# Table of Contents

What Is Amazon Kinesis Data Analytics? ................................................................. 1
When Should I Use Amazon Kinesis Data Analytics? ............................................. 1
Are You a First-Time User of Amazon Kinesis Data Analytics? ......................... 1
Kinesis Data Analytics SQL Reference ................................................................. 3
How It Works ........................................................................................................ 4
Input ....................................................................................................................... 6
  Configuring a Streaming Source ........................................................................ 6
  Configuring a Reference Source ....................................................................... 8
  Working with JSONPath .................................................................................... 9
Mapping Streaming Source Elements to SQL Input Columns .......................... 13
Using the Schema Discovery Feature on Streaming Data ................................. 17
Using the Schema Discovery Feature on Static Data ......................................... 18
Using the Lambda Preprocessing Feature ......................................................... 21
Parallelizing Input Streams for Increased Throughput ...................................... 26
Application Code .............................................................................................. 29
Output ............................................................................................................... 30
  Creating an Output Using the AWS CLI ........................................................... 31
  Using a Lambda Function as Output ............................................................... 32
  Application Output Delivery Model ............................................................... 36
Error Handling .................................................................................................. 36
  Reporting Errors Using an In-Application Error Stream .................................. 37
Granting Permissions ......................................................................................... 37
  Trust Policy .................................................................................................... 38
  Permissions Policy ......................................................................................... 38
Auto Scaling Applications .................................................................................. 40
Getting Started ................................................................................................. 41
Step 1: Set Up an Account ................................................................................. 41
  Sign Up for AWS ............................................................................................ 41
  Create an IAM User ....................................................................................... 42
  Next Step ..................................................................................................... 42
Step 2: Set Up the AWS CLI .............................................................................. 42
  Next Step ..................................................................................................... 43
Step 3: Create Your Starter Analytics Application .......................................... 43
  Step 3.1: Create an Application ..................................................................... 45
  Step 3.2: Configure Input ............................................................................... 46
  Step 3.3: Add Real-Time Analytics (Add Application Code) ......................... 48
  Step 3.4: (Optional) Update the Application Code ......................................... 50
Step 4 (Optional) Edit the Schema and SQL Code Using the Console ............... 52
  Working with the Schema Editor .................................................................. 52
  Working with the SQL Editor ........................................................................ 59
Streaming SQL Concepts ..................................................................................... 62
  In-Application Streams and Pumps ............................................................... 62
  Timestamps and the ROWTIME Column ...................................................... 63
  Understanding Various Times in Streaming Analytics .................................. 63
Continuous Queries ........................................................................................... 65
Windowed Queries ............................................................................................ 66
  Tumbling Windows ....................................................................................... 66
  Sliding Windows .......................................................................................... 67
Stream Joins ....................................................................................................... 71
  Example 1: Report Orders Where There Are Trades Within One Minute of the Order Being Placed .................................................................................................................. 71
Example Applications ....................................................................................... 73
Examples: Preprocessing Streams ................................................................... 73
  Example: Manipulating Strings and Date Times ............................................. 73
<table>
<thead>
<tr>
<th>Data Types</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationDetail</td>
<td>186</td>
</tr>
<tr>
<td>ApplicationSummary</td>
<td>189</td>
</tr>
<tr>
<td>ApplicationUpdate</td>
<td>190</td>
</tr>
<tr>
<td>CloudWatchLoggingOption</td>
<td>191</td>
</tr>
<tr>
<td>CloudWatchLoggingOptionDescription</td>
<td>192</td>
</tr>
<tr>
<td>CloudWatchLoggingOptionUpdate</td>
<td>193</td>
</tr>
<tr>
<td>CSVMappingParameters</td>
<td>194</td>
</tr>
<tr>
<td>DestinationSchema</td>
<td>195</td>
</tr>
<tr>
<td>Input</td>
<td>196</td>
</tr>
<tr>
<td>InputConfiguration</td>
<td>198</td>
</tr>
<tr>
<td>InputDescription</td>
<td>199</td>
</tr>
<tr>
<td>InputLambdaProcessor</td>
<td>201</td>
</tr>
<tr>
<td>InputLambdaProcessorDescription</td>
<td>202</td>
</tr>
<tr>
<td>InputLambdaProcessorUpdate</td>
<td>203</td>
</tr>
<tr>
<td>InputParallelism</td>
<td>204</td>
</tr>
<tr>
<td>InputParallelismUpdate</td>
<td>205</td>
</tr>
<tr>
<td>InputProcessingConfiguration</td>
<td>206</td>
</tr>
<tr>
<td>InputProcessingConfigurationDescription</td>
<td>207</td>
</tr>
<tr>
<td>InputProcessingConfigurationUpdate</td>
<td>208</td>
</tr>
<tr>
<td>InputSchemaUpdate</td>
<td>209</td>
</tr>
<tr>
<td>InputStartingPositionConfiguration</td>
<td>210</td>
</tr>
<tr>
<td>InputUpdate</td>
<td>211</td>
</tr>
<tr>
<td>JSONMappingParameters</td>
<td>213</td>
</tr>
<tr>
<td>KinesisFirehoseInput</td>
<td>214</td>
</tr>
<tr>
<td>KinesisFirehoseInputDescription</td>
<td>215</td>
</tr>
<tr>
<td>KinesisFirehoseInputUpdate</td>
<td>216</td>
</tr>
<tr>
<td>KinesisFirehoseOutput</td>
<td>217</td>
</tr>
<tr>
<td>KinesisFirehoseOutputDescription</td>
<td>218</td>
</tr>
<tr>
<td>KinesisFirehoseOutputUpdate</td>
<td>219</td>
</tr>
<tr>
<td>KinesisStreamsInput</td>
<td>220</td>
</tr>
<tr>
<td>KinesisStreamsInputDescription</td>
<td>221</td>
</tr>
<tr>
<td>KinesisStreamsInputUpdate</td>
<td>222</td>
</tr>
<tr>
<td>KinesisStreamsOutput</td>
<td>223</td>
</tr>
<tr>
<td>KinesisStreamsOutputDescription</td>
<td>224</td>
</tr>
<tr>
<td>KinesisStreamsOutputUpdate</td>
<td>225</td>
</tr>
<tr>
<td>LambdaOutput</td>
<td>226</td>
</tr>
<tr>
<td>LambdaOutputDescription</td>
<td>227</td>
</tr>
<tr>
<td>LambdaOutputUpdate</td>
<td>228</td>
</tr>
<tr>
<td>MappingParameters</td>
<td>229</td>
</tr>
<tr>
<td>Output</td>
<td>230</td>
</tr>
<tr>
<td>OutputDescription</td>
<td>232</td>
</tr>
<tr>
<td>OutputUpdate</td>
<td>234</td>
</tr>
<tr>
<td>RecordColumn</td>
<td>236</td>
</tr>
<tr>
<td>RecordFormat</td>
<td>237</td>
</tr>
<tr>
<td>ReferenceDataSource</td>
<td>238</td>
</tr>
</tbody>
</table>
What Is Amazon Kinesis Data Analytics?

With Amazon Kinesis Data 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 Kinesis Data Analytics, you create a Kinesis data analytics application that continuously reads and processes streaming data. The service supports ingesting data from Amazon Kinesis Data Streams and Amazon Kinesis Data 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 Kinesis Data Analytics to send the results.

Kinesis Data Analytics supports Amazon Kinesis Data Firehose (Amazon S3, Amazon Redshift, and Amazon Elasticsearch Service), AWS Lambda, and Amazon Kinesis Data Streams as destinations.

When Should I Use Amazon Kinesis Data Analytics?

Amazon Kinesis Data 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 provide insights into your data. Following are some of example scenarios for using Kinesis Data Analytics:

- **Generate time-series analytics** – You can calculate metrics over time windows, and then stream values to Amazon S3 or Amazon Redshift through a Kinesis data 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.

For information about the SQL language elements that are supported by Kinesis Data Analytics, see [Amazon Kinesis Data Analytics SQL Reference](#).

Are You a First-Time User of Amazon Kinesis Data Analytics?

If you are a first-time user of Amazon Kinesis Data 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 Kinesis Data Analytics components that you work with to create an end-to-end experience. For more information, see [Amazon Kinesis Data Analytics: How It Works (p. 4)](#).

2. **Try the Getting Started exercises.** For more information, see [Getting Started with Amazon Kinesis Data Analytics (p. 41)](#).

3. **Explore the streaming SQL concepts.** For more information, see [Streaming SQL Concepts (p. 62)](#).
4. Try additional examples. For more information, see Example Amazon Kinesis Data Analytics Applications (p. 73).
Kinesis Data Analytics SQL Reference

For information about the SQL language elements that are supported by Amazon Kinesis Data Analytics, see Amazon Kinesis Data Analytics SQL Reference.
Amazon Kinesis Data Analytics: How It Works

An application is the primary resource in Amazon Kinesis Data Analytics that you can create in your account. You can create and manage applications using the AWS Management Console or the Amazon Kinesis Data Analytics API. Kinesis Data Analytics provides API operations to manage applications. For a list of API operations, see Actions (p. 138).

Amazon Kinesis data 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, Kinesis Data 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 Data 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, Kinesis Data Analytics updates the current application version ID. Kinesis Data Analytics also maintains time stamps for 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 a Kinesis data stream or a Kinesis data 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. 120) and Configuring Application Input (p. 6).
Amazon Kinesis Data Analytics provides a time stamp column in each application stream called Timestamps and the ROWTIME Column (p. 63). You can use this column in time-based windowed queries. For more information, see Windowed Queries (p. 66).

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 Data Analytics reads the Amazon S3 object and creates an in-application table. For more information, see Configuring Application Input (p. 6).

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

For information about the SQL language elements that are supported by Kinesis Data Analytics, see Amazon Kinesis Data Analytics SQL Reference.

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. 29).

- **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 Kinesis data delivery stream or a Kinesis data stream. Note the following about these destinations:
  - You can configure a Kinesis data delivery stream to write results to Amazon S3, Amazon Redshift, or Amazon Elasticsearch Service (Amazon ES).
  - You can also write application output to a custom destination, instead of Amazon S3 or Amazon Redshift. To do that, you specify a Kinesis data 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 Data Analytics.

For more information, see Configuring Application Output (p. 30).

In addition, note the following:

- Amazon Kinesis Data 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.
• Kinesis Data 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 Kinesis Data Analytics to persist the error stream data to an external destination for further evaluation. For more information, see Error Handling (p. 36).

• Amazon Kinesis Data 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 an External Destination (p. 36).

**Topics**

- Configuring Application Input (p. 6)
- Application Code (p. 29)
- Configuring Application Output (p. 30)
- Error Handling (p. 36)
- Granting Amazon Kinesis Data Analytics Permissions to Access Streaming Sources (Creating an IAM Role) (p. 37)
- Automatically Scaling Applications to Increase Throughput (p. 40)

**Configuring Application Input**

Your Amazon Kinesis Data Analytics application can receive input from a single streaming source and, optionally, use one reference data source. For more information, see Amazon Kinesis Data Analytics: How It Works (p. 4). The sections in this topic describe the application input sources.

**Configuring a Streaming Source**

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

- A Kinesis stream
- An Amazon Kinesis Data Firehose delivery stream

Amazon Kinesis Data 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 Data Analytics can assume to access the stream on your behalf.

- **In-application stream name prefix** – When you start the application, Amazon Kinesis Data 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. 120). In this case, Amazon Kinesis Data Analytics creates the specified number of in-application streams with names as follows: `prefix_001`, `prefix_002`, and `prefix_003`. By
default, Amazon Kinesis Data Analytics maps the streaming source to one in-application stream called \textit{prefix}_001.

There is a limit on the rate that you can insert rows in an in-application stream. Therefore, Amazon Kinesis Data 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.

\textbf{Note}

Amazon Kinesis Data 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 Data Analytics SQL Reference.

You can create an application and configure inputs in the Amazon Kinesis Data 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 \textit{CreateApplication (p. 152)} and \textit{AddApplicationInput (p. 141)}. The following is the input configuration part of the CreateApplication API request body:

\begin{verbatim}
"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"  
  }  
]  
\end{verbatim}
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 Data Analytics reads the Amazon 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 Amazon S3 object using supported formats (CSV, JSON). For example, suppose that your application performs analytics on stock orders. Assume the following record format on the streaming source:

<table>
<thead>
<tr>
<th>Ticker</th>
<th>SalePrice</th>
<th>OrderId</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMZN</td>
<td>$700</td>
<td>1003</td>
</tr>
<tr>
<td>XYZ</td>
<td>$250</td>
<td>1004</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Ticker</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMZN</td>
<td>Amazon</td>
</tr>
<tr>
<td>XYZ</td>
<td>SomeCompany</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Amazon Kinesis Data Analytics provides the following API actions to manage reference data sources:

- `AddApplicationReferenceDataSource` (p. 149)
- `UpdateApplication` (p. 181)

**Note**

Amazon Kinesis Data 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 Data Analytics Application (p. 87).

Note the following:

- If the application is running, Amazon Kinesis Data 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 Data Analytics only saves the updated input configuration. When the application starts running, Amazon Kinesis Data Analytics loads the reference data in your application as a table.

If you want to refresh the data after Amazon Kinesis Data Analytics creates the in-application reference table, perhaps because you updated the Amazon S3 object or you want to use a different Amazon S3 object, you must explicitly call the `UpdateApplication` (p. 181). Amazon Kinesis Data Analytics does not refresh the in-application reference table automatically.

There is a limit on the size of the Amazon S3 object that you can create as a reference data source. For more information, see Limits (p. 120). If the object size exceeds the limit, Amazon Kinesis Data 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 the bucket name and object key, you also provide an IAM role that Amazon Kinesis Data Analytics can assume to read the object on your behalf.
• **In-application reference table name** – Amazon Kinesis Data Analytics creates this in-application table and populates it by reading the Amazon 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 Amazon 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.

```json
{
  "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"
  }
}
```

### Working with JSONPath

JSONPath is a standardized way to query elements of a JSON object. JSONPath uses path expressions to navigate elements, nested elements, and arrays in a JSON document. For more information about JSON, see [Introducing JSON](#).

### Accessing JSON Elements with JSONPath

Following, you can find how to use JSONPath expressions to access various elements in JSON-formatted data. For the examples in this section, assume that the source stream contains a JSON record in the following format.

```json
{
  "customerName": "John Doe",
}
```
"address":
{
  "streetAddress":
  [
    "number":"123",
    "street":"AnyStreet"
  ],
  "city":"Anytown"
}

"orders":
[
  { "orderId":"23284", "itemName":"Widget", "itemPrice":"33.99" },
  { "orderId":"63122", "itemName":"Gadget", "itemPrice":"22.50" },
  { "orderId":"77284", "itemName":"Sprocket", "itemPrice":"12.00" }
]

Accessing JSON Elements

To query an element in JSON data using JSONPath, use the following syntax. Here, $ represents the root of the data hierarchy and elementName is the name of the element node to query.

$.elementName

The following expression queries the customerName element in the preceding JSON example.

$.customerName

The preceding expression returns the following from the preceding JSON record.

John Doe

Note
Path expressions are case-sensitive. The expression $.Name returns null from the preceding JSON example.

Note
If no element appears at the location where the path expression specifies, the expression returns null. The following expression returns null from the preceding JSON example, because there is no matching element.

$.customerId

Accessing Nested JSON Elements

To query a nested JSON element, use the following syntax.

$.parentElement.element

The following expression queries the city element in the preceding JSON example.

$.address.city

The preceding expression returns the following from the preceding JSON record.
Anytown

You can query further levels of subelements using the following syntax.

$.parentElement.element.subElement

The following expression queries the `street` element in the preceding JSON example.

$.address.streetAddress.street

The preceding expression returns the following from the preceding JSON record.

AnyStreet

**Accessing Arrays**

Arrays are queried using an array index expression inside square brackets (`[]`). Currently, the only index expression supported is `0:`, meaning that all the elements in the array are returned.

The format of the data returned depends on whether the array index expression is the last expression in the path:

- When the array index is the last expression in the path expression, all of the contents of the array are returned as a single field in a single data row.
- When there is a nested expression after the array index expression, the array is "flattened." In other words, each element in the array is returned as a separate data row.

To query the entire contents of an array as a single row, use the following syntax.

`.arrayObject[0:]`

The following expression queries the `orders` element in the preceding JSON example. It returns the array contents in a single column in a single row.

`.orders[0:]`

The preceding expression returns the following from the preceding JSON record.

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

To query the individual elements in an array as separate rows, use the following syntax.

`.arrayObject[0:].element`

The following expression queries the `orderId` elements in the preceding JSON example, and returns each array element as a separate row.

`.orders[0:].orderId`

The preceding expression returns the following from the preceding JSON record, with each data item returned as a separate row.
Note
If expressions that query nonarray elements are included in a schema that queries individual array elements, the nonarray elements are repeated for each element in the array. For example, suppose that a schema for the preceding JSON example includes the following expressions:

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

In this case, the returned data rows from the sample input stream element resemble the following, with the name element repeated for every orderId element.

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>23284</td>
</tr>
<tr>
<td>John Doe</td>
<td>63122</td>
</tr>
<tr>
<td>John Doe</td>
<td>77284</td>
</tr>
</tbody>
</table>

Note
The following limitations apply to array expressions in Amazon Kinesis Data Analytics:

- Only one level of dereferencing is supported in an array expression. The following expression format is not supported.

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

- Only one array can be flattened in a schema. Multiple arrays can be referenced—returned as one row containing all of the elements in the array. However, only one array can have each of its elements returned as individual rows.

A schema containing elements in the following format is valid. This format returns the contents of the second array as a single column, repeated for every element in the first array.

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

A schema containing elements in the following format is not valid.

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

Other Considerations
Additional considerations for working with JSONPath are as follows:

- If no arrays are accessed by individual element in the JSONPath expression, then a single row is created for each JSON record processed. Every JSONPath expression corresponds to a single column.
• When an array is flattened, any missing elements result in a null value being created in the in-application stream.

• An array is always flattened to at least one row. If no values would be returned (that is, the array is empty or none of its elements are queried), a single row with all null values is returned.

The following expression returns records with null values from the preceding JSON example, because there is no matching element at the specified path.

\$.orders[0:].itemId

The preceding expression returns the following from the preceding JSON example record.

null
null
null

Related Topics

• Introducing JSON

Mapping Streaming Source Elements to SQL Input Columns

With Amazon Kinesis Data Analytics, you can process and analyze streaming data in either JSON or CSV formats using standard SQL.

• To process and analyze streaming CSV data, you assign column names and data types for the columns of the input stream. Your application imports one column from the input stream per column definition, in order.

You don't have to include all of the columns in the application input stream, but you cannot skip columns from the source stream. For example, you can import the first three columns from an input stream containing five elements, but you cannot import only columns 1, 2, and 4.

• To process and analyze streaming JSON data, you use JSONPath expressions to map JSON elements from a streaming source to SQL columns in an input stream. For more information about using JSONPath with Amazon Kinesis Data Analytics, see Working with JSONPath (p. 9). The columns in the SQL table have data types that are mapped from JSON types. For supported data types, see Data Types. For details about converting JSON data to SQL data, see Mapping JSON Data Types to SQL Data Types (p. 15).

For more information about how to configure input streams, see Configuring Application Input (p. 6).

Mapping JSON Data to SQL Columns

You can map JSON elements to input columns using the AWS Management Console or the Amazon Kinesis Data Analytics API.

• To map elements to columns using the AWS Management Console, see Working with the Schema Editor (p. 52).
Mapping Streaming Source Elements to SQL Input Columns

• To map elements to columns using the Kinesis Analytics API, see the following section.

To map JSON elements to columns in the in-application input stream, your schema needs the following information for each column:

• Source Expression: The JSONPath expression that identifies the location of the data for the column.
• Column Name: The name that your SQL queries use to reference the data.
• Data Type: The SQL data type for the column.

Using the API

To map elements from a streaming source to input columns, you can use the Amazon Kinesis API CreateApplication (p. 152) action. To create the in-application stream, you need to specify a schema to transform your data into a schematized version used in SQL. The CreateApplication (p. 152) action configures your application to receive input from a single streaming source. To map JSON elements or CSV columns to SQL columns, you create a RecordColumn (p. 236) object in the SourceSchema (p. 247) RecordColumns array. The RecordColumn (p. 236) object has the following schema.

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

The fields in the RecordColumn (p. 236) object have the following values:

• Mapping: The JSONPath expression that identifies the location of the data in the input stream record. This value is not present for an input schema for a source stream in CSV format.
• Name: The column name in the in-application SQL data stream.
• SqlType: The data type of the data in the in-application SQL data stream.

