. 142) object
Required: No

InputSchema
Describes the format of the data in the streaming source, and how each data element maps to corresponding columns created in the in-application stream.
Type: SourceSchema (p. 172) object
Required: No

InputStartingPositionConfiguration
Point at which the application is configured to read from the input stream.
Type: InputStartingPositionConfiguration (p. 145) object
Required: No

KinesisFirehoseInputDescription
If an Amazon Kinesis Firehose delivery stream is configured as a streaming source, provides the Firehose delivery stream's Amazon Resource Name (ARN) and an IAM role that enables Amazon Kinesis Analytics to access the stream on your behalf.
Type: KinesisFirehoseInputDescription (p. 149) object
Required: No

KinesisStreamsInputDescription
If an Amazon Kinesis stream is configured as streaming source, provides Amazon Kinesis stream's ARN and an IAM role that enables Amazon Kinesis Analytics to access the stream on your behalf.
Type: KinesisStreamsInputDescription (p. 155) object
Required: No

NamePrefix
In-application name prefix.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 32.
Pattern: `[a-zA-Z][a-zA-Z0-9_]+`
Required: No

InputParallelism

Describes the number of in-application streams to create for a given streaming source. For information about parallelism, see Configuring Application Input.

Contents

Count

Number of in-application streams to create. For more information, see Limits.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

Required: No

InputParallelismUpdate

Provides updates to the parallelism count.

Contents

CountUpdate

Number of in-application streams to create for the specified streaming source.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

Required: No

InputSchemaUpdate

Describes updates for the application's input schema.

Contents

RecordColumnUpdates

A list of `RecordColumn` objects. Each object describes the mapping of the streaming source element to the corresponding column in the in-application stream.

Type: array of RecordColumn (p. 164) objects

Array Members: Minimum number of 1 item. Maximum number of 1000 items.

Required: No

RecordEncodingUpdate

Specifies the encoding of the records in the streaming source. For example, UTF-8.

Type: String

Pattern: `UTF-8`

Required: No

RecordFormatUpdate

Specifies the format of the records on the streaming source.

Type: RecordFormat (p. 165) object

Required: No

InputStartingPositionConfiguration

Describes the point at which the application reads from the streaming source.

Contents

InputStartingPosition

The starting position on the stream.

- `NOW` - Start reading just after the most recent record in the stream, start at the request timestamp that the customer issued.
- `TRIM_HORIZON` - Start reading at the last untrimmed record in the stream, which is the oldest record available in the stream. This option is not available for an Amazon Kinesis Firehose delivery stream.
- `LAST_STOPPED_POINT` - Resume reading from where the application last stopped reading.

Type: String

Valid Values: `NOW | TRIM_HORIZON | LAST_STOPPED_POINT`

Required: No

InputUpdate

Describes updates to a specific input configuration (identified by the `InputId` of an application).

Contents

InputId
Input ID of the application input to be updated.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 50.
Pattern: `[a-zA-Z0-9_.-]+`
Required: Yes

InputParallelismUpdate
Describes the parallelism updates (the number in-application streams Amazon Kinesis Analytics creates for the specific streaming source).
Type: InputParallelismUpdate (p. 143) object
Required: No

InputSchemaUpdate
Describes the data format on the streaming source, and how record elements on the streaming source map to columns of the in-application stream that is created.
Type: InputSchemaUpdate (p. 144) object
Required: No

KinesisFirehoseInputUpdate
If an Amazon Kinesis Firehose delivery stream is the streaming source to be updated, provides an updated stream Amazon Resource Name (ARN) and IAM role ARN.
Type: KinesisFirehoseInputUpdate (p. 150) object
Required: No

KinesisStreamsInputUpdate
If a Amazon Kinesis stream is the streaming source to be updated, provides an updated stream ARN and IAM role ARN.
Type: KinesisStreamsInputUpdate (p. 156) object
Required: No

NamePrefixUpdate
Name prefix for in-application streams that Amazon Kinesis Analytics creates for the specific streaming source.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 32.
Pattern: `[a-zA-Z][a-zA-Z0-9_]+`
Required: No

JSONMappingParameters

Provides additional mapping information when JSON is the record format on the streaming source.

Contents

RecordRowPath

Path to the top-level parent that contains the records.

For example, consider the following JSON record:

In the `RecordRowPath`, `"$"` refers to the root and path `"$.vehicle.Model"` refers to the specific `"Model"` key in the JSON.

Type: String

Required: Yes

KinesisFirehoseInput

Identifies an Amazon Kinesis Firehose delivery stream as the streaming source. You provide the Firehose delivery stream's Amazon Resource Name (ARN) and an IAM role ARN that enables Amazon Kinesis Analytics to access the stream on your behalf.

