aws

SQL Developer Guide

Amazon Kinesis Data Analytics for SQL Applications Developer Guide

Amazon Kinesis Data Analytics for SQL Applications Developer Guide: SQL Developer Guide

Table of Contents

After careful consideration, we have decided to discontinue Amazon Kinesis Data Analytics for SQL applications:

1. From **September 1, 2025**, we won't provide any bug fixes for Amazon Kinesis Data Analytics for SQL applications because we will have limited support for it, given the upcoming discontinuation.

2. From **October 15, 2025**, you will not be able to create new Kinesis Data Analytics for SQL applications.

3. We will delete your applications starting **January 27, 2026**. You will not be able to start or operate your Amazon Kinesis Data Analytics for SQL applications. Support will no longer be available for Amazon Kinesis Data Analytics for SQL from that time. For more information, see Amazon Kinesis Data Analytics for SQL Applications discontinuation.

Amazon Kinesis Data Analytics for SQL Applications discontinuation

Discontinuation plan

After careful consideration, we have made the decision to discontinue Amazon Kinesis Data Analytics for SQL applications. To help you plan and migrate away from Amazon Kinesis Data Analytics for SQL applications, we will discontinue the offering gradually over 15 months. These are important dates to note, **September 1, 2025**, **October 15, 2025**, and **January 27, 2026**.

1. On **October 15, 2025**, we will stop your applications and place them into a READY state. You will be able to *re-start* your applications at that time and continue to use your applications as normal, subject to service limits.
2. From **October 15, 2025**, you will not be able to create *new* Amazon Kinesis Data Analytics for SQL applications. You will be able to run any *existing* applications as normal, subject to service limits.
3. We will delete your applications starting **January 27, 2026**. You will not be able to start or operate your Amazon Kinesis Data Analytics for SQL applications. Support will no longer be available for Amazon Kinesis Data Analytics for SQL applications from that time.

We recommend that you migrate your applications to Amazon Managed Service for Apache Flink or Amazon Managed Service for Apache Flink Studio before October 15, 2025. For resources to assist with your migration, see Migrating to Managed Service for Apache Flink Studio Examples. To learn more about Amazon Managed Service for Apache Flink or Amazon Managed Service for Apache Flink Studio, see the Amazon Managed Service for Apache Flink developer guide.

> **Note**
>
> To view a full list of your Amazon Kinesis Data Analytics for SQL applications, you can call the `ListApplications` API. For more information, see ListApplications.

Migrating to Amazon Managed Service for Apache Flink Studio

To learn more about migrating your applications and view code and architecture examples, see Migrating to Managed Service for Apache Flink Studio Examples.

FAQ

Why are you no longer offering Amazon Kinesis Data Analytics for SQL applications?

We have found that customers prefer Amazon Managed Service for Apache Flink offerings for real-time data stream processing workloads. Amazon Managed Service for Apache Flink is a serverless, low latency, highly scalable and available real-time stream processing service using Apache Flink, an open source engine for processing data streams. Amazon Managed Service for Apache Flink offers functionality such as native scaling, exactly once processing semantics, multi-language support (including SQL), over 40 source and destination connectors, durable application state, and more. These features help customers build end to end streaming pipelines and ensure the accuracy and timeliness of data.

What are customers' options now?

We recommend customers upgrade their existing Amazon Kinesis Data Analytics for SQL applications to either Amazon Managed Service for Apache Flink Studio or Amazon Managed Service for Apache Flink. In Amazon Managed Service for Apache Flink Studio, customers create queries using SQL, Python, or Scala using interactive notebooks. For long running applications in Amazon Kinesis Data Analytics for SQL, we recommend Amazon Managed Service for Apache Flink, where customers can create applications using Java, Python, Scala, and embedded SQL using all of Apache Flink's APIs, connectors, and more.

How many customers are using Amazon Kinesis Data Analytics for SQL applications?

We can't disclose customer information details.

How do customers upgrade from Amazon Kinesis Data Analytics for SQL applications to an Amazon Managed Service for Apache Flink offering?