JSON Input Schema Example

The following example demonstrates the format of the InputSchema value for a JSON schema.

```
"InputSchema": {
    "RecordColumns": [
        {
            "SqlType": "VARCHAR(4)",
            "Name": "TICKER_SYMBOL",
            "Mapping": ".TICKER_SYMBOL"
        },
        {
            "SqlType": "VARCHAR(16)",
            "Name": "SECTOR",
            "Mapping": ".SECTOR"
        },
        {
            "SqlType": "TINYINT",
            "Name": "CHANGE",
            "Mapping": ".CHANGE"
        }
    ]
}
```
Mapping Streaming Source Elements to SQL Input Columns

```
{ "SqlType": "DECIMAL(5,2)",
  "Name": "PRICE",
  "Mapping": "$.PRICE"
},

"RecordFormat": {
  "MappingParameters": {
    "JSONMappingParameters": {
      "RecordRowPath": "$"
    }
  },
  "RecordFormatType": "JSON"
},

"RecordEncoding": "UTF-8"
```

CSV Input Schema Example

The following example demonstrates the format of the `InputSchema` value for a schema in comma-separated value (CSV) format.

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

  "RecordFormat": {
    "MappingParameters": {
      "CSVMappingParameters": {
        "RecordColumnDelimiter": ",",
        "RecordRowDelimiter": "\n"
      }
    },
    "RecordFormatType": "CSV"
  },
  "RecordEncoding": "UTF-8"
}
```

Mapping JSON Data Types to SQL Data Types

JSON data types are converted to corresponding SQL data types according to the application's input schema. For information on supported SQL data types, see Data Types. Amazon Kinesis Data Analytics converts JSON data types to SQL data types according to the following rules.

**Null Literal**

A null literal in the JSON input stream (that is, "City":null) converts to a SQL null regardless of destination data type.
Boolean Literal

A Boolean literal in the JSON input stream (that is, "Contacted":true) converts to SQL data as follows:

- Numeric (DECIMAL, INT, and so on): true converts to 1; false converts to 0.
- Binary (BINARY or VARBINARY):
  - true: Result has lowest bit set and remaining bits cleared.
  - false: Result has all bits cleared.

Conversion to VARBINARY results in a value 1 byte in length.
- BOOLEAN: Converts to the corresponding SQL BOOLEAN value.
- Character (CHAR or VARCHAR): Converts to the corresponding string value (true or false). The value is truncated to fit the length of the field.
- Datetime (DATE, TIME, or TIMESTAMP): Conversion fails and a coercion error is written to the error stream.

Number

A number literal in the JSON input stream (that is, "CustomerId":67321) converts to SQL data as follows:

- Numeric (DECIMAL, INT, and so on): Converts directly. If the converted value exceeds the size or precision of the target data type (that is, converting 123.4 to INT), conversion fails and a coercion error is written to the error stream.
- Binary (BINARY or VARBINARY): Conversion fails and a coercion error is written to the error stream.
- BOOLEAN:
  - 0: Converts to false.
  - All other numbers: Converts to true.
- Character (CHAR or VARCHAR): Converts to a string representation of the number.
- Datetime (DATE, TIME, or TIMESTAMP): Conversion fails and a coercion error is written to the error stream.

String

A string value in the JSON input stream (that is, "CustomerName":"John Doe") converts to SQL data as follows:

- Numeric (DECIMAL, INT, and so on): Amazon Kinesis Data Analytics attempts to convert the value to the target data type. If the value cannot be converted, conversion fails and a coercion error is written to the error stream.
- Binary (BINARY or VARBINARY): If the source string is a valid binary literal (that is, X'3F67A23A', with an even number of f), the value is converted to the target data type. Otherwise, conversion fails and a coercion error is written to the error stream.
- BOOLEAN: If the source string is "true", converts to true. This comparison is case-insensitive. Otherwise, converts to false.
- Character (CHAR or VARCHAR): Converts to the string value in the input. If the value is longer than the target data type, it is truncated and no error is written to the error stream.
- Datetime (DATE, TIME, or TIMESTAMP): If the source string is in a format that can be converted to the target value, the value is converted. Otherwise, conversion fails and a coercion error is written to the error stream.
Valid datetime formats include:
- "1992-02-14"
- "1992-02-14 18:35:44.0"

Array or Object

An array or object in the JSON input stream converts to SQL data as follows:

- Character (CHAR or VARCHAR): Converts to the source text of the array or object. See Accessing Arrays (p. 11).
- All other data types: Conversion fails and a coercion error is written to the error stream.

For an example of a JSON array, see Working with JSONPath (p. 9).

Related Topics

- Configuring Application Input (p. 6)
- Data Types
- Working with the Schema Editor (p. 52)
- CreateApplication (p. 152)
- RecordColumn (p. 236)
- SourceSchema (p. 247)

Using the Schema Discovery Feature on Streaming Data

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. 171) 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**
To use the Discovery API to generate a schema from a file stored in Amazon S3, see Using the Schema Discovery Feature on Static Data (p. 18).

The console uses the Discovery API to generate a schema for a specified streaming source. Using the console, you can also update the schema, including adding or removing columns, changing column names or data types, and so on. However, you need to make changes carefully to ensure that you do not create an invalid schema.

After you finalize a schema for your in-application stream, there are functions you can use to manipulate string and datetime values. You can use 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. 73).

Column Naming During Schema Discovery

During schema discovery, Amazon Kinesis Data Analytics tries to retain as much of the original column name as possible from the streaming input source, except in the following cases:
• The source stream column name is a reserved SQL keyword, such as TIMESTAMP, USER, VALUES, or YEAR.
• The source stream column name contains unsupported characters. Only letters, numbers, and the underscore character ( _) are supported.
• The source stream column name begins with a number.
• The source stream column name is longer than 100 characters.

If a column is renamed, the renamed schema column name begins with COL_. In some cases, none of the original column name can be retained, for example if the entire name is unsupported characters. In such a case, the column is named COL_#, with # being a number indicating the column’s place in the column order.

After discovery completes, you can update the schema using the console to add or remove columns, or change column names, data types, or data size.

Examples of Discovery-Suggested Column Names

<table>
<thead>
<tr>
<th>Source Stream Column Name</th>
<th>Discovery-Suggested Column Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td>COL_USER</td>
</tr>
<tr>
<td>USER@DOMAIN</td>
<td>COL_USERDOMAIN</td>
</tr>
<tr>
<td>@@</td>
<td>COL_0</td>
</tr>
</tbody>
</table>

Schema Discovery Issues

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

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

If 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. For more information about how to do this, see the String and Date Time manipulation section. For examples, see Example: Manipulating Strings and Date Times (p. 73).

Using the Schema Discovery Feature on Static Data

The schema discovery feature can generate a schema from either the data in a stream or data in a static file that is stored in an Amazon S3 bucket. If you want to generate a schema for a Kinesis Data Analytics application for reference purposes or when live streaming data isn't available, you can use the schema discovery feature on a static file that contains a sample of the data in the expected format of your streaming or reference data. Kinesis Data Analytics can run schema discovery on sample data from a JSON or CSV file that's stored in an Amazon S3 bucket. Using schema discovery on a data file uses the DiscoverInputSchema (p. 171) API with the S3Configuration parameter specified.

To run discovery on a static file, you provide the API with an S3Configuration structure with the following information:
• **BucketARN**: The Amazon Resource Name (ARN) of the Amazon S3 bucket that contains the file. For the format of an Amazon S3 bucket ARN, see *Amazon Resource Names (ARNs) and AWS Service Namespaces: Amazon Simple Storage Service (Amazon S3)*.

• **RoleARN**: The ARN of an IAM role with the `AmazonS3ReadOnlyAccess` policy. For information about how to add a policy to a role, see *Modifying a Role*.

• **FileKey**: The file name of the object.

**Note**
Generating a schema from a data file is currently not available in the AWS Management Console.

To generate a schema from an Amazon S3 object using the `DiscoverInputSchema` API

1. Make sure that you have the AWS CLI set up. For more information, see *Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 42)* in the Getting Started section.
2. Create a file named `data.csv` with the following contents:

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

4. Create an Amazon S3 bucket and upload the `data.csv` file you created. Note the ARN of the created bucket. For information about creating an Amazon S3 bucket and uploading a file, see *Getting Started with Amazon Simple Storage Service*.
5. Open the IAM console at [https://console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/). Create a role with the `AmazonS3ReadOnlyAccess` policy. Note the ARN of the new role. For information about creating a role, see *Creating a Role to Delegate Permissions to an AWS Service*. For information about how to add a policy to a role, see *Modifying a Role*.
6. Run the following `DiscoverInputSchema` command in the AWS CLI, substituting the ARNs for your Amazon S3 bucket and IAM role:

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

7. The response will look similar to the following:

```
{
   "InputSchema": {
      "RecordEncoding": "UTF-8",
      "RecordColumns": [
         {
            "SqlType": "INTEGER",
            "Name": "COL_year"
         },
         {
            "SqlType": "INTEGER",
            "Name": "COL_month"
         },
         {
            "SqlType": "VARCHAR(4)",
            "Name": "state"
         }
      ]
   }
}
```
"SqlType": "VARCHAR(64)",
"Name": "producer_type"
},
{ "SqlType": "VARCHAR(4)",
"Name": "energy_source"
},
{ "SqlType": "VARCHAR(16)",
"Name": "units"
},
{ "SqlType": "INTEGER",
"Name": "consumption"
}
], "RecordFormat": {
 "RecordFormatType": "CSV",
 "MappingParameters": {
 "CSVMappingParameters": {
 "RecordRowDelimiter": "\r\n",
 "RecordColumnDelimiter": "",
 
 }
 } 
 }
}
}
"

"RawInputRecords": [
"year,month,state,producer_type,energy_source,units,consumption
2001,1, AK, TotalElectricPowerIndustry, Coal, ShortTons, 47615
2001,1, AK, ElectricGeneratorsElectricUtilities, Coal, ShortTons, 16535
2001,1, AK, CombinedHeatandPowerElectricPower, Coal, ShortTons, 22890
2001,1, AL, TotalElectricPowerIndustry, Coal, ShortTons, 3020601
2001,1, AL, ElectricGeneratorsElectricUtilities, Coal, ShortTons, 2987681"
],

"ParsedInputRecords": [
[null, null, "state", "producer_type", "energy_source", "units", null
],
],
["2001", "1", "AK", "ElectricGeneratorsElectricUtilities", "Coal", "ShortTons", "16535"
],
Using the Lambda Preprocessing Feature

Using the Lambda Preprocessing Feature

If the data in your stream needs format conversion, transformation, enrichment, or filtering, you can preprocess the data using an AWS Lambda function before your application SQL code executes, or before your application creates a schema from your data stream.

Using a Lambda function for preprocessing records is useful in the following scenarios:

- Transforming records from other formats (such as KPL or GZIP) into formats that Kinesis Data Analytics can analyze. Kinesis Data Analytics currently supports JSON or CSV data formats.
- Expanding data into a format that is more accessible for operations such as aggregation or anomaly detection. For instance, if several data values are stored together in a string, you can expand the data into separate columns.
- Data enrichment with other AWS services, such as extrapolation or error correction.
- Applying complex string transformation to record fields.
- Data filtering for cleaning up the data.

Using a Lambda Function for Preprocessing Records

When creating your Kinesis Data Analytics application, you enable Lambda preprocessing in the Connect to a Source page.

To use a Lambda function to preprocess records in a Kinesis Data Analytics application

1. On the Connect to a Source page, choose Enabled in the Record pre-processing with AWS Lambda section.
2. To use a Lambda function that you have already created, choose the function from the Lambda function drop-down list.
3. To create a new Lambda function from one of the the section called “Lambda Preprocessing Templates” (p. 22), choose the template from the drop-down list, and choose the View <template name> in Lambda link to edit the function.
4. To create a new Lambda function, choose **Create new**. For information about creating a Lambda function, see Create a HelloWorld Lambda Function and Explore the Console.

5. Choose the version of the Lambda function to use. To use the latest version, choose **$LATEST**.

When you choose or create a Lambda function for record preprocessing, the records are preprocessed before your application SQL code executes, or your application generates a schema from the records.

### Lambda Preprocessing Permissions

To use Lambda preprocessing, the application's IAM role requires the following permissions policy:

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

For more information about adding permissions policies, see Authentication and Access Control for Amazon Kinesis Data Analytics (p. 126).

### Lambda Preprocessing Metrics

You can monitor the number of Lambda invocations, bytes processed, successes and failures, etc. using Amazon CloudWatch. For information about CloudWatch metrics that are emitted by Kinesis Data Analytics Lambda preprocessing, see Amazon Kinesis Analytics Metrics.

### Lambda Preprocessing Templates

Amazon Kinesis Data Analytics provides the following templates for Lambda functions for preprocessing records in a stream:

<table>
<thead>
<tr>
<th>Lambda Blueprint</th>
<th>Language and version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Kinesis Data Analytics Input Processing</td>
<td>Node.js 6.10</td>
<td>A Kinesis Data Analytics record pre-processor that receives JSON or CSV records as input and then returns them with a processing status. Use this processor as a starting point for custom transformation logic.</td>
</tr>
<tr>
<td>General Kinesis Analytics Input Processing</td>
<td>Python 2.7</td>
<td>A Kinesis Data Analytics record pre-processor that receives JSON or CSV records as input and then returns them with a processing status. Use this processor as a starting point for custom transformation logic.</td>
</tr>
<tr>
<td>Compressed Input Processing</td>
<td>Node.js 6.10</td>
<td>A Kinesis Data Analytics record processor that receives compressed (GZIP or Deflate compressed) JSON or CSV records as input and returns decompressed records with a processing status.</td>
</tr>
<tr>
<td>KPL Input Processing</td>
<td>Python 2.7</td>
<td>A Kinesis Data Analytics record processor that receives Kinesis Producer Library (KPL) aggregates</td>
</tr>
<tr>
<td>Lambda Blueprint</td>
<td>Language and version</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of JSON or CSV records as input and returns disaggregated records with a processing status.</td>
</tr>
</tbody>
</table>

### Data Preprocessing Event Input Data Model/ Record Response Model

To preprocess records, your Lambda function must be compliant with the required event input data and record response models.

#### Event Input Data Model

Kinesis Data Analytics continuously reads data from your Kinesis stream or Kinesis Data Firehose delivery stream. For each batch of records it retrieves, the service manages how each batch gets passed to your Lambda function. Your function receives a list of records as input. Within your function, you iterate through the list and apply your business logic to accomplish your preprocessing requirements (such as data format conversion or enrichment).

The input model to your preprocessing function varies slightly, depending on whether the data was received from a Kinesis stream or a Kinesis Data Firehose delivery stream.

If the source is a Kinesis Data Firehose delivery stream, the event input data model is as follows:

**Kinesis Data Firehose Request Data Model**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invocationId</td>
<td>The Lambda invocation Id (random GUID).</td>
</tr>
<tr>
<td>applicationArn</td>
<td>Kinesis Data Analytics application Amazon Resource Name (ARN)</td>
</tr>
<tr>
<td>streamArn</td>
<td>Delivery stream ARN</td>
</tr>
<tr>
<td>records</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordId</td>
<td>record ID (random GUID)</td>
</tr>
</tbody>
</table>

KinesisFirehoseRecordMetadata

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>approximateArrivalTime</td>
<td>Delivery record approximate arrival time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Base64-encoded source record payload</td>
</tr>
</tbody>
</table>

If the source is a Kinesis stream, the event input data model is as follows:

**Kinesis Streams Request Data Model**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invocationId</td>
<td>The Lambda invocation Id (random GUID).</td>
</tr>
<tr>
<td>applicationArn</td>
<td>Kinesis Data Analytics application Amazon Resource Name (ARN)</td>
</tr>
<tr>
<td>streamArn</td>
<td>Delivery stream ARN</td>
</tr>
<tr>
<td>records</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordId</td>
<td>record ID (random GUID)</td>
</tr>
</tbody>
</table>

KinesisFirehoseRecordMetadata

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>approximateArrivalTime</td>
<td>Delivery record approximate arrival time</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invocationId</td>
<td>The Lambda invocation Id (random GUID).</td>
</tr>
<tr>
<td>applicationArn</td>
<td>Kinesis Data Analytics application ARN</td>
</tr>
<tr>
<td>streamArn</td>
<td>Delivery stream ARN</td>
</tr>
<tr>
<td>records</td>
<td></td>
</tr>
</tbody>
</table>

#### records

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordId</td>
<td>record ID based off of Kinesis record sequence number</td>
</tr>
</tbody>
</table>

#### kinesisStreamRecordMetadata

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequenceNumber</td>
<td>sequence number from the Kinesis stream record</td>
</tr>
<tr>
<td>partitionKey</td>
<td>Partition key from the Kinesis stream record</td>
</tr>
<tr>
<td>shardId</td>
<td>ShardId from the Kinesis stream record</td>
</tr>
<tr>
<td>approximateArrivalTimestamp</td>
<td>approximate arrival time</td>
</tr>
</tbody>
</table>

| data           | Base64-encoded source record payload                                    |

### Record Response Model

All records returned from your Lambda preprocessing function (with record IDs) that are sent to the Lambda function must be returned. They must contain the following parameters, or Kinesis Data Analytics rejects them and treats it as a data preprocessing failure. The data payload part of the record can be transformed to accomplish preprocessing requirements.

### Response Data Model

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>records</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordId</td>
<td>The record ID is passed from Kinesis Data Analytics to Lambda during the invocation. The transformed record must contain the same record ID. Any mismatch between the ID of the original record and the ID of the transformed record is treated as a data preprocessing failure.</td>
</tr>
<tr>
<td>result</td>
<td>The status of the data transformation of the record. The possible values are:</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Ok</td>
<td>The record was transformed successfully. Kinesis Data Analytics ingests the record for SQL processing.</td>
</tr>
<tr>
<td>• Dropped</td>
<td>The record was dropped intentionally by your processing logic. Kinesis Data Analytics drops the record from SQL processing. The data payload field is optional for a Dropped record.</td>
</tr>
<tr>
<td>• ProcessingFailed</td>
<td>The record could not be transformed. Kinesis Data Analytics considers it unsuccessfully processed by your Lambda function and writes an error to the error stream. For more information about the error stream, see Error Handling (p. 36). The data payload field is optional for a ProcessingFailed record.</td>
</tr>
<tr>
<td>data</td>
<td>The transformed data payload, after base64-encoding. Each data payload can contain multiple JSON documents if the application ingestion data format is JSON, or can contain multiple CSV rows (with a row delimiter specified in each row) if the application ingestion data format is CSV. The Kinesis Data Analytics service successfully parses and processes data with either multiple JSON documents or CSV rows within the same data payload.</td>
</tr>
</tbody>
</table>

**Common Data Preprocessing Failures**

The following are common reasons why preprocessing can fail.

- Not all records (with record IDs) in a batch that are sent to the Lambda function are returned back to the Kinesis Data Analytics service.
- The response is missing either the record ID, status, or data payload field. The data payload field is optional for a Dropped or ProcessingFailed record.
- The Lambda function timeouts are not sufficient to preprocess the data.
- The Lambda function response exceeds the response limits imposed by the AWS Lambda service.

In the case of data preprocessing failures, the Kinesis Data Analytics service continues to retry Lambda invocations on the same set of records until successful. You can monitor the following CloudWatch metrics to gain insight into failures.

- Kinesis Data Analytics Application `MillisBehindLatest`: Indicates how far behind an application is reading from the streaming source.
- Kinesis Data Analytics Application `InputPreprocessing` CloudWatch metrics: Indicates the number of successes and failures, among other statistics. For more information, see Amazon Kinesis Analytics Metrics.
- AWS Lambda function CloudWatch metrics and logs.

**For Further Information**

For further information, including architectural information and a sample Lambda function to preprocess stream records, see Pre-Processing Data in Kinesis Analytics with AWS Lambda.
Parallelizing Input Streams for Increased Throughput

Amazon Kinesis Data Analytics applications can support multiple in-application input streams, to scale an application beyond the throughput of a single in-application input stream. For more information on in-application input streams, see Amazon Kinesis Data Analytics: How It Works (p. 4).

In almost all cases, Amazon Kinesis Data Analytics scales your application to handle the capacity of the Amazon Kinesis streams or Amazon Kinesis Data Firehose source streams that feed into your application. However, if your source stream's throughput exceeds the throughput of a single in-application input stream, you can explicitly increase the number of in-application input streams that your application uses. You do so with the `InputParallelism` parameter.

When the `InputParallelism` parameter is greater than one, Amazon Kinesis Data Analytics evenly splits the partitions of your source stream among the in-application streams. For instance, if your source stream has 50 shards, and you have set `InputParallelism` to 2, each in-application input stream receives the input from 25 source stream shards.

When you increase the number of in-application streams, your application must access the data in each stream explicitly. For information on accessing multiple in-application streams in your code, see Accessing Separate In-Application Streams in Your Amazon Kinesis Data Analytics Application (p. 27).

Although Kinesis Data Analytics and Kinesis Data Firehose stream shards are both divided among in-application streams in the same way, they differ in the way they appear to your application:

- The records from a Kinesis Data Analytics stream include a `shard_id` field that can be used to identify the source shard for the record.
- The records from a Kinesis Data Firehose stream don't include a field that identifies the record's source shard or partition, because Kinesis Data Firehose abstracts this information away from your application.

Evaluating Whether to Increase Your Number of In-Application Input Streams

In most cases, a single in-application input stream can handle the throughput of a single source stream, depending on the complexity and data size of the input streams. To determine if you need to increase the number of in-application input streams, you can monitor the `MillisBehindLatest` metric in Amazon CloudWatch. If the `MillisBehindLatest` metric has either of the following characteristics, you should increase your application's `InputParallelism` setting:

- The `MillisBehindLatest` metric is gradually increasing, indicating that your application is falling behind the latest data in the stream.
- The `MillisBehindLatest` metric is consistently above 1000 (one second).

You don't need to increase your application's `InputParallelism` setting if the following are true:

- The `MillisBehindLatest` metric is gradually decreasing, indicating that your application is catching up to the latest data in the stream.
- The `MillisBehindLatest` metric is below 1000 (one second).

For more information on using CloudWatch, see the CloudWatch User Guide.
Implementing Multiple In-Application Input Streams

You can set the number of in-application input streams when an application is created using CreateApplication (p. 152). You set this number after an application is created using UpdateApplication (p. 181).

**Note**

You can only set the `InputParallelism` setting using the Amazon Kinesis Data Analytics API or the AWS CLI. You cannot set this setting using the AWS Management Console. For information on setting up the CLI, see Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 42).

Setting a New Application's Input Stream Count

The following example demonstrates how to use the CreateApplication API action to set a new application's input stream count to 2.

For more information on CreateApplication, see CreateApplication (p. 152).

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

Setting an Existing Application's Input Stream Count

The following example demonstrates how to use the UpdateApplication API action to set an existing application's input stream count to 2.

For more information on UpdateApplication, see UpdateApplication (p. 181).