Contents

ResourceARN

ARN of the input Firehose delivery stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: Yes

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream on your behalf. You need to make sure the role has necessary permissions to access the stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: Yes

KinesisFirehoseInputDescription

Describes the Amazon Kinesis Firehose delivery stream that is configured as the streaming source in the application input configuration.

Contents

ResourceARN

Amazon Resource Name (ARN) of the Amazon Kinesis Firehose delivery stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: No

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics assumes to access the stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: No

KinesisFirehoseInputUpdate

When updating application input configuration, provides information about an Amazon Kinesis Firehose delivery stream as the streaming source.

Contents

ResourceARNUpdate

ARN of the input Amazon Kinesis Firehose delivery stream to read.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: No

RoleARNUpdate

Amazon Resource Name (ARN) of the IAM role that Amazon Kinesis Analytics can assume to access the stream on your behalf. You need to grant necessary permissions to this role.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: No

KinesisFirehoseOutput

When configuring application output, identifies an Amazon Kinesis Firehose delivery stream as the destination. You provide the stream Amazon Resource Name (ARN) and an IAM role that enables Amazon Kinesis Analytics to write to the stream on your behalf.

Contents

ResourceARN

ARN of the destination Amazon Kinesis Firehose delivery stream to write to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: Yes

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to write to the destination stream on your behalf. You need to grant the necessary permissions to this role.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: Yes

KinesisFirehoseOutputDescription

For an application output, describes the Amazon Kinesis Firehose delivery stream configured as its destination.

Contents

ResourceARN

Amazon Resource Name (ARN) of the Amazon Kinesis Firehose delivery stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: No

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: No

KinesisFirehoseOutputUpdate

When updating an output configuration using the UpdateApplication (p. 128) operation, provides information about an Amazon Kinesis Firehose delivery stream configured as the destination.

Contents

ResourceARNUpdate

Amazon Resource Name (ARN) of the Amazon Kinesis Firehose delivery stream to write to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: No

RoleARNUpdate

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream on your behalf. You need to grant necessary permissions to this role.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: No

KinesisStreamsInput

Identifies an Amazon Kinesis stream as the streaming source. You provide the stream's ARN and an IAM role ARN that enables Amazon Kinesis Analytics to access the stream on your behalf.

Contents

ResourceARN

ARN of the input Amazon Kinesis stream to read.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: Yes

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream on your behalf. You need to grant the necessary permissions to this role.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: Yes

KinesisStreamsInputDescription

Describes the Amazon Kinesis stream that is configured as the streaming source in the application input configuration.

Contents

ResourceARN

Amazon Resource Name (ARN) of the Amazon Kinesis stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: No

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: No

KinesisStreamsInputUpdate

When updating application input configuration, provides information about an Amazon Kinesis stream as the streaming source.

Contents

ResourceARNUpdate

Amazon Resource Name (ARN) of the input Amazon Kinesis stream to read.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: No

RoleARNUpdate

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream on your behalf. You need to grant the necessary permissions to this role.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: No

KinesisStreamsOutput

When configuring application output, identifies a Amazon Kinesis stream as the destination. You provide the stream Amazon Resource Name (ARN) and also an IAM role ARN that Amazon Kinesis Analytics can use to write to the stream on your behalf.

Contents

ResourceARN

ARN of the destination Amazon Kinesis stream to write to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: Yes

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to write to the destination stream on your behalf. You need to grant the necessary permissions to this role.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: Yes

KinesisStreamsOutputDescription

For an application output, describes the Amazon Kinesis stream configured as its destination.

Contents

ResourceARN

Amazon Resource Name (ARN) of the Amazon Kinesis stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: No

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: No

KinesisStreamsOutputUpdate

When updating an output configuration using the UpdateApplication (p. 128) operation, provides information about an Amazon Kinesis stream configured as the destination.

Contents

ResourceARNUpdate

Amazon Resource Name (ARN) of the Amazon Kinesis stream where you want to write the output.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]+:[a-zA-Z0-9\-]*:\d{12}:[a-zA-Z_0-9+=,.@\-_/:]+`

Required: No

RoleARNUpdate

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream on your behalf. You need to grant the necessary permissions to this role.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: No

MappingParameters

When configuring application input at the time of creating or updating an application, provides additional mapping information specific to the record format (such as JSON, CSV, or record fields delimited by some delimiter) on the streaming source.

Contents

CSVMappingParameters

Provides additional mapping information when the record format uses delimiters (for example, CSV).

Type: CSVMappingParameters (p. 136) object

Required: No

JSONMappingParameters

Provides additional mapping information when JSON is the record format on the streaming source.