To upgrade to Amazon Managed Service for Apache Flink or Amazon Managed Service for Apache Flink Studio, customers must re-create their application. To help, we have provided a library of common SQL queries and how to re-write them in Amazon Managed Service for Apache Flink Studio. See Replicating Kinesis Data Analytics for SQL Queries in Managed Service for Apache Flink

Studio. We have also provided common pattern architectures that customers can followv if they are building long running applications or using machine learning in Amazon Managed Service for Apache Flink. See Replacing Kinesis Data Firehose as a source with Kinesis Data Streams

To learn more about Amazon Managed Service for Apache Flink, see Amazon Managed Service for Apache Flink.

For migration guides, see Migrating to Managed Service for Apache Flink Studio Examples.

Will Amazon Managed Service for Apache Flink support the existing Amazon Kinesis Data Analytics for SQL applications features?

Amazon Managed Service for Apache Flink supports many of the concepts available in Amazon Kinesis Data Analytics for SQL applications such as connectors and windowing, as well as features that were unavailable in Amazon Kinesis Data Analytics for SQL applications, such as native scaling, exactly-once processing semantics, multi-language support (including SQL), over 40 source and destination connectors, durable application state, and more.

Migrating to Managed Service for Apache Flink Studio Examples

After careful consideration, we have made the decision to discontinue Amazon Kinesis Data Analytics for SQL applications. To help you plan and migrate away from Amazon Kinesis Data Analytics for SQL applications, we will discontinue the offering gradually over 15 months. These are important dates to note, **September 1, 2025**, **October 15, 2025**, and **January 27, 2026**.

1. From **September 1, 2025**, we won't provide any bug fixes for Amazon Kinesis Data Analytics for SQL applications because we will have limited support for it, given the upcoming discontinuation.
2. From **October 15, 2025**, you will not be able to create new Amazon Kinesis Data Analytics for SQL applications.
3. We will delete your applications starting **January 27, 2026**. You will not be able to start or operate your Amazon Kinesis Data Analytics for SQL applications. Support will no longer be available for Amazon Kinesis Data Analytics for SQL applications from that time. To learn more, see Amazon Kinesis Data Analytics for SQL Applications discontinuation.

We recommend that you use Amazon Managed Service for Apache Flink. It combines ease of use with advanced analytical capabilities, letting you build stream processing applications in minutes.

This section provides code and architecture examples to help you move your Amazon Kinesis Data Analytics for SQL applications workloads to Managed Service for Apache Flink.

For additional information, also see this AWS blog post: Migrate from Amazon Kinesis Data Analytics for SQL Applications to Managed Service for Apache Flink Studio.

Replicating Kinesis Data Analytics for SQL Queries in Managed Service for Apache Flink Studio

To migrate your workloads to Managed Service for Apache Flink Studio or Managed Service for Apache Flink, this section provides query translations you can use for common use cases.

Before you explore these examples, we recommend you first review Using a Studio notebook with a Managed Service for Apache Flink.

Re-creating Kinesis Data Analytics for SQL queries in Managed Service for Apache Flink Studio

The following options provide translations of common SQL-based Kinesis Data Analytics application queries to Managed Service for Apache Flink Studio.

Multi-Step application

SQL-based Kinesis Data Analytics application

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

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

Managed Service for Apache Flink Studio

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

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

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

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

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

DROP TABLE IF EXISTS IN_APP_STREAM_02;

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

DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;

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


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

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


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


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

Transforming DateTime values

SQL-based Kinesis Data Analytics application

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

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

Managed Service for Apache Flink Studio

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

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

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

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

Simple alerts

SQL-based Kinesis Data Analytics application

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

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

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

Managed Service for Apache Flink Studio

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

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

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

Throttled alerts

SQL-based Kinesis Data Analytics application

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

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

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

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

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

Managed Service for Apache Flink Studio

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

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

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

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

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

Aggregating Partial Results from a Query

SQL-based Kinesis Data Analytics application

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

CREATE
```

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

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

Managed Service for Apache Flink Studio

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

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

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

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

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

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

```
         )
;