```json
{
   "InputUpdates": [
      {
         "InputId": "yourInputId",
         "InputParallelism": {
            "Count": 2
         }
      },
   ],
}
```

Accessing Separate In-Application Streams in Your Amazon Kinesis Data Analytics Application

To use multiple in-application input streams in your application, you must explicitly select from the different streams. The following code example demonstrates how to query multiple input streams in the application created in the Getting Started tutorial.
In the following example, each source stream is first aggregated using \texttt{COUNT} before being combined into a single in-application stream called \texttt{in_application_stream001}. Aggregating the source streams beforehand helps make sure that the combined in-application stream can handle the traffic from multiple streams without being overloaded.

\textbf{Note}
To run this example and get results from both in-application input streams, you need to update both the number of shards in your source stream and the \texttt{InputParallelism} parameter in your application.

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

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

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

The preceding code example produces output in \texttt{in_application_stream001} similar to the following:

<table>
<thead>
<tr>
<th>ROWTIME</th>
<th>TICKER</th>
<th>TICKER_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-05-17 22:05:00.0</td>
<td>QAZ</td>
<td>15</td>
</tr>
<tr>
<td>2017-05-17 22:06:00.0</td>
<td>SAC</td>
<td>16</td>
</tr>
<tr>
<td>2017-05-17 22:06:00.0</td>
<td>PLM</td>
<td>10</td>
</tr>
<tr>
<td>2017-05-17 22:06:00.0</td>
<td>AMZN</td>
<td>15</td>
</tr>
</tbody>
</table>

\textbf{Additional Considerations}

When using multiple input streams, be aware of the following:

- The maximum number of in-application input streams is 64.
- The in-application input streams are distributed evenly among the shards of the application's input stream.
- The performance gains from adding in-application streams don't scale linearly. That is, doubling the number of in-application streams doesn't double throughput. With a typical row size, each in-application stream can achieve throughput of about 5,000 to 15,000 rows per second. By increasing the in-application stream count to 10, you can achieve a throughput of 20,000 to 30,000 rows per
second. Throughput speed is dependent on the count, data types, and data size of the fields in the input stream.

- Some aggregate functions (such as AVG) can produce unexpected results when applied to input streams partitioned into different shards. Because you need to run the aggregate operation on individual shards before combining them into an aggregate stream, the results might be weighted toward whichever stream contains more records.

- If your application continues to experience poor performance (reflected by a high MillisBehindLatest metric) after you increase your number of input streams, you might have reached your limit of Kinesis Processing Units (KPUs). For more information, see Automatically Scaling Applications to Increase Throughput (p. 40).

### 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 Data Analytics: How It Works (p. 4).

For information about the SQL language elements that are supported by Kinesis Data Analytics, see Amazon Kinesis Data Analytics SQL Reference.

In relational databases, you work with tables, using INSERT statements to add records and the SELECT statement to query the data. In Amazon Kinesis Data 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:

```sql
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 makes the identifier case sensitive. If you don't, the identifier defaults to uppercase. For more information about identifiers, see Identifiers in the Amazon Kinesis Data 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. 62).

If you add an 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 Data Analytics Application (p. 87).

According to the application's output configuration, Amazon Kinesis Data 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. 62)
• Amazon Kinesis Data 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 data stream, a Kinesis data delivery stream, or an AWS Lambda function.

There is a limit on the number of external destinations you can use to persist an application output. For more information, see Limits (p. 120).

**Note**

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

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

• **In-application stream name** – The stream that you want to persist to an external destination.

Amazon Kinesis Data 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). Make sure that your application code creates this in-application stream.

• **External destination** – You can persist data to a Kinesis data stream, a Kinesis data delivery stream, or a Lambda function. You provide the Amazon Resource Name (ARN) of the stream or function, and an IAM role that Kinesis Data Analytics can assume to write to the stream or function on your behalf. You also describe the record format (JSON, CSV) to Kinesis Data Analytics to use when writing to the external destination.

If Amazon Kinesis Data Analytics can't write to the streaming or Lambda destination, the service continues to try indefinitely. This creates back pressure, causing your application to fall behind. If this is not resolved, your application eventually stops processing new data. You can monitor Amazon Kinesis Analytics Metrics and set alarms for failures. For more information about metrics and alarms, see Using Amazon CloudWatch Metrics and Creating Amazon CloudWatch Alarms.
You can configure the application output using the AWS Management Console. The console makes the API call to save the configuration.

## Creating an Output Using the AWS CLI

This section describes how to create the `Outputs` section of the request body for a `CreateApplication` or `AddApplicationOutput` operation.

### Creating a Kinesis Stream Output

The following JSON fragment shows the `Outputs` section in the `CreateApplication` request body for creating an Amazon Kinesis data stream destination.

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

### Creating a Kinesis Data Delivery Stream Output

The following JSON fragment shows the `Outputs` section in the `CreateApplication` request body for creating an Amazon Kinesis data delivery stream destination.

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

### Creating a Lambda Function Output

The following JSON fragment shows the `Outputs` section in the `CreateApplication` request body for creating an AWS Lambda function destination.

```json
"Outputs": [  
{  
   "DestinationSchema": {  
      "RecordFormatType": "string" 
   },  
   "LambdaOutput": {  
      "ResourceARN": "string",  
      "RoleARN": "string" 
   },  
   "Name": "string" 
}  
]
```
Using a Lambda Function as Output

This topic contains the following sections:

- Lambda as Output Permissions (p. 32)
- Lambda as Output Metrics (p. 33)
- Lambda as Output Templates (p. 33)
- Lambda as Output Event Input Data Model and Record Response Model (p. 33)
- Lambda Output Invocation Frequency (p. 34)
- Adding a Lambda Function for Use as an Output (p. 35)
- Common Lambda as Output Failures (p. 35)

Using Lambda as a destination allows you to more easily perform post-processing of your SQL results before sending them to a final destination. Common post-processing tasks include the following:

- Aggregating multiple rows into a single record
- Combining current results with past results to address late-arriving data
- Delivering to different destinations based on the type of information
- Record format translation (such as translating to Protobuf)
- String manipulation or transformation
- Data enrichment after analytical processing
- Custom processing for geospatial use cases
- Data encryption

Lambda functions can deliver analytic information to a variety of AWS services and other destinations, including the following:

- Amazon Simple Storage Service (Amazon S3)
- Custom APIs
- Amazon DynamoDB
- Apache Aurora
- Amazon Redshift
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- Amazon CloudWatch

For more information about creating Lambda applications, see Getting Started with AWS Lambda.

Lambda as Output Permissions

To use Lambda as output, the application's Lambda output IAM role requires the following permissions policy:

```json
{
    "Sid": "UseLambdaFunction",
    "Effect": "Allow",
    "Action": [  
```
Using a Lambda Function as Output

Lambda as Output Metrics

You use Amazon CloudWatch to monitor the number of bytes sent, successes and failures, and so on. For information about CloudWatch metrics that are emitted by Kinesis Data Analytics using Lambda as output, see Amazon Kinesis Analytics Metrics.

Lambda as Output Templates

Kinesis Data Analytics provides templates for creating a Lambda function for use as a destination for an application. You can use these templates as a starting point for post-processing output from your application.

The following templates are available:

<table>
<thead>
<tr>
<th>Lambda as Output Blueprint</th>
<th>Language and Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kinesis-analytics-output</td>
<td>Node.js 6.10</td>
<td>Deliver output records from a Kinesis Data Analytics application to a custom destination.</td>
</tr>
<tr>
<td>kinesis-analytics-output-sns</td>
<td>Python 2.7</td>
<td>Deliver output records from a Kinesis Data Analytics application to Amazon SNS.</td>
</tr>
<tr>
<td>kinesis-analytics-output-ddb</td>
<td>Python 2.7</td>
<td>Deliver output records from a Kinesis Data Analytics application to Amazon DynamoDB.</td>
</tr>
</tbody>
</table>

Lambda as Output Event Input Data Model and Record Response Model

To send Kinesis Data Analytics output records, your Lambda function must be compliant with the required event input data and record response models.

Event Input Data Model

Kinesis Data Analytics continuously sends the output records from the application to the Lambda as output function with the following request model. Within your function, you iterate through the list and apply your business logic to accomplish your output requirements (such as data transformation before sending to a final destination).

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invocationId</td>
<td>The Lambda invocation ID (random GUID).</td>
</tr>
<tr>
<td>applicationArn</td>
<td>The Kinesis data analytics application Amazon Resource Name (ARN).</td>
</tr>
</tbody>
</table>
Field | Description
--- | ---
records

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordId</td>
<td>record ID (random GUID)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambdaDeliveryRecordMetadata</td>
<td></td>
</tr>
<tr>
<td>retryHint</td>
<td>Number of delivery retries</td>
</tr>
</tbody>
</table>

data | Base64-encoded output record payload |

**Note**
The `retryHint` is a value that increases for every delivery failure. This value is not durably persisted, and resets if the application is disrupted.

**Record Response Model**

Each record sent to your Lambda as output function (with record IDs) must be acknowledged with either `Ok` or `DeliveryFailed` and must contain the following parameters, or Kinesis Data Analytics treats them as a delivery failure.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordId</td>
<td>The record ID is passed from Kinesis Data Analytics to Lambda during the invocation. Any mismatch between the ID of the original record and the ID of the acknowledged record is treated as a delivery failure.</td>
</tr>
</tbody>
</table>

result | The status of the delivery of the record. The following are possible values:

- `Ok`: The record was transformed successfully and sent to the final destination. Kinesis Data Analytics ingests the record for SQL processing.
- `DeliveryFailed`: The record was not delivered successfully to the final destination by the Lambda as output function. Kinesis Data Analytics continuously retries sending the delivery failed records to the Lambda as output function.

**Lambda Output Invocation Frequency**

A Kinesis data analytics application buffers the output records and invokes the AWS Lambda destination function frequently.

- If records are emitted to the destination in-application stream within the Kinesis Analytics application as a tumbling window, the AWS Lambda destination function is invoked per tumbling window trigger.
For example, if a tumbling window of 60 seconds is used to emit the records to the destination in-application stream, then the AWS Lambda function is invoked once every 60 seconds.

- If records are emitted to the destination in-application stream with in the Kinesis data analytics application as a continuous query or a sliding window, the AWS Lambda destination function is invoked approximately once per second.

**Note**
Per-Lambda function invoke request payload size limits apply, and exceeding those limits will result in output records being split and sent across multiple Lambda function calls.

### Adding a Lambda Function for Use as an Output

The following procedure demonstrates how to add a Lambda function as an output for an Amazon Kinesis data analytics application.

1. Sign in to the AWS Management Console and open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose the application in the list, and then choose **Application details**.
3. In the **Destination** section, choose **Connect new destination**.
4. For the **Destination** item, choose **AWS Lambda function**.
5. In the **Deliver records to AWS Lambda** section, either choose an existing Lambda function, or choose **Create new**.
6. If you are creating a new Lambda function, do the following:
   a. Choose one of the Lambda as Output Templates (p. 33).
   b. In the **Create Function** page that opens in a new browser tab, in the **Name** field, give the function a meaningful name (for example, *myLambdaFunction*).
   c. Update the template with post-processing functionality for your application. For information about creating a Lambda function, see Getting Started in the AWS Lambda Developer Guide.
   d. In the Kinesis Data Analytics console, in the **Lambda function** drop-down list, choose the Lambda function that you just created.
7. In the **In-application stream** section, choose **Choose an existing in-application stream**. For **In-application stream name**, choose your application's output stream. The results from the selected output stream are sent to the Lambda output function.
8. Leave the rest of the form with the default values, and choose **Save and continue**.

Your application now sends records from the in-application stream to your Lambda function. You can see the results of the default template in the Amazon CloudWatch console. Monitor the **AWS/KinesisAnalytics/LambdaDelivery.OkRecords** metric to see the number of records being delivered to the Lambda function.

### Common Lambda as Output Failures

The following are common reasons why delivery to a Lambda function can fail.

- Not all records (with record IDs) in a batch that are sent to the Lambda function are returned to the Kinesis Data Analytics service.
- The response is missing either the record ID or the status field.
- The Lambda function timeouts are not sufficient to accomplish the business logic within the Lambda function.
- The business logic within the Lambda function does not catch all the errors, resulting in a timeout and backpressure due to unhandled exceptions. These are often referred as “poison pill” messages.
In the case of data delivery failures, Kinesis Data Analytics continues to retry Lambda invocations on the same set of records until successful. To gain insight into failures, you can monitor the following CloudWatch metrics:

- Kinesis Data Analytics application Lambda as Output CloudWatch metrics: Indicates the number of successes and failures, among other statistics. For more information, see Amazon Kinesis Analytics Metrics.
- AWS Lambda function CloudWatch metrics and logs.

Delivery Model for Persisting Application Output to an External Destination

Amazon Kinesis Data Analytics uses an "at least once" delivery model for application output to the configured destinations. When an application is running, Amazon Kinesis Data 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 that your application output is delivered at least once to the configured destinations.

In a normal situation, your application processes incoming data continuously, and Kinesis Data Analytics writes the output to the configured destinations such as a Kinesis stream or a Kinesis data 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 Data Analytics needs to write your application output to the configured destination. Without the IAM role, Kinesis Data 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, Kinesis Data Analytics ensures that 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 the same in-application stream, after the application recovers from failure, Kinesis Data 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 Data Analytics returns API or SQL errors directly to you. For more information about API operations, see Actions (p. 138). For more information about handling SQL errors, see Amazon Kinesis Data Analytics SQL Reference.

Amazon Kinesis Data Analytics reports runtime errors using an in-application error stream called error_stream.
Reporting Errors Using an In-Application Error Stream

Amazon Kinesis Data Analytics reports runtime errors to the in-application error stream called `error_stream`. The following are examples of errors that might occur:

- 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).
- The data in the source stream can't be converted to the data type specified in the schema (Coercion error). For information about what data types can be converted, see Mapping JSON Data Types to SQL Data Types (p. 15).

We recommend that you handle these errors programmatically in your SQL code and/or persist the data on the error stream to an external destination. This requires that you add an output configuration (see Configuring Application Output (p. 30)) to your application. For an example of how the in-application error stream works, see Example: Explore the In-Application Error Stream (p. 109).

Error Stream Schema

The error stream has the following schema:

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRORTIME</td>
<td>TIMESTAMP</td>
<td>The time when the error occurred</td>
</tr>
<tr>
<td>ERROR_LEVEL</td>
<td>VARCHAR(10)</td>
<td></td>
</tr>
<tr>
<td>ERROR_NAME</td>
<td>VARCHAR(32)</td>
<td></td>
</tr>
<tr>
<td>MESSAGE</td>
<td>VARCHAR(4096)</td>
<td></td>
</tr>
<tr>
<td>DATA_ROWTIME</td>
<td>TIMESTAMP</td>
<td>The row time of the incoming record</td>
</tr>
<tr>
<td>DATA_ROW</td>
<td>VARCHAR(49152)</td>
<td></td>
</tr>
<tr>
<td>PUMP_NAME</td>
<td>VARCHAR(128)</td>
<td>The originating pump, as defined with CREATE PUMP</td>
</tr>
</tbody>
</table>

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

Amazon Kinesis Data Analytics needs permissions to read records from a streaming source that you specify in your application input configuration. Amazon Kinesis Data 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 Data Analytics can assume. Permissions that you grant to this role determine what Amazon Kinesis Data 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 Data 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 Data Analytics permissions to assume a role, you can attach the following trust policy to an IAM role:

```json
{
  "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 Data 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 Kinesis stream or a Kinesis Data Firehose delivery stream), you can attach the following permissions policy.

### Permissions Policy for Reading an Kinesis Stream

```json
{
  "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 Kinesis Data 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 Data 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 Kinesis Stream

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "WriteOutputKinesis",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords"
      ],
      "Resource": [
      ]
    }
  ]
}
```

Permissions Policy for Writing to a Firehose Delivery Stream

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "WriteOutputFirehose",
      "Effect": "Allow",
      "Action": [
        "firehose:DescribeDeliveryStream",
        "firehose:PutRecord",
        "firehose:PutRecords"
      ],
      "Resource": [
        "arn:aws:firehose:aws-region:aws-account-id:deliverystream/outputFirehoseName"
      ]
    }
  ]
}
```
Automatically Scaling Applications to Increase Throughput

Amazon Kinesis Data Analytics elastically scales your application to accommodate the data throughput of your source stream and your query complexity for most scenarios. Kinesis Data Analytics provisions capacity in the form of Kinesis Processing Units (KPU). A single KPU provides you with the memory (4 GB) and corresponding computing and networking.

The default limit for KPUs for your application is eight. For instructions on how to request an increase to this limit, see To request a limit increase in AWS Service Limits.

**Note**

The drop-down item that is used to select a limit increase for KPUs is not yet available. When requesting an increase, choose the following options on the support form:

- **Regarding:** Service limit increase
- **Limit Type:** Kinesis Analytics
- **Region:** Select your application's Region
- **Limit:** Number of applications limit
- **New limit value:** 100
- **Use Case Description:** Provide your application prefix, and specify that you are requesting a limit increase for KPUs.
Getting Started with Amazon Kinesis Data Analytics

Following, you can find topics to help get you started using Amazon Kinesis Data Analytics. If you are new to Kinesis Data Analytics, we recommend that you review the concepts and terminology presented in Amazon Kinesis Data Analytics: How It Works (p. 4) before performing the steps in the Getting Started section.

Topics

- Step 1: Set Up an AWS Account and Create an Administrator User (p. 41)
- Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 42)
- Step 3: Create Your Starter Amazon Kinesis Data Analytics Application (p. 43)
- Step 4 (Optional) Edit the Schema and SQL Code Using the Console (p. 52)

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

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

1. Sign Up for AWS (p. 41)
2. Create an IAM User (p. 42)

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 Data Analytics. You are charged only for the services that you use.

With Kinesis Data Analytics, you pay only for the resources you use. If you are a new AWS customer, you can get started with Kinesis Data 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 https://aws.amazon.com/, and then choose Create an AWS Account.
   