Type: JSONMappingParameters (p. 147) object

Required: No

Output

Describes application output configuration in which you identify an in-application stream and a destination where you want the in-application stream data to be written. The destination can be an Amazon Kinesis stream or an Amazon Kinesis Firehose delivery stream.

For limits on how many destinations an application can write and other limitations, see Limits.

Contents

DestinationSchema

Describes the data format when records are written to the destination. For more information, see Configuring Application Output.

Type: DestinationSchema (p. 137) object

Required: Yes

KinesisFirehoseOutput

Identifies an Amazon Kinesis Firehose delivery stream as the destination.

Type: KinesisFirehoseOutput (p. 151) object

Required: No

KinesisStreamsOutput

Identifies an Amazon Kinesis stream as the destination.

Type: KinesisStreamsOutput (p. 157) object

Required: No

Name

Name of the in-application stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `[a-zA-Z][a-zA-Z0-9_]+`

Required: Yes

OutputDescription

Describes the application output configuration, which includes the in-application stream name and the destination where the stream data is written. The destination can be an Amazon Kinesis stream or an Amazon Kinesis Firehose delivery stream.

Contents

DestinationSchema
Data format used for writing data to the destination.
Type: DestinationSchema (p. 137) object
Required: No

KinesisFirehoseOutputDescription
Describes the Amazon Kinesis Firehose delivery stream configured as the destination where output is written.
Type: KinesisFirehoseOutputDescription (p. 152) object
Required: No

KinesisStreamsOutputDescription
Describes Amazon Kinesis stream configured as the destination where output is written.
Type: KinesisStreamsOutputDescription (p. 158) object
Required: No

Name
Name of the in-application stream configured as output.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 32.
Pattern: `[a-zA-Z][a-zA-Z0-9_]+`
Required: No

OutputId
A unique identifier for the output configuration.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 50.
Pattern: `[a-zA-Z0-9_.-]+`
Required: No

OutputUpdate

Describes updates to the output configuration identified by the `OutputId`.

Contents

DestinationSchemaUpdate
Describes the data format when records are written to the destination. For more information, see Configuring Application Output.
Type: DestinationSchema (p. 137) object
Required: No

KinesisFirehoseOutputUpdate
Describes a Amazon Kinesis Firehose delivery stream as the destination for the output.
Type: KinesisFirehoseOutputUpdate (p. 153) object
Required: No

KinesisStreamsOutputUpdate
Describes an Amazon Kinesis stream as the destination for the output.
Type: KinesisStreamsOutputUpdate (p. 159) object
Required: No

NameUpdate
If you want to specify a different in-application stream for this output configuration, use this field to specify the new in-application stream name.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 32.
Pattern: `[a-zA-Z][a-zA-Z0-9_]+`
Required: No

OutputId
Identifies the specific output configuration that you want to update.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 50.
Pattern: `[a-zA-Z0-9_.-]+`
Required: Yes

RecordColumn

Describes the mapping of each data element in the streaming source to the corresponding column in the in-application stream.

Also used to describe the format of the reference data source.

Contents

Mapping

Reference to the data element in the streaming input of the reference data source.

Type: String

Required: No

Name

Name of the column created in the in-application input stream or reference table.

Type: String

Pattern: `[a-zA-Z][a-zA-Z0-9_]+`

Required: Yes

SqlType

Type of column created in the in-application input stream or reference table.

Type: String

Required: Yes

RecordFormat

Describes the record format and relevant mapping information that should be applied to schematize the records on the stream.

Contents

MappingParameters

When configuring application input at the time of creating or updating an application, provides additional mapping information specific to the record format (such as JSON, CSV, or record fields delimited by some delimiter) on the streaming source.

Type: MappingParameters (p. 160) object

Required: No

RecordFormatType

The type of record format.

Type: String

Valid Values: `JSON` | `CSV`

Required: Yes

ReferenceDataSource

Describes the reference data source by providing the source information (S3 bucket name and object key name), the resulting in-application table name that is created, and the necessary schema to map the data elements in the Amazon S3 object to the in-application table.

Contents

ReferenceSchema

Describes the format of the data in the streaming source, and how each data element maps to corresponding columns created in the in-application stream.

Type: SourceSchema (p. 172) object

Required: Yes

S3ReferenceDataSource

Identifies the S3 bucket and object that contains the reference data. Also identifies the IAM role Amazon Kinesis Analytics can assume to read this object on your behalf.

An Amazon Kinesis Analytics application loads reference data only once. If the data changes, you call the UpdateApplication (p. 128) operation to trigger reloading of data into your application.

Type: S3ReferenceDataSource (p. 169) object

Required: No

TableName

Name of the in-application table to create.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `[a-zA-Z][a-zA-Z0-9_]+`

Required: Yes

ReferenceDataSourceDescription

Describes the reference data source configured for an application.