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

Transforming string values

SQL-based Kinesis Data Analytics application

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

Managed Service for Apache Flink Studio

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

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

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

Replacing a substring using Regex

SQL-based Kinesis Data Analytics application

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

Managed Service for Apache Flink Studio

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

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

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

Regex log parse

SQL-based Kinesis Data Analytics application

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

Managed Service for Apache Flink Studio

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

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

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

Transforming DateTime values

SQL-based Kinesis Data Analytics application

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

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

Managed Service for Apache Flink Studio

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

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

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

Windows and aggregation

SQL-based Kinesis Data Analytics application

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

Managed Service for Apache Flink Studio

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

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

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

Tumbling Window using Rowtime

SQL-based Kinesis Data Analytics application

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

Managed Service for Apache Flink Studio

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

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

Retrieving the most frequently occurring values (TOP_K_ITEMS_TUMBLING)

SQL-based Kinesis Data Analytics application

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

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

Managed Service for Apache Flink Studio

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

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

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

Approximate Top-K items

SQL-based Kinesis Data Analytics application

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

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

Managed Service for Apache Flink Studio

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


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

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

Parsing Web Logs (W3C_LOG_PARSE Function)

SQL-based Kinesis Data Analytics application

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

Managed Service for Apache Flink Studio

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

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

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

Split Strings into Multiple Fields (VARIABLE_COLUMN_LOG_PARSE Function)