   **Note**
   This might be unavailable in your browser if you previously signed into the AWS Management Console. In that case, choose Sign in to a different account, and then choose Create a new 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 Data 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:

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

Next Step

Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 42)

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: Create Your Starter Amazon Kinesis Data Analytics Application (p. 43), 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.

```plaintext
[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:

```plaintext
aws help
```

Next Step

Step 3: Create Your Starter Amazon Kinesis Data Analytics Application (p. 43)

Step 3: Create Your Starter Amazon Kinesis Data Analytics Application

By following the steps in this section, you can create your first Amazon Kinesis data analytics application using the console.

**Note**

We suggest that you review Amazon Kinesis Data Analytics: How It Works (p. 4) before trying the Getting Started exercise.

For this Getting Started exercise, you can use the console to work with either the demo stream or templates with application code.

- If you choose to use the demo stream, the console creates a Kinesis data stream in your account that is called `kinesis-analytics-demo-stream`.

A Kinesis data analytics application requires a streaming source. For this source, several SQL examples in this guide use the demo stream `kinesis-analytics-demo-stream`. The console also runs a script that continuously adds sample data (simulated stock trade records) to this stream, as shown following.
You can use `kinesis-analytics-demo-stream` as the streaming source for your application in this exercise.

**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.

- If you choose to use the templates with example application code, you use 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. 171)`. Kinesis Data Analytics uses this schema to create an in-application stream.

   When you start the application, Kinesis Data 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:

```sql
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
```
Step 3.1: Create an Application

The application code queries the in-application stream `SOURCE_SQL_STREAM_001`. The code then 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. 29).

For information about the SQL language elements that are supported by Kinesis Data Analytics, see Amazon Kinesis Data Analytics SQL Reference.

4. **Configuring output** – In this exercise, you don’t configure any output. That is, you don’t persist data in the in-application stream that your application creates to any external destination. Instead, you verify query results in the console. Additional examples in this guide show how to configure output. For one example, see Example: Simple Alerts (p. 93).

   **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 AWS Regions.

Next Step

Step 3.1: Create an Application (p. 45)

**Step 3.1: Create an Application**

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

**To create a data analytics application**

1. Sign in to the AWS Management Console and open the Kinesis Data 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**.

   Doing this creates a Kinesis data 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` 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**

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.

**To add a streaming source to your application**

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 option shows the name of the in-application input stream that is created, `SOURCE_SQL_STREAM_001`. You can change the name, but for this exercise, keep 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 Data 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.

• **Record pre-processing with AWS Lambda**: This option is where you specify an AWS Lambda expression that modifies the records in the input stream before your application code executes. In this exercise, leave the *Disabled* option selected. For more information about Lambda preprocessing, see Using the Lambda Preprocessing Feature (p. 21).

After you provide all the information on this page, the console sends an update request (see `UpdateApplication (p. 181)`) to add the input configuration the application.

3. On the **Source** page, choose **Configure a new stream**.
4. Choose **Create demo stream**. The console configures the application input by doing the following:

   • The console creates a Kinesis data stream called `kinesis-analytics-demo-stream`.
   • The console populates the stream with sample stock ticker data.
   • Using the `DiscoverInputSchema (p. 171)` input action, the console infers a schema by reading sample records on the stream. The schema that is inferred is the schema for the in-application input stream that is created. For more information, see Configuring Application Input (p. 6).
   • 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.

The following appear on the **Stream sample** console page:

• The **Raw stream sample** tab shows the raw stream records sampled by the `DiscoverInputSchema (p. 171)` API action to infer the schema.
The **Formatted stream sample** tab shows the tabular version of the data in the **Raw stream sample** tab.

- If you choose **Edit schema**, you can edit the inferred schema. For this exercise, don't change the inferred schema. For more information about editing a schema, see **Working with the Schema Editor** (p. 52).

- If you choose **Rediscover schema**, you can request the console to run **DiscoverInputSchema** (p. 171) again 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)**

You can write your own SQL queries against the in-application stream, but for the following step 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 “GSEExample1”?** dialog box, choose **Yes, start application**.

   The console sends a request to start the application (see **StartApplication** (p. 177)), 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**. The sample code reads data from one in-application stream (the WHERE clause filters the rows) and inserts it in another in-application stream as follows:

- It creates the in-application stream `DESTINATION_SQL_STREAM`.
- It creates a pump `STREAM_PUMP`, and 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 Data 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, choose **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 weren't specified in the input configuration. These include the following time stamp columns:

  • **ROWTIME** – Each row in an in-application stream has a special column called **ROWTIME**. This column is the time stamp when Amazon Kinesis Data 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 Kinesis Data Analytics record includes a value called **Approximate_Arrival_Time**. This value is the approximate arrival time stamp that is set when the streaming source successfully receives and stores the record. When Kinesis Data Analytics reads records from a streaming source, it fetches this column into the in-application input stream.

These time stamp values are useful in windowed queries that are time-based. For more information, see [Windowed Queries](#). 

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

Choose **DESTINATION_SQL_STREAM** to view the rows your application code inserted. Note the additional columns that your application code didn't create. These columns include the **ROWTIME** time stamp column. Kinesis Data Analytics simply copies these values from the source (**SOURCE_SQL_STREAM_001**).

• The **Destination** tab shows the external destination where Kinesis Data Analytics writes the query results. You haven't configured any external destination for your application output yet.

### Next Step

**Step 3.4: (Optional) Update the Application Code**

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

### To update 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:
Step 3.4: (Optional) Update the 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 the stock ticker, and then insert them into 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.

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

• Created your first Kinesis data 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. Kinesis Data 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, you can configure the application output to write records in the
DESTINATION_SQL_STREAM to an external destination. For this exercise, this is an optional step. To learn how to configure the destination, go to the next step.

Next Step

Step 4 (Optional) Edit the Schema and SQL Code Using the Console (p. 52).

Step 4 (Optional) Edit the Schema and SQL Code Using the Console

Following, you can find information about how to edit an inferred schema and how to edit SQL code for Amazon Kinesis Data Analytics. You do so by working with the schema editor and SQL editor that are part of the Kinesis Data Analytics console.

Topics

• Working with the Schema Editor (p. 52)
• Working with the SQL Editor (p. 59)

Working with the Schema Editor

The schema for an Amazon Kinesis data analytics application's input stream defines how data from the stream is made available to SQL queries in the application.

The schema contains selection criteria for determining what part of the streaming input is transformed into a data column in the in-application input stream. This input can be one of the following:

• A JSONPath expression for JSON input streams. JSONPath is a tool for querying JSON data.
• A column number for input streams in comma-separated values (CSV) format.
• A column name and a SQL data type for presenting the data in the in-application data stream. The data type also contains a length for character or binary data.

The console attempts to generate the schema using DiscoverInputSchema (p. 171). If schema discovery fails or returns an incorrect or incomplete schema, you must edit the schema manually by using the schema editor.
Schema Editor Main Screen

The following screenshot shows the main screen for the Schema Editor.

You can apply the following edits to the schema:

- Add a column (1): You might need to add a data column if a data item is not detected automatically.
- Delete a column (2): You can exclude data from the source stream if your application doesn't require it. This exclusion doesn't affect the data in the source stream. If data is excluded, that data simply isn't made available to the application.
• Rename a column (3). A column name can’t be blank, must be longer than a single character, and
must not contain reserved SQL keywords. The name must also meet naming criteria for SQL ordinary
identifiers: The name must start with a letter and contain only letters, underscore characters, and
digits.

• Change the data type (4) or length (5) of a column: You can specify a compatible data type for a
column. If you specify an incompatible data type, the column is either populated with NULL or the in-
application stream is not populated at all. In the latter case, errors are written to the error stream. If
you specify a length for a column that is too small, the incoming data is truncated.

• Change the selection criteria of a column (6): You can edit the JSONPath expression or CSV column
order used to determine the source of the data in a column. To change the selection criteria for a
JSON schema, type a new value for the row path expression. A CSV schema uses the column order as
selection criteria. To change the selection criteria for a CSV schema, change the order of the columns.

Editing the Schema for a Streaming Source

If you need to edit a schema for a streaming source, follow these steps.

To edit the schema for a streaming source


2. On the Edit schema page, edit the source schema.
3. For **Format**, choose **JSON** or **CSV**. For JSON or CSV format, the supported encoding is ISO 8859-1.

For further information on editing the schema for JSON or CSV format, see the procedures in the next sections.

**Editing a JSON Schema**

You can edit a JSON schema by using the following steps.

**To edit a JSON schema**

1. In the schema editor, choose **Add column** to add a column.

   A new column appears in the first column position. To change column order, choose the up and down arrows next to the column name.

   For a new column, provide the following information:

   - For **Column name**, type a name.

     A column name cannot be blank, must be longer than a single character, and must not contain reserved SQL keywords. It must also meet naming criteria for SQL ordinary identifiers: It must start with a letter and contain only letters, underscore characters, and digits.

   - For **Column type**, type an SQL data type.

     A column type can be any supported SQL data type. If the new data type is CHAR, VARBINARY, or VARCHAR, specify a data length for **Length**. For more information, see [Data Types](#).
• For **Row path**, provide a row path. A row path is a valid JSONPath expression that maps to a JSON element.

  **Note**
  The base **Row path** value is the path to the top-level parent that contains the data to be imported. This value is $ by default. For more information, see `RecordRowPath` in `JSONMappingParameters`.

2. To delete a column, choose the x icon next to the column number.

3. To rename a column, type a new name for **Column name**. The new column name cannot be blank, must be longer than a single character, and must not contain reserved SQL keywords. It must also meet naming criteria for SQL ordinary identifiers: It must start with a letter and contain only letters, underscore characters, and digits.

4. To change the data type of a column, choose a new data type for **Column type**. If the new data type is CHAR, VARBINARY, or VARCHAR, specify a data length for **Length**. For more information, see `Data Types`.

5. Choose **Save schema and update stream** to save your changes.

The modified schema appears in the editor and looks similar to the following.
If your schema has many rows, you can filter the rows using **Filter by column name**. For example, to edit column names that start with `P`, such as a `Price` column, type `P` in the **Filter by column name** box.

**Editing a CSV Schema**

You can edit a CSV schema by using the following steps.

**To edit a CSV schema**

1. In the schema editor, for **Row delimiter**, choose the delimiter used by your incoming data stream. This is the delimiter between records of data in your stream, such as a newline character.
2. For **Column delimiter**, choose the delimiter used by your incoming data stream. This is the delimiter between fields of data in your stream, such as a comma.
3. To add a column, choose **Add column**.

   A new column appears in the first column position. To change column order, choose the up and down arrows next to the column name.

   For a new column, provide the following information:

   - For **Column name**, type a name.
     
     A column name cannot be blank, must be longer than a single character, and must not contain reserved SQL keywords. It must also meet naming criteria for SQL ordinary identifiers: It must start with a letter and contain only letters, underscore characters, and digits.
   
   - For **Column type**, type a SQL data type.
     
     A column type can be any supported SQL data type. If the new data type is `CHAR`, `VARBINARY`, or `VARCHAR`, specify a data length for **Length**. For more information, see [Data Types](#).
4. To delete a column, choose the $x$ icon next to the column number.

5. To rename a column, type a new name in Column name. The new column name cannot be blank, must be longer than a single character, and must not contain reserved SQL keywords. It must also meet naming criteria for SQL ordinary identifiers: It must start with a letter and contain only letters, underscore characters, and digits.

6. To change the data type of a column, choose a new data type for Column type. If the new data type is CHAR, VARBINARY, or VARCHAR, specify a data length for Length. For more information, see Data Types.

7. Choose Save schema and update stream to save your changes.

The modified schema appears in the editor and looks similar to the following.
If your schema has many rows, you can filter the rows using **Filter by column name**. For example, to edit column names that start with `P`, such as a `Price` column, type `P` in the **Filter by column name** box.

## Working with the SQL Editor

Following, you can find information about sections of the SQL editor and how each works. In the SQL editor, you can either author your own code yourself or choose **Add SQL from templates**. A SQL template gives you example SQL code that can help you write common Amazon Kinesis data analytics applications. The example applications in this guide use some of these templates. For more information, see **Example Amazon Kinesis Data Analytics Applications** (p. 73).

Kinesis Analytics dashboard > DemoApplication > SQL editor

### Source Data Tab

The **Source data** tab identifies a streaming source. It also identifies the in-application input stream that this source maps to and that provides the application input configuration.
Amazon Kinesis Data Analytics provides the following time stamp columns, so that 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. This column is the time stamp for the point when Kinesis Data Analytics inserted the row in the first in-application stream.

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

These time stamp values are useful in windowed queries that are time-based. For more information, see Windowed Queries (p. 66).

**Real-Time Analytics Tab**

The Real-time analytics tab shows all the in-application streams that your application code creates. This group of streams includes the error stream (error_stream) that Amazon Kinesis Data Analytics provides for all applications.
Destination Tab

The 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. 30).
Streaming SQL Concepts

Amazon Kinesis Data 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 (p. 62)
- Timestamps and the ROWTIME Column (p. 63)
- Continuous Queries (p. 65)
- Windowed Queries (p. 66)
- Streaming Data Operations: Stream Joins (p. 71)

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 Kinesis Data Firehose delivery streams. In-application streams exist only in the context of an Amazon Kinesis Data Analytics application. Amazon Kinesis streams and Kinesis Data 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 Data 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
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. 66).

You can also join streams. For examples of joining streams, see Streaming Data Operations: Stream Joins (p. 71).

---

### Timestamps and the ROWTIME Column

In-application streams include a special column called **ROWTIME**. It stores a timestamp when Amazon Kinesis Data Analytics inserts a row in the first in-application stream. **ROWTIME** reflects the timestamp at which Amazon Kinesis Data 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 Data Analytics copies this column for you.

Amazon Kinesis Data 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. 66).

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

```sql
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 Data Streams includes a field called `ApproximateArrivalTime` 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 Data Analytics inserts a row in the first in-application stream. Amazon Kinesis Data 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.

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

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

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 down to the nearest 60 second interval. 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).

<table>
<thead>
<tr>
<th>ROWTIME</th>
<th>INGEST_TIME</th>
<th>TICKER_SYMBOL</th>
<th>SYMBOL_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-07-19 17:05:00.0</td>
<td>2016-07-19 17:05:00.0</td>
<td>ABC</td>
<td>10</td>
</tr>
<tr>
<td>2016-07-19 17:05:00.0</td>
<td>2016-07-19 17:05:00.0</td>
<td>DEF</td>
<td>15</td>
</tr>
<tr>
<td>2016-07-19 17:05:00.0</td>
<td>2016-07-19 17:05:00.0</td>
<td>XYZ</td>
<td>6</td>
</tr>
</tbody>
</table>

--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 Kinesis Data 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 XYZ, 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 Kinesis stream). Following is the schema:

```sql
(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.

```sql
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 Data 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:

```sql
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
```
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 Data 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. 63).

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) (p. 66)
- Sliding Windows (p. 67)

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. This stream 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:

```sql
-- 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;
```
The preceding is an example of a windowed query that is time-based. The query groups records by `ROWTIME` values. For reporting on a per-minute basis, the `STEP` function rounds down the `ROWTIME` values to the nearest minute.

**Note**
You can also use the `FLOOR` function to group records into windows, but `FLOOR` can only round time values down to a whole time unit (hour, minute, second, and so on). `STEP` is recommended for grouping records into tumbling windows because it can round values down to an arbitrary interval, e.g. 30 seconds.

This query is an example of a nonoverlapping (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 from the input data stream. In this example, reports are generated each minute.

**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:

```sql
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (  
ticker_symbol VARCHAR(4),  
Min_Price     DOUBLE,  
Max_Price     DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS  
INSERT INTO "DESTINATION_SQL_STREAM"  
SELECT STREAM Ticker_Symbol,  
       MIN(Price) AS Min_Price,  
       MAX(Price) AS Max_Price  
FROM    "SOURCE_SQL_STREAM_001"  
GROUP BY Ticker_Symbol,  
        STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

### Sliding Windows

Instead of grouping records using `GROUP BY`, you can define a time-based or row-based window. You do this by adding an explicit `WINDOW` clause.

In this case, as the window slides with time, Amazon Kinesis Data Analytics emits an output when new records appear on the stream. Kinesis Data Analytics emits this output by processing rows in the window. Windows can overlap in this type of processing, and a record can be part of multiple windows and be processed with each window. The following example illustrates a sliding window.

Consider a simple query that counts records on the stream. We assume a 5-second window. In the following example stream, new records arrive at time \( t_1 \), \( t_2 \), \( t_6 \), and \( t_7 \), and three records arrive at time \( t_8 \) seconds.
Keep the following in mind:

- We assume a 5-second window. The 5-second window slides continuously with time.
- For every row that enters a window, an output row is emitted by the sliding window. Soon after the application starts, you see the query emit output for every new record that appears on the stream, even though a 5-second window hasn’t passed 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 5-second window.
- The windows slide with time. If an old record on the stream falls out of the window, the query doesn’t emit output unless there is also a new record on the stream that falls within that 5-second window.

Suppose the query starts executing at \( t_0 \). If so, the following occurs:

1. At the time \( t_0 \), the query starts. The query doesn’t 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 emits count value 1.

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

4. The 5-second window slides with time:
   - At \( t_3 \), the sliding window \( t_3 \) to \( t_0 \)
   - At \( t_4 \) (sliding window \( t_4 \) to \( t_0 \))
   - At \( t_5 \) the sliding window \( t_5 \)–\( t_0 \)

   At all of these times, the 5-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 5-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 doesn’t count.
6. At time $t_7$, the 5-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 5-second window, and therefore isn’t counted.

7. At time $t_8$, the 5-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.

**Note**
We recommend that you use a sliding window no longer than one hour. If you use a longer window, the application takes longer to restart after regular system maintenance, because the source data needs to be read from the stream again.

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 1-minute Sliding Window**

Consider the demo stream in the Getting Started exercise that populates the in-application stream, `SOURCE_SQL_STREAM_001`. The following is the schema.

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

Suppose that you want your application to compute aggregates using a sliding 1-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 1-minute window.

You can use the following time-based windowed query. The query uses the `WINDOW` clause to define the 1-minute range interval. The `PARTITION BY` in the `WINDOW` clause groups records by ticker values within the sliding window.

```sql
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 the following.

```sql
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 on the demo stream returns the average of the percent change in the price of each ticker in a 10-second window.

```sql
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 the following.

```sql
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 the following example, the query emits the output ticker, price, a2, and a10. It emits output for ticker symbols whose two-row moving average crosses the ten-row moving average. 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. 69).

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

- **TradeStream** – Receives resulting stock trades for those orders.
  
  ```sql
  (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 1-minute window over the TradeStream. For information about windowed queries, see Sliding Windows (p. 67).

```
SELECT STREAM
    ROWTIME,
```
Example 1: Report Orders Where There Are Trades Within One Minute of the Order Being Placed

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

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

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

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 1-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.

```sql
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 PRECEDING) AS t
ON o.orderId = t.orderId;
```
Example Amazon Kinesis Data Analytics Applications

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

Before you explore these walkthroughs, we recommend that you first review Amazon Kinesis Data Analytics: How It Works (p. 4) and Getting Started with Amazon Kinesis Data Analytics (p. 41).

Topics

- Examples: Preprocessing Streams (p. 73)
- Examples: Basic Analytics (p. 91)
- Examples: Advanced Analytics (p. 96)
- Examples: Post Processing In-Application Stream (p. 105)
- Examples: Other Amazon Kinesis Data Analytics Applications (p. 108)

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 (p. 73)
- Example: Streaming Source With Multiple Record Types (p. 82)
- Example: Adding Reference Data to an Amazon Kinesis Data Analytics Application (p. 87)

Example: Manipulating Strings and Date Times

String Manipulation

Amazon Kinesis Data Analytics supports formats such as JSON and CSV for records on a streaming source. For details, see RecordFormat (p. 237). These records then map to rows in in-application stream as per the input configuration. For details, see Configuring Application Input (p. 6). 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, which 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 contains 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.

```json
{
    "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_Unstructured field value is a series of comma-separated values.

```json
{ "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.

```json
{ "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 Data 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 Data Analytics SQL Reference.

• **SQL functions** – The following functions are useful when manipulating individual strings. For more information, see Scalar Functions in the Amazon Kinesis Data 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 functions, see the following topics:
• Example: String Manipulation (W3C_LOG_PARSE Function) (p. 75)
• Example: String Manipulation (VARIABLE_COLUMN_LOG_PARSE Function) (p. 77)
• Example: String Manipulation (SUBSTRING Function) (p. 80)

Example: String Manipulation (W3C_LOGPARSE Function)

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

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

You then create an Amazon Kinesis data analytics application in the console, with the Kinesis data 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 \texttt{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 a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis Data Analytics console at \url{https://console.aws.amazon.com/kinesisanalytics}.
2. Choose \textbf{Data Streams} 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.

```python
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:
```
Step 2: Create the Amazon Kinesis Data Analytics Application

Create an Amazon Kinesis Data Analytics application as follows:

1. Sign in to the AWS Management Console and open the Kinesis Data 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 the 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 the application code and verify the results as follows:
   - Copy the following application code and paste it into the editor.

```sql
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 data stream. The example records are as follows:
You then create an Amazon Kinesis Data Analytics application in the console, with the 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:

![Stream sample]

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 a Kinesis Data Stream]

### Step 1: Create a Kinesis Data Stream

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

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

```python
import json
from boto import kinesis
import random
```
Step 2: Create the Amazon Kinesis Data Analytics Application

Create an Amazon Kinesis Data Analytics application as follows:

1. Sign in to the AWS Management Console and open the Kinesis Data 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 the 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 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 Data Analytics application in the console, with the 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 the URL string to retrieve the company name. Then insert the resulting data into another in-application stream, as shown following:

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis Data 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.

```python
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 Data Analytics Application

Create an Amazon Kinesis Data Analytics application as follows:

1. Sign in to the AWS Management Console and open the Kinesis Data 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 the 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 the application code and verify the results as follows:
   • Copy the following application code and paste it into the editor.

```sql
-- 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 Data Analytics supports converting columns to time stamps. For example, you might want to use your own time stamp as part of a GROUP BY clause as another time-based window, in addition to the ROWTIME column. Kinesis Data 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 Data Analytics SQL Reference*.

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

• **SQL Extensions** – These include the following:
  - `CURRENT_ROW_TIMESTAMP` – Returns a new time stamp for each row in the stream.
  - `TSDIFF` – Returns the difference of two time stamps 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 time stamp.
  - `DATE_TO_CHAR` – Converts a date to a string.
  - `TIME_TO_CHAR` – Converts a time to a string.
  - `TIMESTAMP_TO_CHAR` – Converts a time stamp 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 time stamp. For more information, Char To Timestamp(Sys) in the *Amazon Kinesis Data Analytics SQL Reference*.