Contents

ReferenceId

ID of the reference data source. This is the ID that Amazon Kinesis Analytics assigns when you add the reference data source to your application using the AddApplicationReferenceDataSource (p. 106) operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

ReferenceSchema

Describes the format of the data in the streaming source, and how each data element maps to corresponding columns created in the in-application stream.

Type: SourceSchema (p. 172) object

Required: No

S3ReferenceDataSourceDescription

Provides the S3 bucket name, the object key name that contains the reference data. It also provides the Amazon Resource Name (ARN) of the IAM role that Amazon Kinesis Analytics can assume to read the Amazon S3 object and populate the in-application reference table.

Type: S3ReferenceDataSourceDescription (p. 170) object

Required: Yes

TableName

The in-application table name created by the specific reference data source configuration.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `[a-zA-Z][a-zA-Z0-9_]+`

Required: Yes

ReferenceDataSourceUpdate

When you update a reference data source configuration for an application, this object provides all the updated values (such as the source bucket name and object key name), the in-application table name that is created, and updated mapping information that maps the data in the Amazon S3 object to the in-application reference table that is created.

Contents

ReferenceId

ID of the reference data source being updated. You can use the DescribeApplication (p. 117) operation to get this value.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

ReferenceSchemaUpdate

Describes the format of the data in the streaming source, and how each data element maps to corresponding columns created in the in-application stream.

Type: SourceSchema (p. 172) object

Required: No

S3ReferenceDataSourceUpdate

Describes the S3 bucket name, object key name, and IAM role that Amazon Kinesis Analytics can assume to read the Amazon S3 object on your behalf and populate the in-application reference table.

Type: S3ReferenceDataSourceUpdate (p. 171) object

Required: No

TableNameUpdate

In-application table name that is created by this update.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `[a-zA-Z][a-zA-Z0-9_]+`

Required: No

S3ReferenceDataSource

Identifies the S3 bucket and object that contains the reference data. Also identifies the IAM role Amazon Kinesis Analytics can assume to read this object on your behalf.

An Amazon Kinesis Analytics application loads reference data only once. If the data changes, you call the UpdateApplication (p. 128) operation to trigger reloading of data into your application.

Contents

BucketARN

Amazon Resource Name (ARN) of the S3 bucket.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:.*`

Required: Yes

FileKey

Object key name containing reference data.

Type: String

Required: Yes

ReferenceRoleARN

ARN of the IAM role that the service can assume to read data on your behalf. This role must have permission for the `s3:GetObject` action on the object and trust policy that allows Amazon Kinesis Analytics service principal to assume this role.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: Yes

S3ReferenceDataSourceDescription

Provides the bucket name and object key name that stores the reference data.

Contents

BucketARN

Amazon Resource Name (ARN) of the S3 bucket.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:.*`

Required: Yes

FileKey

Amazon S3 object key name.

Type: String

Required: Yes

ReferenceRoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to read the Amazon S3 object on your behalf to populate the in-application reference table.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: Yes

S3ReferenceDataSourceUpdate

Describes the S3 bucket name, object key name, and IAM role that Amazon Kinesis Analytics can assume to read the Amazon S3 object on your behalf and populate the in-application reference table.

Contents

BucketARNUpdate

Amazon Resource Name (ARN) of the S3 bucket.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:.*`

Required: No

FileKeyUpdate

Object key name.

Type: String

Required: No

ReferenceRoleARNUpdate

ARN of the IAM role that Amazon Kinesis Analytics can assume to read the Amazon S3 object and populate the in-application.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: No

SourceSchema

Describes the format of the data in the streaming source, and how each data element maps to corresponding columns created in the in-application stream.

Contents

RecordColumns

A list of `RecordColumn` objects.

Type: array of RecordColumn (p. 164) objects

Array Members: Minimum number of 1 item. Maximum number of 1000 items.

Required: Yes

RecordEncoding

Specifies the encoding of the records in the streaming source. For example, UTF-8.

Type: String

Pattern: `UTF-8`

Required: No

RecordFormat

Specifies the format of the records on the streaming source.

Type: RecordFormat (p. 165) object

Required: Yes

Document History for the Amazon Kinesis Analytics

The following table describes the documentation for this release of Amazon Kinesis Analytics.

- **API version: 2015-08-14**
- **Latest documentation update:** August 11, 2016

Change	Description	Date
Public Release	Public release of the *Amazon Kinesis Analytics Developer Guide*.	August 11, 2016
Preview release	Preview release of the *Amazon Kinesis Analytics Developer Guide*.	January 29, 2016

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.