SQL-based Kinesis Data Analytics application

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

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

Managed Service for Apache Flink Studio

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

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

Joins

SQL-based Kinesis Data Analytics application

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

Managed Service for Apache Flink Studio

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

Query 2 - CREATE TABLE CompanyName (
```

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

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

Errors

SQL-based Kinesis Data Analytics application

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

Managed Service for Apache Flink Studio

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

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

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

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

Migrating Random Cut Forest workloads

If you are looking to move workloads that use Random Cut Forest from Kinesis Analytics for SQL to Managed Service for Apache Flink, this AWS blog post demonstrates how to use Managed Service for Apache Flink to run an online RCF algorithm for anomaly detection.

Replacing Kinesis Data Firehose as a source with Kinesis Data Streams

See Converting-KDASQL-KDAStudio/ for a full tutorial.

In the following exercise, you will change your data flow to use Amazon Managed Service for Apache Flink Studio. This will also mean switching from Amazon Kinesis Data Firehose to Amazon Kinesis Data Streams.

First we share a typical KDA-SQL architecture, before showing how you can replace this using Amazon Managed Service for Apache Flink Studio and Amazon Kinesis Data Streams. Alternatively you can launch the CloudFormation template here:

Amazon Kinesis Data Analytics-SQL and Amazon Kinesis Data Firehose

Here is the Amazon Kinesis Data Analytics SQL architectural flow:

We first examine the setup of a legacy Amazon Kinesis Data Analytics-SQL and Amazon Kinesis Data Firehose. The use case is a trading market where trading data, including stock ticker and price, streams from external sources to Amazon Kinesis systems. Amazon Kinesis Data Analytics for SQL uses the input stream to execute Windowed queries like Tumbling window to determine the trade volume and the `min`, `max`, and `average` trade price over a one-minute window for each stock ticker.

Amazon Kinesis Data Analytics-SQL is set up to ingest data from the Amazon Kinesis Data Firehose API. After processing, Amazon Kinesis Data Analytics-SQL sends the processed data to another Amazon Kinesis Data Firehose, which then saves the output in an Amazon S3 bucket.

In this case, you use Amazon Kinesis Data Generator. Amazon Kinesis Data Generator allows you to send test data to your Amazon Kinesis Data Streams or Amazon Kinesis Data Firehose delivery streams. To get started, follow the instructions here. Use the CloudFormation template here in place of the one provided in the instructions:.

Once you run the CloudFormation template, the output section will provide the Amazon Kinesis Data Generator url. Log in to the portal using the Cognito user id and password you set up here. Select the Region and the target stream name. For current state, choose the Amazon Kinesis Data Firehose Delivery streams. For the new state, choose the Amazon Kinesis Data Firehose Streams name. You can create multiple templates, depending on your requirements, and test the template using the **Test template** button before sending it to the target stream.

Following is a sample payload using Amazon Kinesis Data Generator. The data generator targets the input Amazon Kinesis Firehose Streams to stream the data continuously. The Amazon Kinesis SDK client can send data from other producers as well.

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

The following JSON is used to generate a random series of trade time and date, stock ticker, and stock price:

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

Once you choose **Send data**, the generator will start sending mock data.

External systems stream the data to Amazon Kinesis Data Firehose. Using Amazon Kinesis Data Analytics for SQL Applications, you can analyze streaming data using standard SQL. The service enables you to author and run SQL code against streaming sources to perform time-series analytics, feed real-time dashboards, and create real-time metrics. Amazon Kinesis Data Analytics for SQL Applications could create a destination stream from SQL queries on the input stream and send the destination stream to another Amazon Kinesis Data Firehose. The destination Amazon Kinesis Data Firehose could send the analytical data to Amazon S3 as the final state.

Amazon Kinesis Data Analytics-SQL legacy code is based on an extension of SQL Standard.

You use the following query in Amazon Kinesis Data Analytics-SQL. You first create a destination stream for the query output. Then, you would use PUMP, which is an Amazon Kinesis Data Analytics Repository Object (an extension of the SQL Standard) that provides a continuously running `INSERT INTO stream SELECT ... FROM` query functionality, thereby enabling the results of a query to be continuously entered into a named stream.

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (EVENT_TIME TIMESTAMP,
INGEST_TIME TIMESTAMP,
TICKER VARCHAR(16),
VOLUME BIGINT,
AVG_PRICE DOUBLE,
MIN_PRICE DOUBLE,
MAX_PRICE DOUBLE);

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

The preceding SQL uses two time windows – `tradeTimestamp` that comes from the incoming stream payload and `ROWTIME`. `tradeTimestamp` is also called `Event Time` or `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.

In-application streams also 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.

The SQL determines the count of ticker as `volume`, `min`, `max`, and `average` price over a 60-second interval.

Using each of these times in windowed queries that are time-based has advantages and disadvantages. Choose one or more of these times, and a strategy to deal with the relevant disadvantages based on your use case scenario.

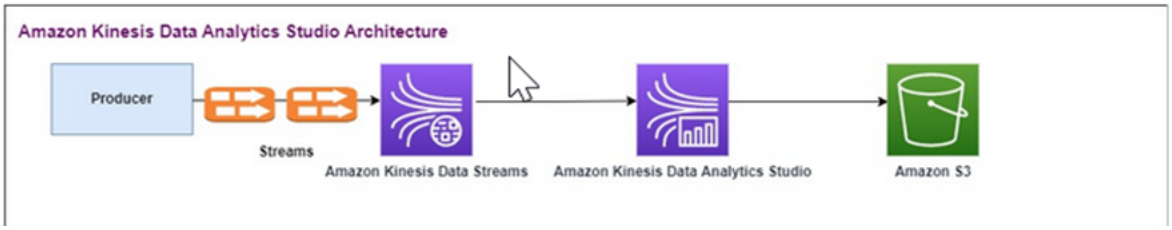
A two-window strategy uses two time-based, both `ROWTIME` and one of the other times like the event time.

- 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 that you want to associate 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.

Amazon Managed Service for Apache Flink Studio

In the updated architecture, you replace Amazon Kinesis Data Firehose with Amazon Kinesis Data Streams. Amazon Kinesis Data Analytics for SQL Applications are replaced by Amazon Managed Service for Apache Flink Studio. Apache Flink code is run interactively within an Apache Zeppelin Notebook. Amazon Managed Service for Apache Flink Studio sends the aggregated trade data to an Amazon S3 bucket for storage. The steps are shown following:

Here is the Amazon Managed Service for Apache Flink Studio architectural flow:



Create a Kinesis Data Stream

To create a data stream using the console

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. In the navigation bar, expand the Region selector and choose a Region.
3. Choose **Create data stream**.
4. On the **Create Kinesis stream** page, enter a name for your data stream and accept the default **On-demand** capacity mode.

   With the **On-demand** mode, you can then choose **Create Kinesis stream** to create your data stream.

   On the **Kinesis streams** page, your stream's **Status** is **Creating** while the stream is being created. When the stream is ready to use, the **Status** changes to **Active**.
5. Choose the name of your stream. The **Stream Details** page displays a summary of your stream configuration, along with monitoring information.
6. In the Amazon Kinesis Data Generator, change the Stream/delivery stream to the new Amazon Kinesis Data Streams: **TRADE_SOURCE_STREAM**.

   JSON and Payload will be the same as you used for Amazon Kinesis Data Analytics-SQL. Use the Amazon Kinesis Data Generator to produce some sample trading payload data and target the **TRADE_SOURCE_STREAM** Data Stream for this exercise:

   ```
   {{date.now(YYYY-MM-DD HH:mm:ss.SSS)}},
   "{{random.arrayElement(["AAPL","AMZN","MSFT","META","GOOGL"])}}",
   {{random.number(2000)}}
   ```
7. On the AWS Management Console go to Managed Service for Apache Flink and then choose **Create application**.
8. On the left navigation pane, choose **Studio notebooks** and then choose **Create studio notebook**.
9. Enter a name for the studio notebook.
10. Under **AWS Glue database**, provide an existing AWS Glue database that will define the metadata for your sources and destinations. If you don’t have an AWS Glue database, choose **Create** and do the following:

a. In the AWS Glue console, choose **Databases** under **Data catalog** from the left-hand menu.
b. Choose **Create database**
c. In the **Create database** page, enter a name for the database. In the **Location - optional** section, choose **Browse Amazon S3** and select the Amazon S3 bucket. If you don't have an Amazon S3 bucket already set up, you can skip this step and come back to it later.
d. (Optional). Enter a description for the database.
e. Choose **Create database**.

11. Choose **Create notebook**
12. Once your notebook is created, choose **Run**.
13. Once the notebook has been successfully started, launch a Zeppelin notebook by choosing **Open in Apache Zeppelin**.
14. On the Zeppelin Notebook page, choose **Create new note** and name it *MarketDataFeed*.

The Flink SQL code is explained following, but first this is what a Zeppelin notebook screen looks like. Each window within the notebook is a separate code block, and they can be run one at a time.

Amazon Managed Service for Apache Flink Studio Code

Amazon Managed Service for Apache Flink Studio uses Zeppelin Notebooks to run the code. Mapping is done for this example to ssql code based on Apache Flink 1.13. The code in the Zeppelin Notebook is shown following, one block at a time.

Before running any code in your Zeppelin Notebook, Flink configuration commands must be run. If you need to change any configuration setting after running code (ssql, Python, or Scala), you must stop and restart your notebook. In this example, you must set checkpointing. Checkpointing is required so that you can stream data to a file in Amazon S3. This allows data streaming to Amazon S3 to be flushed to a file. The following statement sets the interval to 5000 milliseconds.

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

`%flink.conf` indicates that this block is configuration statements. For more information about Flink configuration including checkpointing, see Apache Flink Checkpointing.

The input table for the source Amazon Kinesis Data Streams is created with the following Flink ssql code. Note that the `TRADE_TIME` field stores the date/time created by the data generator.