**Example: Streaming Source With Multiple Record Types**

**Topics**
- **Step 1: Prepare** (p. 85)
- **Step 2: Create an Application** (p. 86)

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 Data 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 Kinesis data 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**

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

**Trade record**

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

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

When you save the configuration, Amazon Kinesis Data 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, in the application code you first create two additional in-application streams, `Order_Stream`, `Trade_Stream`. You then filter the rows from the `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. 29).

- 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`.

```sql
--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)
);
```
Example: Streaming Source With Multiple Record Types

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 the number of trades by the ticker in a one-minute tumbling window and save the 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 OUTPUT_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. 85)

Step 1: Prepare

In this section, you create a Kinesis data 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 a Kinesis data 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 Kinesis console at https://console.aws.amazon.com/kinesis. Choose Data Streams, and then create a stream with one shard.
- Using the AWS CLI – Use the following Kinesis create-stream AWS CLI command to create the stream:

```bash
$ 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 a 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.

```python
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.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. 86)

Step 2: Create an Application

In this section, you create an Amazon Kinesis Data 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 Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose Create new application. This example uses the application name 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 the 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, choose Yes in the dialog box that appears.
6. In the SQL editor, write the application code and verify the results:
   a. Copy the following application code and paste it into the editor.

```
--Create Order_Stream.
CREATE OR REPLACE STREAM "Order_Stream"
```
Example: Add Reference Data Source

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

Next Step

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

Example: Adding Reference Data to an Amazon Kinesis Data Analytics Application

Topics

- Step 1: Prepare (p. 88)
- Step 2: Add Reference Data Source to the Application Configuration (p. 89)
- Step 3: Test: Query the In-Application Reference Table (p. 91)
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,

<table>
<thead>
<tr>
<th>Ticker, Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMZN, Amazon</td>
</tr>
<tr>
<td>ASD, SomeCompanyA</td>
</tr>
<tr>
<td>MMB, SomeCompanyB</td>
</tr>
<tr>
<td>WAS, SomeCompanyC</td>
</tr>
</tbody>
</table>

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 Data Analytics can assume to read the S3 object on your behalf.

2. Add the reference data source to your application. Amazon Kinesis Data 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 Data 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 Data 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 Data 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 Data 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 Data Analytics permissions to assume the role. The trust policy is shown following:

   ```json
   {
   "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:

   ```json
   {
   "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 Data 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://kinesisanalytics.aws-region.amazonaws.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":"
   "RecordColumnDelimiter":","
   },
   },
   },
   "RecordEncoding":"UTF-8",
   "RecordColumns":[
   {
   "Name":"Ticker",
   "SqlType":"VARCHAR(64)"
   },
   {
   "Name":"Company",
   "SqlType":"VARCHAR(64)"
   }
   ]
   }
   ' 
   ```
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.

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

2. 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 Data Analytics applications that perform basic analytics. The examples provide step-by-step instructions to set up an Amazon Kinesis Data Analytics application.

Topics
- Example: Most Frequently Occurring Values (the TOP_K_ITEMS_TUMBLING Function) (p. 91)
- Example: Counting Distinct Values (the COUNT_DISTINCT_ITEMS_TUMBLING function) (p. 92)
- Example: Simple Alerts (p. 93)
- Example: Throttled Alerts (p. 95)

Example: Most Frequently Occurring Values (the TOP_K_ITEMS_TUMBLING Function)

In this exercise, you set up an Amazon Kinesis Data 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.

Use the following application code:

```sql
CREATE OR REPLACE STREAM DESTINATION_SQL_STREAM (ITEM VARCHAR(1024),
```
Example: Count Distinct Values

In this exercise, you set up a Amazon Kinesis Data 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.

Use the following application code:

```sql
CREATE OR REPLACE STREAM DESTINATION_SQL_STREAM (  
    NUMBER_OF_DISTINCT_ITEMS BIGINT);
```
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. 65). If any rows show a stock price change that is greater than 1 percent, 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 Data Analytics application as described in the Getting Started Exercise.
2. In the SQL editor, replace the application code with the following:

```sql
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. 29).

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

      d. Return to the Amazon Kinesis Data 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 Data 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 Kinesis stream, where Amazon Kinesis Data Analytics persists your application output in the DESTINATION_SQL_STREAM in-application stream.

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

   For instructions, see Example: Integrating Amazon Kinesis Data Analytics with AWS Lambda (p. 105).
Example: Throttled 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. 65). If any rows show stock price change is greater than 1%, those rows are inserted in another in-application stream. The application throttles the alerts such that an alert is sent immediately when the stock price changes, but no more than one alert per minute per stock symbol is sent to the in-application stream.

To create a throttled alerts application

1. Create the Amazon Kinesis Data 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 "CHANGE_STREAM"
  (ticker_symbol VARCHAR(4),
   sector VARCHAR(12),
   change DOUBLE,
   price DOUBLE);

CREATE OR REPLACE PUMP "change_pump" AS
  INSERT INTO "CHANGE_STREAM"
  SELECT STREAM ticker_symbol, sector, change, price
  FROM "SOURCE_SQL_STREAM_001"
  WHERE (ABS(Change / (Price - Change)) * 100) > 1;

-- ** Trigger Count and Limit **
-- Counts "triggers" or those values that evaluated true against the previous where clause
-- Then provides its own limit on the number of triggers per hour per ticker symbol to what
-- is specified in the WHERE clause

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

CREATE OR REPLACE PUMP trigger_count_pump AS
  INSERT INTO TRIGGER_COUNT_STREAM
  SELECT STREAM ticker_symbol, change, trigger_count
  FROM (SELECT STREAM ticker_symbol, change, COUNT(*) OVER W1 as trigger_count
          FROM "CHANGE_STREAM"
          WINDOW W1 AS (PARTITION BY ticker_symbol RANGE INTERVAL '1' MINUTE PRECEDING)
       )
  WHERE trigger_count >= 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 CHANGE_STREAM using a pump.

The application then creates a second stream called TRIGGER_COUNT_STREAM for the throttled alerts. A second query selects records from a window that hops forward every time a record is admitted into it, such that only one record per stock ticker per minute is written to the stream.

3. Click Save and run SQL.

The preceding example outputs a stream to TRIGGER_COUNT_STREAM similar to the following:
Examples: Advanced Analytics

This section provides additional examples of Amazon Kinesis Data 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) (p. 96)
- Example: Detecting Data Anomalies and Getting an Explanation (RANDOM_CUT_FOREST_WITH_EXPLANATION Function) (p. 101)
- Example: Using Different Types of Times in Streaming Analytics (p. 105)

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

Amazon Kinesis Data 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 Data 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 Kinesis stream and write sample heartRate data as shown following:

   ```json
   {"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 Data 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 Data Analytics continuously reads the streaming source and inserts records into the in-application stream.

3. **Specify application code** – Use the following application code:

```sql
-- Creates a temporary stream.
CREATE OR REPLACE STREAM "TEMP_STREAM" (  
    "heartRate" INTEGER,
    "rateType" varchar(20),
    "ANOMALY_SCORE" DOUBLE);

-- Creates another stream for application output.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (  
    "heartRate" INTEGER,
    "rateType" varchar(20),
    "ANOMALY_SCORE" DOUBLE);

-- Compute an anomaly score for each record in the input stream
-- using Random Cut Forest
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "TEMP_STREAM"
    SELECT STREAM "heartRate", "rateType", ANOMALY_SCORE
    FROM TABLE(RANDOM_CUT_FOREST(
        CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001")));

-- Sort records by descending anomaly score, insert into output stream
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM * FROM "TEMP_STREAM"
    ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), ANOMALY_SCORE DESC;
```

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. 62).

4. **Configure output** – You configure the application output to persist data in the DESTINATION_SQL_STREAM to an external destination, which is another 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. 97)

**Step 1: Prepare**

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

**Step 1.1: Create Two Kinesis Streams**

In this section, you create two Kinesis streams (ExampleInputStream and ExampleOutputStream).
Amazon Kinesis Data Analytics Developer Guide
Example: Detect Anomalies

1.

You can create these streams using the console or the AWS CLI.
• Sign in to the AWS Management Console and open the Kinesis Data Analytics console at https://
console.aws.amazon.com/kinesisanalytics.
• Choose Kinesis Stream, and then create a stream with one shard.
• Use the following 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())

98


Step 2: Create an Application

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

- Configure the application input to use the 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. 45)) 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 Data 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):

```sql
-- 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
  SELECT STREAM "heartRate", "rateType", ANOMALY_SCORE
  FROM TABLE(RANDOM_CUT_FOREST("TEMP_STREAM")
```
CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001");

-- Sort records by descending anomaly score, insert into output stream
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM * FROM "TEMP_STREAM"
    ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), ANOMALY_SCORE DESC;

3. Run the SQL code and review results:

Next Step

Step 3: Configure Application Output (p. 100)

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 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 Data 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 Data Analytics with AWS Lambda (p. 105). If not, you can review the records that Amazon Kinesis Data Analytics writes to the external destination, the Kinesis stream OutputStreamTestingAnomalyScores, as described in the next step.

Next Step

Step 4: Verify Output (p. 101)
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.

   ```bash
   aws kinesis get-shard-iterator \
   --shard-id shardId-000000000000 \n   --shard-iterator-type TRIM_HORIZON \n   --stream-name OutputStreamTestingAnomalyScores \n   --region us-east-1 \n   --profile adminuser
   
   You get a response with a shard iterator value, as shown in the following example response:
   
   ```json
   {   "ShardIterator":   "shard-iterator-value"  }
   ```
   
   Copy the shard iterator value.

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

   ```bash
   aws kinesis get-records \
   --shard-iterator shard-iterator-value \n   --region us-east-1 \n   --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: Detecting Data Anomalies and Getting an Explanation
(RANDOM_CUT_FOREST_WITH_EXPLANATION Function)

Amazon Kinesis Data Analytics provides the `RANDOM_CUT_FOREST_WITH_EXPLANATION` function, which assigns an anomaly score to each record based on values in the numeric columns. The function also provides an explanation of the anomaly. For more information, see `RANDOM_CUT_FOREST_WITH_EXPLANATION`.

In this exercise, you write application code to obtain anomaly scores for records in your application's streaming source. You also obtain an explanation for each anomaly.

First Step

Step 1: Prepare the Data (p. 101)

Step 1: Prepare the Data

Before you create an Amazon Kinesis Data Analytics application for this example (p. 101), you create a Kinesis data stream to use as the streaming source for your application. You also run Python code to write simulated blood pressure data to the stream.
Step 1.1: Create a Kinesis Data Stream

In this section, you create a Kinesis data stream named `ExampleInputStream`. You can create this data stream using the AWS Management Console or the AWS CLI.

- To use the console:
  1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
  2. Go to the Data Streams dashboard, and choose Create Kinesis stream.
  3. For the name, type `ExampleInputStream`. For the number of shards, type 1.

- Alternatively, to use the AWS CLI to create the data stream, run the following command:

```
$ aws kinesis create-stream --stream-name ExampleInputStream --shard-count 1
```

Step 1.2: Write Sample Records to the Input Stream

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

1. Install Python and pip.

   For information about installing Python, see Python.

   You can install dependencies using pip. For information about installing pip, see Installation in the pip documentation.

2. Run the following Python code. You can change the Region to the one you want to use for this example. 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 blood pressure with a 0.995 probability
def getNormalBloodPressure():
    data = {
        'Systolic': random.randint(90, 120),
        'Diastolic': random.randint(60, 80),
        'BloodPressureLevel': 'NORMAL',
    }
    return data

# generate high blood pressure with probability 0.005
def getHighBloodPressure():
    data = {
        'Systolic': random.randint(130, 200),
        'Diastolic': random.randint(90, 150),
        'BloodPressureLevel': 'HIGH',
    }
    return data

# generate low blood pressure with probability 0.005
def getLowBloodPressure():
    data = {
        'Systolic': random.randint(50, 80),
        'Diastolic': random.randint(30, 50),
    }
```
data['BloodPressureLevel'] = 'LOW'
return data

while True:
    rnd = random.random()
    if (rnd < 0.005):
        data = json.dumps(getLowBloodPressure())
        print(data)
        kinesis.put_record("BloodPressureExampleInputStream", data, "partitionkey")
    elif (rnd > 0.995):
        data = json.dumps(getHighBloodPressure())
        print(data)
        kinesis.put_record("BloodPressureExampleInputStream", data, "partitionkey")
    else:
        data = json.dumps(getNormalBloodPressure())
        print(data)
        kinesis.put_record("BloodPressureExampleInputStream", data, "partitionkey")

The previous code writes to ExampleInputStream records similar to the following examples:

{"Systolic": 109, "Diastolic": 64, "BloodPressureLevel": "NORMAL"}
{"Systolic": 99, "Diastolic": 72, "BloodPressureLevel": "NORMAL"}
{"Systolic": 159, "Diastolic": 100, "BloodPressureLevel": "HIGH"}
{"Systolic": 94, "Diastolic": 75, "BloodPressureLevel": "NORMAL"}
{"Systolic": 91, "Diastolic": 78, "BloodPressureLevel": "NORMAL"}
{"Systolic": 91, "Diastolic": 74, "BloodPressureLevel": "NORMAL"}
{"Systolic": 102, "Diastolic": 75, "BloodPressureLevel": "NORMAL"}
{"Systolic": 50, "Diastolic": 31, "BloodPressureLevel": "LOW"}
{"Systolic": 100, "Diastolic": 66, "BloodPressureLevel": "NORMAL"}
{"Systolic": 115, "Diastolic": 65, "BloodPressureLevel": "NORMAL"}
{"Systolic": 99, "Diastolic": 74, "BloodPressureLevel": "NORMAL"}

Next Step

Step 2: Create an Analytics Application (p. 103)

Step 2: Create an Analytics Application

In this section, you create an Amazon Kinesis data analytics application and configure it to use the Kinesis data stream you created in the preceding section (p. 101) as the streaming source. You then run application code that uses the RANDOM_CUT_FOREST_WITH_EXPLANATION function.

To create an application

2. Go to the Amazon Kinesis Data Analytics dashboard, and choose Create application.
3. Provide an application name and description (optional), and choose Create application.
4. Choose Connect to a source, and then choose ExampleInputStream from the list.
5. Choose Discover schema, and make sure that Systolic and Diastolic appear as INTEGER columns. If they have another type, choose Edit schema, and assign the type INTEGER to both of them.
6. Under Real time analytics, choose Go to SQL editor. When prompted, choose to run your application.
7. Paste the following code into the SQL editor, and then choose Save and run SQL.

```sql
--Creates a temporary stream.
```
Example: Detect Anomalies and Obtain an Explanation

```
CREATE OR REPLACE STREAM "TEMP_STREAM" (  
  "Systolic"          INTEGER,  
  "Diastolic"         INTEGER,  
  "BloodPressureLevel" varchar(20),  
  "ANOMALY_SCORE"     DOUBLE,  
  "ANOMALY_EXPLANATION"  varchar(512));

-- Creates another stream for application output.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (  
  "Systolic"          INTEGER,  
  "Diastolic"         INTEGER,  
  "BloodPressureLevel" varchar(20),  
  "ANOMALY_SCORE"     DOUBLE,  
  "ANOMALY_EXPLANATION"  varchar(512));

-- Compute an anomaly score with explanation for each record in the input stream
-- using RANDOM_CUT_FOREST_WITH_EXPLANATION
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "TEMP_STREAM"
  SELECT STREAM "Systolic", "Diastolic", "BloodPressureLevel", "ANOMALY_SCORE",
          "ANOMALY_EXPLANATION"
  FROM TABLE(RANDOM_CUT_FOREST_WITH_EXPLANATION(
    CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"), 100, 256, 100000,
    1, true));

-- Sort records by descending anomaly score, insert into output stream
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
  SELECT STREAM * FROM "TEMP_STREAM"
  ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), "ANOMALY_SCORE" DESC;
```

Next Step

Step 3: Examine the Results (p. 104)

Step 3: Examine the Results

When you run the SQL code for this example (p. 101), you first see rows with an anomaly score equal to zero. This happens during the initial learning phase. Then you get results similar to the following.

```
27:49.0 101 66 NORMAL 0.711460417 {"Systolic":
  {"DIRECTION":"LOW","STRENGTH":"0.0922","ATRIBUTION_SCORE":"0.3792"},
  "Diastolic":
  {"DIRECTION":"HIGH","STRENGTH":"0.0210","ATRIBUTION_SCORE":"0.3323"}}
27:50.0 144 123 HIGH 3.855851061 {"Systolic":
  {"DIRECTION":"HIGH","STRENGTH":"0.8567","ATRIBUTION_SCORE":"1.7447"},
  "Diastolic":
  {"DIRECTION":"HIGH","STRENGTH":"7.0982","ATRIBUTION_SCORE":"2.1111"}}
27:50.0 113 69 NORMAL 0.740069409 {"Systolic":
  {"DIRECTION":"LOW","STRENGTH":"0.0549","ATRIBUTION_SCORE":"0.3750"},
  "Diastolic":
  {"DIRECTION":"LOW","STRENGTH":"0.0394","ATRIBUTION_SCORE":"0.3650"}}
27:50.0 105 64 NORMAL 0.739644157 {"Systolic":
  {"DIRECTION":"HIGH","STRENGTH":"0.0245","ATRIBUTION_SCORE":"0.3667"},
  "Diastolic":
  {"DIRECTION":"LOW","STRENGTH":"0.0524","ATRIBUTION_SCORE":"0.3729"}}
27:50.0 100 65 NORMAL 0.736993425 {"Systolic":
  {"DIRECTION":"HIGH","STRENGTH":"0.0203","ATRIBUTION_SCORE":"0.3516"},
  "Diastolic":
  {"DIRECTION":"LOW","STRENGTH":"0.0454","ATRIBUTION_SCORE":"0.3854"}}
27:50.0 108 69 NORMAL 0.733767202 {"Systolic":
  {"DIRECTION":"LOW","STRENGTH":"0.0974","ATRIBUTION_SCORE":"0.3961"},
  "Diastolic":
  {"DIRECTION":"LOW","STRENGTH":"0.0189","ATRIBUTION_SCORE":"0.3377"}}
```

- The algorithm in the RANDOM_CUT_FOREST_WITH_EXPLANATION function sees that the Systolic and Diastolic columns are numeric, and uses them as input.
The BloodPressureLevel column has text data, and is therefore not taken into account by the algorithm. This column is simply a visual aide to help you quickly spot the normal, high, and low blood pressure levels in this example.

In the ANOMALY_SCORE column, records with higher scores are more anomalous. The second record in this sample set of results is the most anomalous, with an anomaly score of 3.855851061.

To understand the extent to which each of the numeric columns taken into account by the algorithm contributes to the anomaly score, consult the JSON field named ATtribution_Score in the ANOMALY_SCORE column. In the case of the second row in this set of sample results, the Systolic and Diastolic columns contribute to the anomaly in the ratio of 1.7447:2.1111. In other words, 45 percent of the explanation for the anomaly score is attributable to the systolic value, and the remaining attribution is due to the diastolic value.

To determine the direction in which the point represented by the second row in this sample is anomalous, consult the JSON field named DIRECTION. Both the diastolic and systolic values are marked as HIGH in this case. To determine the confidence with which these directions are correct, consult the JSON field named STRENGTH. In this example, the algorithm is more confident that the diastolic value is high. Indeed, the normal value for the diastolic reading is usually 60–80, and 123 is much higher than expected.

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. 63). 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 Data 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 a Kinesis data stream or a Kinesis data delivery stream, for further analysis.

For example, if application output is persisted to a Kinesis data 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 Data Analytics with AWS Lambda (p. 105)

Example: Integrating Amazon Kinesis Data Analytics with AWS Lambda

Integrating Amazon Kinesis Data Analytics applications with AWS Lambda enable additional scenarios. If you persist your application output an 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 Data Analytics application in the following sections persist output to an Kinesis stream:
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 Data Analytics stream.

You configure AWS Lambda as follows:

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

2. 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.

3. Add event source mapping in AWS Lambda to associate the Lambda function with your 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 Kinesis chapter also provides an AWS Lambda and Amazon Kinesis Data 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 Data Analytics Application (p. 106)
- Step 2: Create an Amazon SNS Topic (p. 107)
- Step 3: Create a Lambda Function (p. 107)
- Step 4: Verify Results (p. 108)

**Step 1: Create an Amazon Kinesis Data Analytics Application**

In this section you set up an Amazon Kinesis Data 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. 96).

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.