```
%flink.ssql

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

You can view the input stream with this statement:

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

select * from TRADE_SOURCE_STREAM;
```

Before sending the aggregate data to Amazon S3, you can view it directly in Amazon Managed Service for Apache Flink Studio with a tumbling window select query. This aggregates the trading data in a one-minute time window. Note that the %flink.ssql statement must have a (type=update) designation:

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

select TUMBLE_ROWTIME(TRADE_TIME,
INTERVAL '1' MINUTE) as TRADE_WINDOW,
TICKER, COUNT(*) as VOLUME,
AVG(PRICE) as AVG_PRICE,
MIN(PRICE) as MIN_PRICE,
MAX(PRICE) as MAX_PRICE FROM TRADE_SOURCE_STREAMGROUP BY TUMBLE(TRADE_TIME, INTERVAL
 '1' MINUTE), TICKER;
```

You can then create a table for the destination in Amazon S3. You must use a watermark. A watermark is a progress metric that indicates a point in time when you are confident that no more delayed events will arrive. The reason for the watermark is to account for late arrivals. The interval ‘5’ Second allows trades to enter the Amazon Kinesis Data Stream 5 seconds late and still be included if they have a timestamp within the window. For more information, see Generating Watermarks.

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

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

This statement inserts the data into the TRADE_DESTINATION_S3. TUMPLE_ROWTIME is the timestamp of the inclusive upper bound of the tumbling window.

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

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

Let your statement run for 10 to 20 minutes to accumulate some data in Amazon S3. Then abort your statement.

This closes the file in Amazon S3 so that it is viewable.

Here is what the contents looks like:

You can use the CloudFormation template to create the infrastructure.

CloudFormation will create the following resources in your AWS account:

- Amazon Kinesis Data Streams
- Amazon Managed Service for Apache Flink Studio
- AWS Glue database
- Amazon S3 bucket
- IAM roles and policies for Amazon Managed Service for Apache Flink Studio to access appropriate resources

Import the notebook and change the Amazon S3 bucket name with the new Amazon S3 bucket created by CloudFormation.

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

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

See more

Here are some additional resources that you can use to learn more about using Managed Service for Apache Flink Studio:

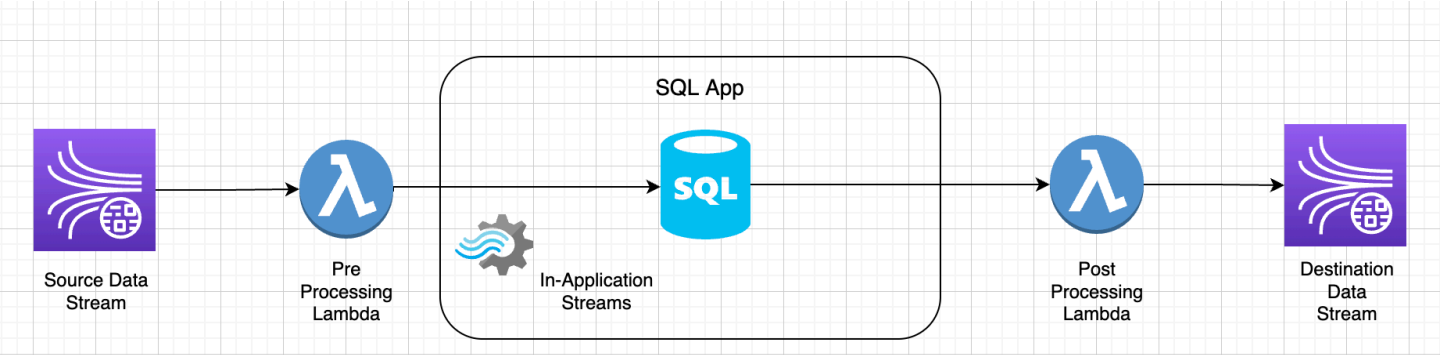
- Managed Service for Apache Flink Studio Notebooks Developers Guide
- Apache Flink 1.13 Documentation
- Managed Service for Apache Flink Studio Workshop
- Apache Flink Windowing
- Amazon Kinesis Data Analytics Developer Guide – Writing from a Kinesis Data Analytics Stream to an S3 Bucket

Leveraging user-defined functions (UDFs)

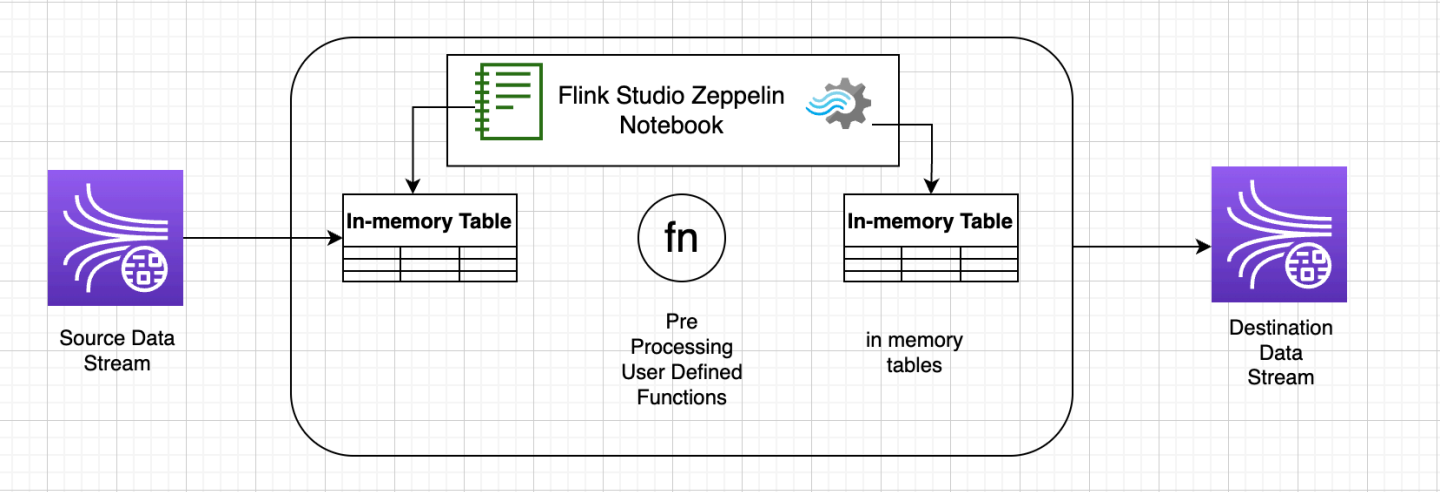
The purpose of the pattern is to demonstrate how to leverage UDFs in Kinesis Data Analytics-Studio Zeppelin notebooks for processing data in the Kinesis stream. Managed Service for Apache Flink Studio uses Apache Flink to provide advanced analytical capabilities, including exactly once processing semantics, event-time windows, extensibility using user defined functions and customer integrations, imperative language support, durable application state, horizontal scaling, support for multiple data sources, extensible integrations, and more. These are critical for ensuring accuracy, completeness, consistency, and reliability of data streams processing and are not available with Amazon Kinesis Data Analytics for SQL.