   ```sql
   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](https://docs.aws.amazon.com/sns/latest/dg/gs-sns.html) 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](https://docs.aws.amazon.com/lambda/latest/dg/start.html) in the *AWS Lambda Developer Guide*.

In the AWS Lambda console at [https://console.aws.amazon.com/lambda/](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 **Step 2: Configure event sources** page, specify the following values:
   
   - **Event source type** – Kinesis
   - **Kinesis stream** – Select the Kinesis stream from the that is the configured destination for your Amazon Kinesis Data 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:

   ```python
   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 = base64.b64decode(record['kinesis']['data'])

           response = snsClient.publish(
               TopicArn='SNS-topic-ARN',
               Message='Anomaly detected ... ' + payload,
               Subject='Anomaly detected',
               MessageStructure='string',
               MessageAttributes={
                   'String': {
                   
```
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 Data Analytics is writing records from the in-application output stream to the output destination (an 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 Data Analytics Applications

This section provides examples that help you explore Amazon Kinesis Data 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 Data 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. 45).
2. On the newly created application hub, choose Connect to a source.
   If you followed the Getting Started exercise, you have a demo stream in your account.
4. Amazon Kinesis Data Analytics takes a sample from the demo stream to infer a schema for the in-application input stream it creates. The console shows the inferred schema and sample data in the Formatted stream sample tab.
5. Now you edit the schema and modify the 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 Data Analytics can't bring in data in the in-application stream. Instead Kinesis Data Analytics sends the rows to the 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 Data 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 the application code to introduce a runtime error (division by zero). Notice that Amazon Kinesis Data 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: Create Your Starter Amazon Kinesis Data Analytics Application (p. 43).

   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:

```sql
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 the results to the `DESTINATION_SQL_STREAM`, Amazon Kinesis Data 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 Data Analytics

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Kinesis Data Analytics and your Amazon Kinesis Data Analytics application. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multipoint failure if one occurs. Before you start monitoring Amazon Kinesis Data 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 Data Analytics performance in your environment, by measuring performance at various times and under different load conditions. As you monitor Amazon Kinesis Data Analytics, you can store historical monitoring data. If you do, you can compare it with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

With Amazon Kinesis Data 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 Data Analytics processes data streams, see Amazon Kinesis Data Analytics: How It Works (p. 4).

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 milliseconds 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 Data Analytics supports, see Amazon Kinesis Data Analytics Metrics.

Topics
- Monitoring Tools (p. 111)
- Monitoring with Amazon CloudWatch (p. 112)

Monitoring Tools

AWS provides various tools that you can use to monitor Amazon Kinesis Data 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 Data 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 Amazon EC2 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. 112).

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

- The CloudWatch home page shows the following:
  - 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 Data Analytics applications using CloudWatch, which collects and processes raw data from Amazon Kinesis Data 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 Data Analytics metric data is automatically sent to CloudWatch. For more information, see What Are
Viewing Amazon Kinesis Data Analytics Metrics and Dimensions

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

In the console, metrics are grouped first by service namespace, and then by the dimension combinations within each namespace.

For a list of metrics Amazon Kinesis Data Analytics supports, see Amazon Kinesis Data Analytics Metrics.

To view metrics using the CloudWatch console

2. In the navigation pane, choose Metrics.
3. In the CloudWatch Metrics by Category pane for Amazon Kinesis Data Analytics, select a metrics category.
4. In the upper pane, scroll 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 Data Analytics metrics are collected at the following levels:

- Application
- Input stream
- Output stream

Creating CloudWatch Alarms to Monitor Amazon Kinesis Data 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. It 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. For a CloudWatch alarm to invoke an action, the state must have changed and been maintained for a specified amount of time.

You can set alarms using the AWS Management Console, CloudWatch CLI, or CloudWatch API, as described following.
To set an alarm 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. The Create Alarm Wizard launches.
3. Choose Kinesis Analytics Metrics, and then scroll through the Amazon Kinesis Data Analytics metrics to locate the metric you want to place an alarm on.

To display just Amazon Kinesis Data Analytics metrics, search for the file system ID of your file system. Select the metric to create an alarm for, and then choose Next.
4. Type values for Name, Description, and Whenever 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. In the Alarm Preview section, preview the alarm you're about to create.
7. Choose Create Alarm to create the alarm.

To set an alarm using the CloudWatch CLI

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

To set an alarm using the CloudWatch API

- Call PutMetricAlarm. For more information, see the Amazon CloudWatch API Reference.

Working with Amazon CloudWatch Logs

If an Amazon Kinesis Data Analytics application is misconfigured, it can transition to a running state during application start or update but not process any data into the in-application input stream. By adding a CloudWatch log option to the application, you can monitor for application configuration problems.

Amazon Kinesis Data Analytics can generate configuration errors under the following conditions:

- The Kinesis Stream used for input doesn't exist.
- The Amazon Kinesis Data Firehose delivery stream used for input doesn't exist.
- The Amazon S3 bucket used as a reference data source doesn't exist.
- The specified file in the reference data source in the S3 bucket doesn't exist.
- The correct resource is not defined in the AWS Identity and Access Management (IAM) role that manages related permissions.
- The correct permission is not defined in the IAM role that manages related permissions.
- Kinesis Data Analytics doesn't have permission to assume the IAM role that manages related permissions.

For more information on Amazon CloudWatch, see the CloudWatch User Guide.
Adding the PutLogEvents Policy Action

Amazon Kinesis Data Analytics needs permissions to write misconfiguration errors to CloudWatch. You can add these permissions to the IAM role that Amazon Kinesis Data Analytics assumes, as described following. For more information on using an IAM role for Amazon Kinesis Data Analytics, see Granting Amazon Kinesis Data Analytics Permissions to Access Streaming Sources (Creating an IAM Role) (p. 37).

Trust Policy

To grant Kinesis Data Analytics permissions to assume an IAM role, you can attach the following trust policy to the role.

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

Permissions Policy

To grant an application permissions to write log events to CloudWatch from a Kinesis Data Analytics resource, you can use the following IAM permissions policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt0123456789000",
      "Effect": "Allow",
      "Action": ["logs:PutLogEvents"],
      "Resource": [
      ]
    }
  ]
}
```

Adding Configuration Error Monitoring

Use the following API actions to add a CloudWatch log option to a new or existing application or change a log option for an existing application.

**Note**

You can currently only add a CloudWatch log option to an application by using API actions. You can't add CloudWatch log options by using the console.

Adding a CloudWatch Log Option When Creating an Application

The following code example demonstrates how to use the `CreateApplication` action to add a CloudWatch log option when you create an application. For more information on `Create_Application`, see CreateApplication (p. 152).
Adding a CloudWatch Log Option to an Existing Application

The following code example demonstrates how to use the AddApplicationCloudWatchLoggingOption action to add a CloudWatch log option to an existing application. For more information on AddApplicationCloudWatchLoggingOption, see AddApplicationCloudWatchLoggingOption (p. 139).

```json
{
   "ApplicationName": "<Name of the application to add the log option to>",
   "CloudWatchLoggingOption": {
      "LogStreamARN": "<ARN of the log stream to add to the application>",
      "RoleARN": "<ARN of the role to use to access the log>",
   },
   "CurrentApplicationVersionId": "<Version of the application to add the log to>
}
```

Updating an Existing CloudWatch Log Option

The following code example demonstrates how to use the UpdateApplication action to modify an existing CloudWatch log option. For more information on UpdateApplication, see UpdateApplication (p. 181).

```json
{
   "ApplicationName": "<Name of the application to update the log option for>",
   "ApplicationUpdate": {
      "CloudWatchLoggingOptionUpdates": [
      {
         "CloudWatchLoggingOptionId": "<ID of the logging option to modify>",
         "LogStreamARNUpdate": "<ARN of the new log stream to use>",
         "RoleARNUpdate": "<ARN of the new role to use to access the log stream>",
      }
      ],
   },
   "CurrentApplicationVersionId": "<ID of the application version to modify>
}
```

Deleting a CloudWatch Log Option from an Application

The following code example demonstrates how to use the DeleteApplicationCloudWatchLoggingOption action to delete an existing CloudWatch log option. For more information on DeleteApplicationCloudWatchLoggingOption, see DeleteApplicationCloudWatchLoggingOption (p. 159).

```json
{
}
```
Configuration Errors

Following, you can learn details about errors that you might see in CloudWatch logs from a misconfigured application.

Error Message Format

Error messages generated by application misconfiguration appear in the following format.

```json
{
  "applicationARN": "string",
  "applicationVersionId": integer,
  "messageType": "ERROR",
  "message": "string",
  "inputId": "string",
  "referenceId": "string",
  "errorCode": "string",
  "messageSchemaVersion": "integer"
}
```

The fields in an error message contain the following information:

- **applicationARN**: The Amazon Resource Name (ARN) of the generating application, for example: `arn:aws:kinesisanalytics:us-east-1:112233445566:application/sampleApp`
- **applicationVersionId**: The version of the application at the time the error was encountered. For more information, see ApplicationDetail (p. 186).
- **messageType**: The message type. Currently, this type can be only `ERROR`.
- **message**: The details of the error, for example:

```text
There is a problem related to the configuration of your input. Please check that the resource exists, the role has the correct permissions to access the resource and that Kinesis Analytics can assume the role provided.
```

- **inputId**: ID associated with the application input. This value is only present if this input is the cause of the error. This value is not present if `referenceId` is present. For more information, see DescribeApplication (p. 167).
- **referenceId**: ID associated with the application reference data source. This value is only present if this source is the cause of the error. This value is not present if `inputId` is present. For more information, see DescribeApplication (p. 167).
- **errorCode**: The identifier for the error. This ID is either InputError or ReferenceDataError.
- **messageSchemaVersion**: A value that specifies the current message schema version, currently 1. You can check this value to see if the error message schema has been updated.

Errors

The errors that might appear in CloudWatch logs for Amazon Kinesis Data Analytics include the following.
**Resource Does Not Exist**

If an ARN is specified for a Kinesis input stream that doesn't exist, but the ARN is syntactically correct, an error like the following is generated.

```json
{
  "applicationVersionId": "5",
  "messageType": "ERROR",
  "message": "There is a problem related to the configuration of your input. Please check that the resource exists, the role has the correct permissions to access the resource and that Kinesis Analytics can assume the role provided.",
  "inputId": "1.1",
  "errorCode": "InputError",
  "messageSchemaVersion": "1"
}
```

If an incorrect Amazon S3 file key is used for reference data, an error like the following is generated.

```json
{
  "applicationVersionId": "5",
  "messageType": "ERROR",
  "message": "There is a problem related to the configuration of your reference data. Please check that the bucket and the file exist, the role has the correct permissions to access these resources and that Kinesis Analytics can assume the role provided.",
  "referenceId": "1.1",
  "errorCode": "ReferenceDataError",
  "messageSchemaVersion": "1"
}
```

**Role Does Not Exist**

If an ARN is specified for an IAM input role that doesn't exist, but the ARN is syntactically correct, an error like the following is generated.

```json
{
  "applicationVersionId": "5",
  "messageType": "ERROR",
  "message": "There is a problem related to the configuration of your input. Please check that the resource exists, the role has the correct permissions to access the resource and that Kinesis Analytics can assume the role provided.",
  "inputId": null,
  "errorCode": "InputError",
  "messageSchemaVersion": "1"
}
```

**Role Does Not Have Permissions to Access the Resource**

If an input role is used that doesn't have permission to access the input resources, such as an Kinesis source stream, an error like the following is generated.

```json
{
  "applicationVersionId": "5",
  "messageType": "ERROR",
  "message": "There is a problem related to the configuration of your input. Please check that the resource exists, the role has the correct permissions to access the resource and that Kinesis Analytics can assume the role provided.",
}
```
"inputId":null,
"errorCode": "InputError",
"messageSchemaVersion": "1"
}
Limits

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

• The 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 Data Analytics in the AWS General Reference.

• You can create up to 50 Kinesis Data Analytics applications per AWS Region in your account. You can create a case to request additional applications via the service limit increase form. For more information, see the AWS Support Center.

• The maximum amount of source parallelism is 64. That is, in your application input configuration, you can request the mapping of a streaming source to up to 64 in-application streams.

• The number of Kinesis processing units (KPU) is limited to eight. For instructions on how to request an increase to this limit, see To request a limit increase in AWS Service Limits.

With Kinesis Data Analytics, you pay only for what you use. You are charged an hourly rate based on the average number of KPUs that are 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.

• You can configure up to three destinations for your Kinesis Data Analytics application. We recommend that you use one of these destinations to persist in-application error stream data.

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

• If you change the reference data that is stored in the S3 bucket after you upload reference data to an in-application table, you need to use the UpdateApplication (p. 181) operation (using the API or AWS CLI) to refresh the data in the in-application table. Currently, the AWS Management Console doesn’t support refreshing reference data in your application.

• Currently, Kinesis Data Analytics doesn't support data generated by the Amazon Kinesis Producer Library (KPL).
Best Practices

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

Topics
- Managing Applications (p. 121)
- Defining Input Schema (p. 122)
- Connecting to Outputs (p. 123)
- Authoring Application Code (p. 123)

Managing Applications

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

- **Set up CloudWatch alarms** – Using the CloudWatch metrics that Amazon Kinesis Data 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 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 Data Analytics application can read from a Kinesis Data Firehose delivery stream. However, many applications can read from an Kinesis stream, such as an Amazon Kinesis Data Analytics application or AWS Lambda. The recommended application limit refers to all applications that you configure to read from a streaming source.

Amazon Kinesis Data 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 Kinesis Data Firehose delivery stream to one application.**
A Kinesis Data 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 Data Analytics application. Therefore, we recommend you do not configure more than one Amazon Kinesis Data Analytics application per Kinesis Data 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. 171)) 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 Data 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 Data 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 Data 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. 73). 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 Data 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 Data 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 Data Analytics can emit output for that window.
  • Amazon Kinesis Data Analytics must maintain everything related to the application’s state, including relevant data, for the duration. This will consume significant Amazon Kinesis Data 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. 63).
Troubleshooting Amazon Kinesis Data Analytics

The following can help you troubleshoot problems you have with Amazon Kinesis Data Analytics.

Get a SQL Statement to Work Correctly

If you need to figure out how to get a particular SQL statement to work correctly, you have several different resources when using Amazon Kinesis Data Analytics:

• For more information about SQL statements, see Example Amazon Kinesis Data Analytics Applications (p. 73) in the Amazon Kinesis Data Analytics Developer Guide. This section provides a number of SQL examples that you can use.
• The Amazon Kinesis Data Analytics SQL Reference provides a detailed guide to authoring streaming SQL statements.
• If you are still running into issues, we recommend that you ask a question on the Kinesis Data Analytics Forums.

Unable to Detect or Discover My Schema

In some cases, Kinesis Data Analytics is unable to detect or discover a schema. In many of these cases, you can still use Kinesis Data Analytics.

Suppose that you have UTF-8 encoded data that doesn’t use a delimiter, data that uses a format other than comma-separated value (CSV) format, or the discovery API did not discover your schema. In these cases, you can define a schema by hand or use string manipulation functions to structure your data.

To discover the schema for your stream, Kinesis Data Analytics randomly samples the latest data in your stream. If you aren’t consistently sending data to your stream, Kinesis Data Analytics might not be able to retrieve a sample and detect a schema. For more information, see Using the Schema Discovery Feature on Streaming Data (p. 17) Using the Schema Discovery Feature and Related Editing in the Amazon Kinesis Data Analytics Developer Guide.

Important Application Health Parameters to Monitor

To make sure that your application is running correctly, we recommend that you monitor certain important parameters.

The most important parameter to monitor is the Amazon CloudWatch metric MillisBehindLatest. This metric represents how far behind the current time you are reading from the stream. This metric helps you determine whether you are processing records from the source stream fast enough.

As a rule of thumb, you should set up a CloudWatch alarm to trigger if you fall behind more than one hour. However, the amount of time depends on your use case. You can adjust it as needed.
Invalid Code Errors When Running an Application

When you cannot save and run the SQL code for your Amazon Kinesis Data Analytics application, the following are common causes:

- **The stream was redefined in your SQL code** – After you create a stream and the pump associated with the stream, you cannot redefine the same stream in your code. For more information about creating a stream, see `CREATE STREAM`. For more information about creating a pump, see `CREATE PUMP`.

- **A GROUP BY clause uses multiple ROWTIME columns** – You can specify only one ROWTIME column in the GROUP BY clause. For more information, see `GROUP BY` and `ROWTIME`.

- **One or more data types have an invalid casting** – In this case, your code has an invalid implicit cast. For example, you might be casting a timestamp to a bigint in your code.

- **A stream has the same name as a service reserved stream name** – A stream cannot have the same name as the service-reserved stream `error_stream`.

Application Doesn't Process Data After Deleting and Re-creating the Kinesis Application Input Stream or Kinesis Data Firehose Delivery Stream with the Same Name

Suppose that you delete the Kinesis stream that provides application input for a running application and create a new Kinesis stream with the same name. In this case, the application doesn't process the input data from the new stream. In addition, no data is delivered to the destination.

The same effect occurs if you delete the Kinesis Data Firehose delivery stream for a running application and create a new Kinesis Data Firehose delivery stream with the same name.

To resolve this issue, stop and restart the application through the AWS Management Console.

Insufficient Throughput or High Latency

If your application's `MillisBehindLatest` metric is steadily increasing or consistently is above 1000 (one second), you can improve your application's throughput. To do so, increase the value of the `InputParallelism` parameter. For more information, see `Parallelizing Input Streams for Increased Throughput (p. 26)`.

If your application is reaching the default limit for Kinesis Processing Units (KPU), you can request a limit increase. For more information, see `Automatically Scaling Applications to Increase Throughput (p. 40)`.
Authentication and Access Control for Amazon Kinesis Data Analytics

Access to Amazon Kinesis Data Analytics requires credentials. Those credentials must have permissions to access AWS resources, such as an Amazon Kinesis Data 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 Data Analytics to help secure access to your resources.

• Authentication (p. 126)
• Access Control (p. 127)

Authentication

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

• **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

• **IAM user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create an application in Amazon Kinesis Data 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 AWS tools, you must sign the request yourself. Amazon Kinesis Data 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 an IAM identity that 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 existing 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
Access Control

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

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

- Overview of Managing Access Permissions to Your Amazon Kinesis Data Analytics Resources (p. 127)
- Using Identity-Based Policies (IAM Policies) for Amazon Kinesis Data Analytics (p. 131)
- Amazon Kinesis Data Analytics API Permissions: Actions, Permissions, and Resources Reference (p. 136)

Overview of Managing Access Permissions to Your Amazon Kinesis Data 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 Data Analytics Resources and Operations (p. 128)
- Understanding Resource Ownership (p. 128)
- Managing Access to Resources (p. 128)
Amazon Kinesis Data Analytics Resources and Operations

In Amazon Kinesis Data 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.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>ARN Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>arn:aws:kinesisanalytics:region:account-id:application/application-name</td>
</tr>
</tbody>
</table>

Amazon Kinesis Data Analytics provides a set of operations to work with Amazon Kinesis Data Analytics resources. For a list of available operations, see Amazon Kinesis Data Analytics Actions (p. 138).

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 Data 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 Data 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 Data Analytics supports only identity-based policies (IAM policies).
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 Data 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 Data Analytics application.

**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 Data Analytics, see Using Identity-Based Policies (IAM Policies) for Amazon Kinesis Data Analytics (p. 131). 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 Data Analytics doesn’t support resource-based policies.

Specifying Policy Elements: Actions, Effects, and Principals

For each Amazon Kinesis Data Analytics resource, the service defines a set of API operations. To grant permissions for these API operations, Amazon Kinesis Data 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 Data Analytics Resources and Operations (p. 128) and Amazon Kinesis Data Analytics Actions (p. 138).

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 Data Analytics Resources and Operations (p. 128).
- **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 Data 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 Data Analytics API operations and the resources that they apply to, see Amazon Kinesis Data Analytics API Permissions: Actions, Permissions, and Resources Reference (p. 136).

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 Data 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 Data 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 Data 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 Data Analytics resources. For more information, see Overview of Managing Access Permissions to Your Amazon Kinesis Data Analytics Resources (p. 127).

Topics
- Permissions Required to Use the Amazon Kinesis Data Analytics Console (p. 131)
- AWS Managed (Predefined) Policies for Amazon Kinesis Data Analytics (p. 132)
- Customer Managed Policy Examples (p. 133)

The following shows an example of a permissions policy.

```json
{
    "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 Data 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 that the permission is granted for any resource.

For a table showing all of the Amazon Kinesis Data Analytics API operations and the resources that they apply to, see Amazon Kinesis Data Analytics API Permissions: Actions, Permissions, and Resources Reference (p. 136).

Permissions Required to Use the Amazon Kinesis Data Analytics Console

For a user to work with Amazon Kinesis Data 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 Data Analytics (p. 132).
- 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. 133).

AWS Managed (Predefined) Policies for Amazon Kinesis Data 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 Data Analytics:

- **AmazonKinesisAnalyticsReadOnly** – Grants permissions for Amazon Kinesis Data Analytics actions that enable a user to list Amazon Kinesis Data Analytics applications and review input/output configuration. It also grants permissions that allow a user to view a list of Kinesis streams and Kinesis Data Firehose delivery streams. As 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 Data Analytics actions and all other permissions that allows a user to create and manage Amazon Kinesis Data 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 Data 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 Data 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 Data 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 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 Data Analytics application, see Creating IAM Roles in the IAM User Guide.

Example Steps

- Step 1: Create an IAM User (p. 133)
- Step 2: Allow the User Permissions for Actions that Are Not Specific to Amazon Kinesis Data Analytics (p. 133)
- Step 3: Allow the User to View a List of Applications and View Details (p. 134)
- Step 4: Allow the User to Start a Specific Application (p. 135)
- Step 5: Allow the User to Create an Amazon Kinesis Data Analytics Application (p. 135)
- Step 6: Allow the Application to Use Lambda Preprocessing (p. 136)

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 Data Analytics

First, grant a user permission for all actions that aren’t specific to Amazon Kinesis Data Analytics that the user will need when working with Amazon Kinesis Data Analytics applications. These include permissions for working with streams (Amazon Kinesis Data Streams actions, Amazon Kinesis Data 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.

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

{
"Effect": "Allow",
"Action": [
"kinesis:CreateStream",
"kinesis:DeleteStream",
"kinesis:PutRecord",
"kinesis:DescribeStream",
"cloudwatch:GetMetricData",
"cloudwatch:PutMetricData"
]
}

]}
```
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.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": ["kinesisanalytics:ListApplications", "kinesisanalytics:DescribeApplication"],
         "Resource": "*"
      }
   ]
}
```
Verify these permissions by signing into the Amazon Kinesis Data 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 Data Analytics applications, you attach the following policy to the user, which provides the permission for \texttt{kinesisanalytics:StartApplication} action. You will need to update the policy by providing your account ID, AWS Region, and application name.

\begin{verbatim}
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [\n        "kinesisanalytics:StartApplication"
      ],
    }
  ]
}
\end{verbatim}

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

Now suppose you want the user to create an Amazon Kinesis Data Analytics application. You can then attach the following policy to the user. You will need to update the policy and provide an AWS 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).

\begin{verbatim}
{
  "Version": "2012-10-17",
  "Statement": [
    {\n      "Sid": "Stmt1473028104000",
      "Effect": "Allow",
      "Action": [\n        "kinesisanalytics:CreateApplication"
      ],
      "Resource": [\n        "*"
      ]
    }
  ]
}
\end{verbatim}
Step 6: Allow the Application to Use Lambda Preprocessing

If you want the application to be able to use Lambda preprocessing, you can attach the following policy to the role. For more information on Lambda preprocessing, see Using the Lambda Preprocessing Feature (p. 21).

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

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

When you are setting up Access Control (p. 127) 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 Data 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 Data 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 Data Analytics API and Required Permissions for Actions

**API Operation:**

**Required Permissions (API Action):**

**Resources:**
Amazon Kinesis Data Analytics API and Required Permissions for Actions

Amazon RDS API and Required Permissions for Actions

API Operation: AddApplicationInput (p. 141)

Action: kinesisanalytics:AddApplicationInput

Resources:

arn:aws:kinesisanalytics: region:accountId:application/application-name
You can use the AWS CLI to explore the Amazon Kinesis Data Analytics API. This guide provides Getting Started with Amazon Kinesis Data Analytics (p. 41) exercises that use the AWS CLI.

Topics

- Actions (p. 138)
- Data Types (p. 184)

Actions

The following actions are supported:

- AddApplicationCloudWatchLoggingOption (p. 139)
- AddApplicationInput (p. 141)
- AddApplicationInputProcessingConfiguration (p. 144)
- AddApplicationOutput (p. 146)
- AddApplicationReferenceDataSource (p. 149)
- CreateApplication (p. 152)
- DeleteApplication (p. 157)
- DeleteApplicationCloudWatchLoggingOption (p. 159)
- DeleteApplicationInputProcessingConfiguration (p. 161)
- DeleteApplicationOutput (p. 163)
- DeleteApplicationReferenceDataSource (p. 165)
- DescribeApplication (p. 167)
- DiscoverInputSchema (p. 171)
- ListApplications (p. 175)
- StartApplication (p. 177)
- StopApplication (p. 179)
- UpdateApplication (p. 181)
AddApplicationCloudWatchLoggingOption

Add a CloudWatch log stream to monitor application configuration errors. For more information about using CloudWatch log streams with Amazon Kinesis Analytics applications, see Working with Amazon CloudWatch Logs.

Request Syntax

```json
{
  "ApplicationName": "string",
  "CloudWatchLoggingOption": {
    "LogStreamARN": "string",
    "RoleARN": "string"
  },
  "CurrentApplicationVersionId": number
}
```

Request Parameters

The request accepts the following data in JSON format.

**ApplicationName (p. 139)**

The Kinesis Analytics application name.

Type: String


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

Required: Yes

**CloudWatchLoggingOption (p. 139)**

Provides the CloudWatch log stream Amazon Resource Name (ARN) and the IAM role ARN. Note: To write application messages to CloudWatch, the IAM role that is used must have the `PutLogEvents` policy action enabled.

Type: `CloudWatchLoggingOption (p. 191)` object

Required: Yes

**CurrentApplicationVersionId (p. 139)**

The version ID of the Kinesis Analytics application.

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

**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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
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. 152).

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. 167) 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
    },
    "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
        "ResourceARN": "string",
        "RoleARN": "string"
      }
    },
    "InputSchema": {
      "RecordColumns": [
        {
          "Mapping": "string",
          "Name": "string",
          "SqlType": "string"
        }
      ],
      "RecordEncoding": "string",
      "RecordFormat": {
        "MappingParameters": {
          "CSVMappingParameters": {
            "RecordColumnDelimiter": "string",
            "RecordRowDelimiter": "string"
          },
          "JSONMappingParameters": {
            "RecordRowPath": "string"
          }
        },
        "RecordFormatType": "string"
      }
    },
    "KinesisFirehoseInput": {
      "ResourceARN": "string",
      "RoleARN": "string"
    },
    "KinesisStreamsInput": {
      "ResourceARN": "string",
      "RoleARN": "string"
    },
    "NamePrefix": "string"
  }
}
```
Request Parameters

The request accepts the following data in JSON format.

**ApplicationName (p. 141)**

Name of your existing Amazon Kinesis Analytics application to which you want to add the streaming source.

Type: String


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

Required: Yes

**CurrentApplicationVersionId (p. 141)**

Current version of your Amazon Kinesis Analytics application. You can use the DescribeApplication (p. 167) operation to find the current application version.

Type: Long

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

Required: Yes

**Input (p. 141)**

The Input (p. 196) to add.

Type: Input (p. 196) object

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

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
AddApplicationInputProcessingConfiguration

Adds an InputProcessingConfiguration (p. 206) to an application. An input processor preprocesses records on the input stream before the application's SQL code executes. Currently, the only input processor available is AWS Lambda.

Request Syntax

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "InputId": "string",
   "InputProcessingConfiguration": {
       "InputLambdaProcessor": {
           "ResourceARN": "string",
           "RoleARN": "string"
       }
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

**ApplicationName (p. 144)**

Name of the application to which you want to add the input processing configuration.

Type: String


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

Required: Yes

**CurrentApplicationVersionId (p. 144)**

Version of the application to which you want to add the input processing configuration. You can use the DescribeApplication (p. 167) 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

**InputId (p. 144)**

The ID of the input configuration to add the input processing configuration to. You can get a list of the input IDs for an application using the DescribeApplication (p. 167) operation.

Type: String


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

Required: Yes
AddApplicationInputProcessingConfiguration

**InputProcessingConfiguration (p. 144)**

The InputProcessingConfiguration (p. 206) to add to the application.

Type: InputProcessingConfiguration (p. 206) 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

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
AddApplicationOutput

Add 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, an Amazon Kinesis Firehose delivery stream, or an AWS Lambda function), 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 more information, see Understanding Application Output (Destination).

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. 167) 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

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

Request Parameters

The request accepts the following data in JSON format.

**ApplicationName (p. 146)**

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

Type: String

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

**CurrentApplicationVersionId (p. 146)**

Version of the application to which you want to add the output configuration. You can use the `DescribeApplication (p. 167)` 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. 146)**

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, an Amazon Kinesis Firehose delivery stream, or an AWS Lambda function), and record the formation to use when writing to the destination.

Type: `Output (p. 230)` 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

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
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

```json
{
  "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. 149)**

Name of an existing application.

Type: String

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

Required: Yes

**CurrentApplicationVersionId (p. 149)**

Version of the application for which you are adding the reference data source. You can use the `DescribeApplication (p. 167)` 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. 149)**

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. 238) 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
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
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 three 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 three 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

```json
{
  "ApplicationCode": "string",
  "ApplicationDescription": "string",
  "ApplicationName": "string",
  "CloudWatchLoggingOptions": [
    {
      "LogStreamARN": "string",
      "RoleARN": "string"
    }
  ],
  "Inputs": [
    {
      "InputParallelism": {
        "Count": number
      },
      "InputProcessingConfiguration": {
        "InputLambdaProcessor": {
          "ResourceARN": "string",
          "RoleARN": "string"
        }
      },
      "InputSchema": {
        "RecordColumns": [
          {
            "Mapping": "string",
            "Name": "string",
            "SqlType": "string"
          }
        ],
        "RecordEncoding": "string",
        "RecordFormat": {
          "MappingParameters": {
            "CSVMappingParameters": {
              "RecordColumnDelimiter": "string",
              "RecordRowDelimiter": "string"
            }
          }
        }
      }
    }
  ]
}
```
Request Parameters

The request accepts the following data in JSON format.

**ApplicationCode (p. 152)**

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 information 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. 152)**

Summary description of the application.

Type: String

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

Required: No

**ApplicationName (p. 152)**

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

Type: String


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

Required: Yes

**CloudWatchLoggingOptions (p. 152)**

Use this parameter to configure a CloudWatch log stream to monitor application configuration errors. For more information, see Working with Amazon CloudWatch Logs.

Type: Array of CloudWatchLoggingOption (p. 191) objects

Required: No

**Inputs (p. 152)**

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. 196) objects

Required: No

**Outputs (p. 152)**

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

These destinations can be Amazon Kinesis streams, Amazon Kinesis Firehose delivery streams, AWS Lambda destinations, or any combination of the three.

In the configuration, you specify the in-application stream name, the destination stream or Lambda function 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 or Lambda function on your behalf.

In the output configuration, you also provide the output stream or Lambda function ARN. For stream destinations, you provide 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 the stream or Lambda function on your behalf.

Type: Array of Output (p. 230) objects

Required: No

**Response Syntax**

```json
{
   "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. 155)**

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. 189) object

**Errors**

**CodeValidationError**

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
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. 157)**

Name of the Amazon Kinesis Analytics application to delete.

- Type: String
- Pattern: [a-zA-Z0-9_.-]+
- Required: Yes

**CreateTimestamp (p. 157)**

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

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteApplicationCloudWatchLoggingOption

Deletes a CloudWatch log stream from an application. For more information about using CloudWatch log streams with Amazon Kinesis Analytics applications, see Working with Amazon CloudWatch Logs.

Request Syntax

```json
{
    "ApplicationName": "string",
    "CloudWatchLoggingOptionId": "string",
    "CurrentApplicationVersionId": number
}
```

Request Parameters

The request accepts the following data in JSON format.

**ApplicationName (p. 159)**

The Kinesis Analytics application name.

Type: String


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

Required: Yes

**CloudWatchLoggingOptionId (p. 159)**

The CloudWatchLoggingOptionId of the CloudWatch logging option to delete. You can get the CloudWatchLoggingOptionId by using the DescribeApplication (p. 167) operation.

Type: String


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

Required: Yes

**CurrentApplicationVersionId (p. 159)**

The version ID of the Kinesis Analytics application.

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

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteApplicationInputProcessingConfiguration

Deletes an InputProcessingConfiguration (p. 206) from an input.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

**ApplicationName (p. 161)**

The Kinesis Analytics application name.

Type: String


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

Required: Yes

**CurrentApplicationVersionId (p. 161)**

The version ID of the Kinesis Analytics application.

Type: Long

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

Required: Yes

**InputId (p. 161)**

The ID of the input configuration from which to delete the input processing configuration. You can get a list of the input IDs for an application by using the DescribeApplication (p. 167) operation.

Type: String


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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**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. 163)**

Amazon Kinesis Analytics application name.

Type: String


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

Required: Yes

**CurrentApplicationVersionId (p. 163)**

Amazon Kinesis Analytics application version. You can use the `DescribeApplication (p. 167)` 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. 163)**

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. 146)` 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. 167)` operation to get the specific `OutputId`.

Type: String


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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
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. 149) 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. 165)**

Name of an existing application.

Type: String


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

Required: Yes

**CurrentApplicationVersionId (p. 165)**

Version of the application. You can use the DescribeApplication (p. 167) 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. 165)**

ID of the reference data source. When you add a reference data source to your application using the AddApplicationReferenceDataSource (p. 149), Amazon Kinesis Analytics assigns an ID. You can use the DescribeApplication (p. 167) operation to get the reference ID.

Type: String


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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
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. 175) 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. 167)**

Name of the application.

Type: String


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

Required: Yes

Response Syntax

```
{
  "ApplicationDetail": {
    "ApplicationARN": "string",
    "ApplicationCode": "string",
    "ApplicationDescription": "string",
    "ApplicationName": "string",
    "ApplicationStatus": "string",
    "ApplicationVersionId": number,
    "CloudWatchLoggingOptionDescriptions": [
      {
        "CloudWatchLoggingOptionId": "string",
        "LogStreamARN": "string",
        "RoleARN": "string"
      }
    ],
    "CreateTimestamp": number,
    "InputDescriptions": [
      {
        "InAppStreamNames": [ "string" ],
        "InputId": "string",
        "InputParallelism": {
          "Count": number
        }
      }
    ]
  }
}
```
"InputProcessingConfigurationDescription": {  "InputLambdaProcessorDescription": {  "ResourceARN": "string",  "RoleARN": "string"  }},  "InputSchema": {  "RecordColumns": [  {  "Mapping": "string",  "Name": "string",  "SqlType": "string"  }  ],  "RecordEncoding": "string",  "RecordFormat": {  "MappingParameters": {  "CSVMappingParameters": {  "RecordColumnDelimiter": "string",  "RecordRowDelimiter": "string"  },  "JSONMappingParameters": {  "RecordRowPath": "string"  }  },  "RecordFormatType": "string"  }},  "InputStartingPositionConfiguration": {  "InputStartingPosition": "string"  },  "KinesisFirehoseInputDescription": {  "ResourceARN": "string",  "RoleARN": "string"  },  "KinesisStreamsInputDescription": {  "ResourceARN": "string",  "RoleARN": "string"  },  "NamePrefix": "string"  },  "LastUpdateTimestamp": number,  "OutputDescriptions": [  {  "DestinationSchema": {  "RecordFormatType": "string"  },  "KinesisFirehoseOutputDescription": {  "ResourceARN": "string",  "RoleARN": "string"  },  "KinesisStreamsOutputDescription": {  "ResourceARN": "string",  "RoleARN": "string"  },  "LambdaOutputDescription": {  "ResourceARN": "string",  "RoleARN": "string"  },  "Name": "string",  "OutputId": "string"  }  ],  "ReferenceDataSourceDescriptions": [  "}
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. 167)

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. 186) object

Errors

ResourceNotFoundException

Specified application can’t be found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DiscoverInputSchema

Infers a schema by evaluating sample records on the specified streaming source (Amazon Kinesis stream or Amazon Kinesis Firehose delivery stream) or S3 object. 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

```
{
  "InputProcessingConfiguration": {
    "InputLambdaProcessor": {
      "ResourceARN": "string",
      "RoleARN": "string"
    }
  },
  "InputStartingPositionConfiguration": {
    "InputStartingPosition": "string"
  },
  "ResourceARN": "string",
  "RoleARN": "string",
  "S3Configuration": {
    "BucketARN": "string",
    "FileKey": "string",
    "RoleARN": "string"
  }
}
```

Request Parameters

The request accepts the following data in JSON format.

**InputProcessingConfiguration (p. 171)**

The InputProcessingConfiguration (p. 206) to use to preprocess the records before discovering the schema of the records.

Type: InputProcessingConfiguration (p. 206) object

Required: No

**InputStartingPositionConfiguration (p. 171)**

Point at which you want Amazon Kinesis Analytics to start reading records from the specified streaming source discovery purposes.

Type: InputStartingPositionConfiguration (p. 210) object

Required: No

**ResourceARN (p. 171)**

Amazon Resource Name (ARN) of the streaming source.
DiscoverInputSchema

Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

Required: No

RoleARN (p. 171)

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: No

S3Configuration (p. 171)

Specify this parameter to discover a schema from data in an Amazon S3 object.

Type: S3Configuration (p. 243) object

Required: No

Response Syntax

```json
{
  "InputSchema": {
    "RecordColumns": [
      {
        "Mapping": "string",
        "Name": "string",
        "SqlType": "string"
      }
    ],
    "RecordEncoding": "string",
    "RecordFormat": {
      "MappingParameters": {
        "CSVMappingParameters": {
          "RecordColumnDelimiter": "string",
          "RecordRowDelimiter": "string"
        },
        "JSONMappingParameters": {
          "RecordRowPath": "string"
        }
      },
      "RecordFormatType": "string"
    }
  },
  "ParsedInputRecords": [ "string" ],
  "ProcessedInputRecords": [ "string" ],
  "RawInputRecords": [ "string" ]
}
```

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. 172)**

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. 247)` object

**ParsedInputRecords (p. 172)**

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 arrays of strings

**ProcessedInputRecords (p. 172)**

Stream data that was modified by the processor specified in the `InputProcessingConfiguration` parameter.

Type: Array of strings

**RawInputRecords (p. 172)**

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

**ServiceUnavailableException**

The service is unavailable. Back off and retry the operation.

HTTP Status Code: 500

**UnableToDetectSchemaException**

Data format is not valid. Amazon Kinesis Analytics is not able to detect schema for the given streaming source.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS Command Line Interface
• AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
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. 167).

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. 175)**

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


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

Required: No

**Limit (p. 175)**

Maximum number of applications to list.

Type: Integer


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. 175)**

List of ApplicationSummary objects.

Type: Array of ApplicationSummary (p. 189) objects

**HasMoreApplications (p. 175)**

Returns true if there are more applications to retrieve.

Type: Boolean

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
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. 167) operation.

After you start the application, you can stop the application from processing the input by calling the StopApplication (p. 179) operation.

This operation requires permissions to perform the kinesisanalytics:StartApplication action.

Request Syntax

```json
{
    "ApplicationName": "string",
    "InputConfigurations": [
        {
            "Id": "string",
            "InputStartingPositionConfiguration": {
                "InputStartingPosition": "string"
            }
        }
    ]
}
```

Request Parameters

The request accepts the following data in JSON format.

**ApplicationName (p. 177)**

Name of the application.

Type: String


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

Required: Yes

**InputConfigurations (p. 177)**

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. 198) 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
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. 167) 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

```json
{
    "ApplicationName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

**ApplicationName (p. 179)**

Name of the running application to stop.

Type: String


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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
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

```json
{
    "ApplicationName": "string",
    "ApplicationUpdate": {
        "ApplicationCodeUpdate": "string",
        "CloudWatchLoggingOptionUpdates": [
            {
                "CloudWatchLoggingOptionId": "string",
                "LogStreamARNUpdate": "string",
                "RoleARNUpdate": "string"
            }
        ],
        "InputUpdates": [
            {
                "InputId": "string",
                "InputParallelismUpdate": {
                    "CountUpdate": number
                },
                "InputProcessingConfigurationUpdate": {
                    "InputLambdaProcessorUpdate": {
                        "ResourceARNUpdate": "string",
                        "RoleARNUpdate": "string"
                    }
                },
                "InputSchemaUpdate": {
                    "RecordColumnUpdates": [
                        {
                            "Mapping": "string",
                            "Name": "string",
                            "SqlType": "string"
                        }
                    ],
                    "RecordEncodingUpdate": "string",
                    "RecordFormatUpdate": {
                        "MappingParameters": {
                            "CSVMappingParameters": {
                                "RecordColumnDelimiter": "string",
                                "RecordRowDelimiter": "string"
                            },
                            "JSONMappingParameters": {
                                "RecordRowPath": "string"
                            }
                        }
                    },
                    "RecordFormatType": "string"
                }
            }
        ],
        "KinesisFirehoseInputUpdate": {
            "ResourceARNUpdate": "string",
            "RoleARNUpdate": "string"
        },
        "KinesisStreamsInputUpdate": {
            "ResourceARNUpdate": "string",
            "RoleARNUpdate": "string"
        }
    }
}
```
"RoleARNUpdate": "string",

"NamePrefixUpdate": "string"

],

"OutputUpdates": [

{ "DestinationSchemaUpdate": {
    "RecordFormatType": "string"
  },
  "KinesisFirehoseOutputUpdate": {
    "ResourceARNUpdate": "string",
    "RoleARNUpdate": "string"
  },
  "KinesisStreamsOutputUpdate": {
    "ResourceARNUpdate": "string",
    "RoleARNUpdate": "string"
  },
  "LambdaOutputUpdate": {
    "ResourceARNUpdate": "string",
    "RoleARNUpdate": "string"
  },
  "NameUpdate": "string",
  "OutputId": "string"

}

],

"ReferenceDataSourceUpdates": [

{ "ReferenceId": "string",
  "ReferenceSchemaUpdate": {
    "RecordColumns": [
      { "Mapping": "string",
        "Name": "string",
        "SqlType": "string"
      }
    ],
    "RecordEncoding": "string",
    "RecordFormat": {
      "MappingParameters": {
        "CSVMappingParameters": {
          "RecordColumnDelimiter": "string",
          "RecordRowDelimiter": "string"
        },
        "JSONMappingParameters": {
          "RecordRowPath": "string"
        }
      },
      "RecordFormatType": "string"
    }
  },
  "S3ReferenceDataSourceUpdate": {
    "BucketARNUpdate": "string",
    "FileKeyUpdate": "string",
    "ReferenceRoleARNUpdate": "string"
  },
  "TableNameUpdate": "string"

}

],

"CurrentApplicationVersionId": number

Request Parameters

The request accepts the following data in JSON format.

**ApplicationName (p. 181)**

Name of the Amazon Kinesis Analytics application to update.

Type: String


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

Required: Yes

**ApplicationUpdate (p. 181)**

Describes application updates.

Type: ApplicationUpdate (p. 190) object

Required: Yes

**CurrentApplicationVersionId (p. 181)**

The current application version ID. You can use the DescribeApplication (p. 167) 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

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

**Data Types**

The following data types are supported:

- ApplicationDetail (p. 186)
- ApplicationSummary (p. 189)
- ApplicationUpdate (p. 190)
- CloudWatchLoggingOption (p. 191)
- CloudWatchLoggingOptionDescription (p. 192)
- CloudWatchLoggingOptionUpdate (p. 193)
- CSVMappingParameters (p. 194)
- DestinationSchema (p. 195)
- Input (p. 196)
- InputConfiguration (p. 198)
- InputDescription (p. 199)
- InputLambdaProcessor (p. 201)
- InputLambdaProcessorDescription (p. 202)
- InputLambdaProcessorUpdate (p. 203)
- InputParallelism (p. 204)
- InputParallelismUpdate (p. 205)
- InputProcessingConfiguration (p. 206)
- InputProcessingConfigurationDescription (p. 207)
- InputProcessingConfigurationUpdate (p. 208)
- InputSchemaUpdate (p. 209)
- InputStartingPositionConfiguration (p. 210)
- InputUpdate (p. 211)
- JSONMappingParameters (p. 213)
- KinesisFirehoseInput (p. 214)
- KinesisFirehoseInputDescription (p. 215)
- KinesisFirehoseInputUpdate (p. 216)
- KinesisFirehoseOutput (p. 217)
- KinesisFirehoseOutputDescription (p. 218)
- KinesisFirehoseOutputUpdate (p. 219)
- KinesisStreamsInput (p. 220)
- KinesisStreamsInputDescription (p. 221)
- KinesisStreamsInputUpdate (p. 222)
- KinesisStreamsOutput (p. 223)
- KinesisStreamsOutputDescription (p. 224)
- KinesisStreamsOutputUpdate (p. 225)
- LambdaOutput (p. 226)
- LambdaOutputDescription (p. 227)
- LambdaOutputUpdate (p. 228)
- MappingParameters (p. 229)
- Output (p. 230)
- OutputDescription (p. 232)
- OutputUpdate (p. 234)
- RecordColumn (p. 236)
- RecordFormat (p. 237)
- ReferenceDataSource (p. 238)
- ReferenceDataSourceDescription (p. 239)
- ReferenceDataSourceUpdate (p. 241)
- S3Configuration (p. 243)
- S3ReferenceDataSource (p. 244)
- S3ReferenceDataSourceDescription (p. 245)
- S3ReferenceDataSourceUpdate (p. 246)
- SourceSchema (p. 247)
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
  - Pattern: `arn:.*`
  - 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
  - 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.
ApplicationDetail

Type: Long

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

Required: Yes

CloudWatchLoggingOptionDescriptions

Describes the CloudWatch log streams that are configured to receive application messages. For more information about using CloudWatch log streams with Amazon Kinesis Analytics applications, see Working with Amazon CloudWatch Logs.

Type: Array of CloudWatchLoggingOptionDescription (p. 192) objects

Required: No

CreateTimestamp

Time stamp 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. 199) objects

Required: No

LastUpdateTimestamp

Time stamp 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. 232) objects

Required: No

ReferenceDataSourceDescriptions

Describes reference data sources configured for the application. For more information, see Configuring Application Input.

Type: Array of ReferenceDataSourceDescription (p. 239) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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:.*

Required: Yes

ApplicationName

Name of the application.

Type: String


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

Required: Yes

ApplicationStatus

Status of the application.

Type: String

Valid Values: DELETING | STARTING | STOPPING | READY | RUNNING | UPDATING

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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

**CloudWatchLoggingOptionUpdates**

Describes application CloudWatch logging option updates.

Type: Array of `CloudWatchLoggingOptionUpdate (p. 193)` objects

Required: No

**InputUpdates**

Describes application input configuration updates.

Type: Array of `InputUpdate (p. 211)` objects

Required: No

**OutputUpdates**

Describes application output configuration updates.

Type: Array of `OutputUpdate (p. 234)` objects

Required: No

**ReferenceDataSourceUpdates**

Describes application reference data source updates.

Type: Array of `ReferenceDataSourceUpdate (p. 241)` objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
CloudWatchLoggingOption

Provides a description of CloudWatch logging options, including the log stream Amazon Resource Name (ARN) and the role ARN.

Contents

LogStreamARN

ARN of the CloudWatch log to receive application messages.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

Required: Yes

RoleARN

IAM ARN of the role to use to send application messages. Note: To write application messages to CloudWatch, the IAM role that is used must have the PutLogEvents policy action enabled.

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
CloudWatchLoggingOptionDescription

Description of the CloudWatch logging option.

Contents

CloudWatchLoggingOptionId

ID of the CloudWatch logging option description.

Type: String


Pattern: \[a-zA-Z0-9\-_]+\+

Required: No

LogStreamARN

ARN of the CloudWatch log to receive application messages.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:\.*

Required: Yes

RoleARN

IAM ARN of the role to use to send application messages. Note: To write application messages to CloudWatch, the IAM role used must have the PutLogEvents policy action enabled.

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
CloudWatchLoggingOptionUpdate

Describes CloudWatch logging option updates.

Contents

CloudWatchLoggingOptionId

ID of the CloudWatch logging option to update

Type: String


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

Required: Yes

LogStreamARNUpdate

ARN of the CloudWatch log to receive application messages.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

Required: No

RoleARNUpdate

IAM ARN of the role to use to send application messages. Note: To write application messages to
CloudWatch, the IAM role used must have the PutLogEvents policy action enabled.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws:iam::\d{12}:role/\?[a-zA-Z0-9=,.@\-_]+\]

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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

Length Constraints: Minimum length of 1.

Required: Yes

RecordRowDelimiter

Row delimiter. For example, in a CSV format, \'\n\' is the typical row delimiter.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**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 is routed to these in-application input streams.
  
  (see Configuring Application Input.

  Type: InputParallelism (p. 204) object

  Required: No

- **InputProcessingConfiguration**
  
  The InputProcessingConfiguration (p. 206) for the input. An input processor transforms records as they are received from the stream, before the application's SQL code executes. Currently, the only input processing configuration available is InputLambdaProcessor (p. 201).

  Type: InputProcessingConfiguration (p. 206) 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. 247) object

  Required: Yes

- **KinesisFirehoseInput**

  If the streaming source is an Amazon Kinesis Firehose delivery stream, identifies the delivery stream's ARN and an IAM role that enables Amazon Kinesis Analytics to access the stream on your behalf.

  Note: Either KinesisStreamsInput or KinesisFirehoseInput is required.

  Type: KinesisFirehoseInput (p. 214) 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.

  Note: Either KinesisStreamsInput or KinesisFirehoseInput is required.

  Type: KinesisStreamsInput (p. 220) object

  Required: No
**NamePrefix**

Name prefix to use when creating an in-application stream. Suppose that you specify a prefix "MyInApplicationStream." Amazon Kinesis Analytics then creates one or more (as per the InputParallelism count you specified) in-application streams with names "MyInApplicationStream_001," "MyInApplicationStream_002," and so on.

Type: String


Pattern: [a-zA-Z][a-zA-Z0-9_]+

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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. 167) operation.

Type: String


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. 210) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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
  
  
  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
  
  
  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. 204) object
  
  Required: No

InputProcessingConfigurationDescription

  The description of the preprocessor that executes on records in this input before the application's code is run.
  
  Type: InputProcessingConfigurationDescription (p. 207) 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.
  
  Type: SourceSchema (p. 247) object
  
  Required: No

InputStartingPositionConfiguration

  Point at which the application is configured to read from the input stream.
  
  Type: InputStartingPositionConfiguration (p. 210) object
Required: No

**KinesisFirehoseInputDescription**

If an Amazon Kinesis Firehose delivery stream is configured as a streaming source, provides the delivery stream's ARN and an IAM role that enables Amazon Kinesis Analytics to access the stream on your behalf.

Type: `KinesisFirehoseInputDescription (p. 215)` object

Required: No

**KinesisStreamsInputDescription**

If an Amazon Kinesis stream is configured as streaming source, provides Amazon Kinesis stream's Amazon Resource Name (ARN) and an IAM role that enables Amazon Kinesis Analytics to access the stream on your behalf.

Type: `KinesisStreamsInputDescription (p. 221)` object

Required: No

**NamePrefix**

In-application name prefix.

Type: String


Pattern: `[a-zA-Z][a-zA-Z0-9_]`

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
InputLambdaProcessor

An object that contains the Amazon Resource Name (ARN) of the AWS Lambda function that is used to preprocess records in the stream, and the ARN of the IAM role that is used to access the AWS Lambda function.

Contents

ResourceARN

The ARN of the AWS Lambda function that operates on records in the stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

Required: Yes

RoleARN

The ARN of the IAM role that is used to access the AWS Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws:iam::\d{12}:role/\?[a-zA-Z0-9+=,.@\-_/]+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
InputLambdaProcessorDescription

An object that contains the Amazon Resource Name (ARN) of the AWS Lambda function that is used to preprocess records in the stream, and the ARN of the IAM role that is used to access the AWS Lambda expression.

Contents

ResourceARN

The ARN of the AWS Lambda function that is used to preprocess the records in the stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

Required: No

RoleARN

The ARN of the IAM role that is used to access the AWS Lambda function.

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
InputLambdaProcessorUpdate

Represents an update to the InputLambdaProcessor (p. 201) that is used to preprocess the records in the stream.

Contents

ResourceARNUpdate

The Amazon Resource Name (ARN) of the new AWS Lambda function that is used to preprocess the records in the stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

Required: No

RoleARNUpdate

The ARN of the new IAM role that is used to access the AWS Lambda function.

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**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 64.

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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 64.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
InputProcessingConfiguration

Provides a description of a processor that is used to preprocess the records in the stream before being processed by your application code. Currently, the only input processor available is AWS Lambda.

Contents

InputLambdaProcessor

The InputLambdaProcessor (p. 201) that is used to preprocess the records in the stream before being processed by your application code.

Type: InputLambdaProcessor (p. 201) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**InputProcessingConfigurationDescription**

Provides configuration information about an input processor. Currently, the only input processor available is **AWS Lambda**.

**Contents**

**InputLambdaProcessorDescription**

Provides configuration information about the associated **InputLambdaProcessorDescription (p. 202)**.

Type: **InputLambdaProcessorDescription (p. 202)** object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
InputProcessingConfigurationUpdate

Describes updates to an InputProcessingConfiguration (p. 206).

Contents

InputLambdaProcessorUpdate

Provides update information for an InputLambdaProcessor (p. 201).

Type: InputLambdaProcessorUpdate (p. 203) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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` 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` object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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 time stamp 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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
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. 205) object
Required: No

InputProcessingConfigurationUpdate

Describes updates for an input processing configuration.
Type: InputProcessingConfigurationUpdate (p. 208) 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. 209) object
Required: No

KinesisFirehoseInputUpdate

If an Amazon Kinesis Firehose delivery stream is the streaming source to be updated, provides an updated stream ARN and IAM role ARN.
Type: KinesisFirehoseInputUpdate (p. 216) object
Required: No

KinesisStreamsInputUpdate

If an Amazon Kinesis stream is the streaming source to be updated, provides an updated stream Amazon Resource Name (ARN) and IAM role ARN.
Type: KinesisStreamsInputUpdate (p. 222) object
Required: No

NamePrefixUpdate

Name prefix for in-application streams that Amazon Kinesis Analytics creates for the specific streaming source.
Type: String


Pattern: [a-zA-Z][a-zA-Z0-9_]+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
KinesisFirehoseInput

Identifies an Amazon Kinesis Firehose delivery stream as the streaming source. You provide the 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 delivery stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

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 that the role has the 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-Z0-9+=,.@\-_/]+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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:.*

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
KinesisFirehoseInputUpdate

When updating application input configuration, provides information about an Amazon Kinesis Firehose delivery stream as the streaming source.

Contents

ResourceARNUpdate

Amazon Resource Name (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:.*

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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:.*

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-Z0-9+=,.@/-_\]\]+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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:.*

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-Z0-9_+=,.@\-_]+\+\.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
KinesisFirehoseOutputUpdate

When updating an output configuration using the UpdateApplication (p. 181) 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:.*

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-Z0-9+=,.@\-_/]+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
KinesisStreamsInput

Identifies an Amazon Kinesis stream as the streaming source. You provide the 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 Amazon Kinesis stream to read.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

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/?\[a-zA-Z0-9+=,.@\-_\//\]+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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:.*

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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:.*

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
KinesisStreamsOutput

When configuring application output, identifies an 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:.*`

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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
- Pattern: arn:.*
- Required: No

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to access the stream.

- Type: String
- Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z0-9+=,.@\-_\ ]+
- Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
KinesisStreamsOutputUpdate

When updating an output configuration using the UpdateApplication (p. 181) 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:.*

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-Z0-9+=,.@\-_/]+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LambdaOutput

When configuring application output, identifies an AWS Lambda function as the destination. You provide the function Amazon Resource Name (ARN) and also an IAM role ARN that Amazon Kinesis Analytics can use to write to the function on your behalf.

Contents

ResourceARN

Amazon Resource Name (ARN) of the destination Lambda function to write to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

Required: Yes

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to write to the destination function 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-Z0-9+=,.@\-_\/]+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LambdaOutputDescription

For an application output, describes the AWS Lambda function configured as its destination.

Contents

ResourceARN

Amazon Resource Name (ARN) of the destination Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

Required: No

RoleARN

ARN of the IAM role that Amazon Kinesis Analytics can assume to write to the destination function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws:iam::\d{12}:role/\?_[a-zA-Z0-9+=,.@\-_]+\+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LambdaOutputUpdate

When updating an output configuration using the UpdateApplication (p. 181) operation, provides information about an AWS Lambda function configured as the destination.

Contents

ResourceARNUpdate

Amazon Resource Name (ARN) of the destination Lambda function.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Pattern: arn:.*
Required: No

RoleARNUpdate

ARN of the IAM role that Amazon Kinesis Analytics can assume to write to the destination function 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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. 194) object

Required: No

JSONMappingParameters

Provides additional mapping information when JSON is the record format on the streaming source.

Type: JSONMappingParameters (p. 213) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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. 195) object

Required: Yes

KinesisFirehoseOutput

Identifies an Amazon Kinesis Firehose delivery stream as the destination.

Type: KinesisFirehoseOutput (p. 217) object

Required: No

KinesisStreamsOutput

Identifies an Amazon Kinesis stream as the destination.

Type: KinesisStreamsOutput (p. 223) object

Required: No

LambdaOutput

Identifies an AWS Lambda function as the destination.

Type: LambdaOutput (p. 226) object

Required: No

Name

Name of the in-application stream.

Type: String


Pattern: [a-zA-Z][a-zA-Z0-9_]+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
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. 195) object
Required: No

KinesisFirehoseOutputDescription

Describes the Amazon Kinesis Firehose delivery stream configured as the destination where output is written.
Type: KinesisFirehoseOutputDescription (p. 218) object
Required: No

KinesisStreamsOutputDescription

Describes Amazon Kinesis stream configured as the destination where output is written.
Type: KinesisStreamsOutputDescription (p. 224) object
Required: No

LambdaOutputDescription

Describes the AWS Lambda function configured as the destination where output is written.
Type: LambdaOutputDescription (p. 227) object
Required: No

Name

Name of the in-application stream configured as output.
Type: String
Pattern: [a-zA-Z][a-zA-Z0-9_.-]+
Required: No

OutputId

A unique identifier for the output configuration.
Type: String
Pattern: [a-zA-Z0-9_.-]+
Required: No
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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. 195) object

Required: No

KinesisFirehoseOutputUpdate

Describes an Amazon Kinesis Firehose delivery stream as the destination for the output.

Type: KinesisFirehoseOutputUpdate (p. 219) object

Required: No

KinesisStreamsOutputUpdate

Describes an Amazon Kinesis stream as the destination for the output.

Type: KinesisStreamsOutputUpdate (p. 225) object

Required: No

LambdaOutputUpdate

Describes an AWS Lambda function as the destination for the output.

Type: LambdaOutputUpdate (p. 228) 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


Pattern: [a-zA-Z][a-zA-Z0-9_]+

Required: No

OutputId

Identifies the specific output configuration that you want to update.

Type: String


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

Required: Yes
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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
Length Constraints: Minimum length of 1.
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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. 229) object

Required: No

RecordFormatType

The type of record format.

Type: String

Valid Values: JSON | CSV

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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. 247) 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. 181) operation to trigger reloading of data into your application.

Type: S3ReferenceDataSource (p. 244) object

Required: No

TableName

Name of the in-application table to create.

Type: String


Pattern: [a-zA-Z][a-zA-Z0-9_]+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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. 149)
operation.

Type: String
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. 247) 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. 245) object
Required: Yes

TableName

The in-application table name created by the specific reference data source configuration.

Type: String
Pattern: [a-zA-Z][a-zA-Z0-9_.-]+
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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. 167) operation to get this value.

Type: String


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. 247) 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. 246) object

Required: No

TableNameUpdate

In-application table name that is created by this update.

Type: String


Pattern: [a-zA-Z][a-zA-Z0-9_]*

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
S3Configuration

Provides a description of an Amazon S3 data source, including the Amazon Resource Name (ARN) of the S3 bucket, the ARN of the IAM role that is used to access the bucket, and the name of the Amazon S3 object that contains the data.

Contents

BucketARN

ARN of the S3 bucket that contains the data.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:.*

Required: Yes

FileKey

The name of the object that contains the data.

Type: String


Required: Yes

RoleARN

IAM ARN of the role used to access the data.

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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. 181) 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
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+=,.@-_]+\d*

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**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 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 object

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
The following table describes the important changes to the documentation since the last release of Amazon Kinesis Data Analytics.

- **API version:** 2015-08-14
- **Latest documentation update:** December 20, 2017

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lambda function as a destination</td>
<td>Send analytics results to a Lambda function as a destination. For more information, see Using a Lambda Function as Output (p. 32).</td>
<td>December 20, 2017</td>
</tr>
<tr>
<td>New RandomCutForestWithExplanation function</td>
<td>Get an explanation of what fields contribute to an anomaly score in a data stream. For more information, see Example: Detecting Data Anomalies and Getting an Explanation (RANDOM_CUT_FOREST_WITH_EXPLANATION Function) (p. 101).</td>
<td>November 2, 2017</td>
</tr>
<tr>
<td>Schema discovery on static data</td>
<td>Run schema discovery on static data stored in an Amazon S3 bucket. For more information, see Using the Schema Discovery Feature on Static Data (p. 18).</td>
<td>October 6, 2017</td>
</tr>
<tr>
<td>Lambda preprocessing feature</td>
<td>Preprocess records in an input stream with AWS Lambda before analysis. For more information, see Using the Lambda Preprocessing Feature (p. 21).</td>
<td>October 6, 2017</td>
</tr>
<tr>
<td>Auto scaling applications</td>
<td>Automatically increase the data throughput of your application with auto scaling. For more information, see Automatically Scaling Applications to Increase Throughput (p. 40).</td>
<td>September 13, 2017</td>
</tr>
<tr>
<td>Multiple in-application input streams</td>
<td>Increase application throughput with multiple in-application streams. For more information, see Parallelizing Input Streams for Increased Throughput (p. 26).</td>
<td>June 29, 2017</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>Guide to using the AWS Management Console for Kinesis Data Analytics</td>
<td>Edit an inferred schema and SQL code using the schema editor and SQL editor in the Kinesis Data Analytics console. For more information, see Step 4 (Optional) Edit the Schema and SQL Code Using the Console (p. 52).</td>
<td>April 7, 2017</td>
</tr>
<tr>
<td>Public release</td>
<td>Public release of the <em>Amazon Kinesis Data Analytics Developer Guide</em>.</td>
<td>August 11, 2016</td>
</tr>
<tr>
<td>Preview release</td>
<td>Preview release of the <em>Amazon Kinesis Data Analytics Developer Guide</em>.</td>
<td>January 29, 2016</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.