In this sample application, we will demonstrate how to leverage UDFs in KDA-Studio Zeppelin notebook for processing data in the Kinesis stream. Studio notebooks for Kinesis Data Analytics allows you to interactively query data streams in real time, and easily build and run stream processing applications using standard SQL, Python, and Scala. With a few clicks in the AWS Management Console, you can launch a serverless notebook to query data streams and get results in seconds. For more information, see Using a Studio notebook with Kinesis Data Analytics for Apache Flink.

Lambda functions used for pre/post processing of data in KDA-SQL applications:



User-defined functions for pre/post processing of data using KDA-Studio Zeppelin notebooks



User-defined functions (UDFs)

To reuse common business logic into an operator, it can be useful to reference a user-defined function to transform your data stream. This can be done either within the Managed Service for Apache Flink Studio notebook, or as an externally referenced application jar file. Utilizing User-defined functions can simplify the transformations or data enrichments that you might perform over streaming data.

In your notebook, you will be referencing a simple Java application jar that has functionality to anonymize personal phone numbers. You can also write Python or Scala UDFs for use within the notebook. We chose a Java application jar to highlight the functionality of importing an application jar into a Pyflink notebook.

Environment setup

To follow this guide and interact with your streaming data, you will use an AWS CloudFormation script to launch the following resources:

- Source and target Kinesis Data Streams
- Glue Database
- IAM role
- Managed Service for Apache Flink Studio Application
- Lambda Function to start Managed Service for Apache Flink Studio Application
- Lambda Role to execute the preceding Lambda function
- Custom resource to invoke the Lambda function

Download the CloudFormation template here.

Create the CloudFormation stack

1. Go to the AWS Management Console and choose **CloudFormation** under the list of services.
2. On the **CloudFormation** page, choose **Stacks** and then choose **Create Stack with new resources (standard)**.
3. On the **Create stack** page, choose **Upload a Template File**, and then choose the `kda-flink-udf.yml` that you downloaded previously. Upload the file and then choose **Next**.
4. Give the template a name, such as `kinesis-UDF` so that it is easy to remember, and update input Parameters such as input-stream if you want a different name. Choose **Next**.
5. On the **Configure stack options** page, add **Tags** if you want and then choose **Next**.
6. On the **Review** page, check the boxes allowing for the creation of IAM resources and then choose **Submit**.

The CloudFormation stack may take 10 to 15 minutes to launch depending on the Region you are launching in. Once you see `CREATE_COMPLETE` status for the entire stack, you are ready to continue.

Working with Managed Service for Apache Flink Studio notebook

Studio notebooks for Kinesis Data Analytics allow you to interactively query data streams in real time, and easily build and run stream processing applications using standard SQL, Python, and Scala. With a few clicks in the AWS Management Console, you can launch a serverless notebook to query data streams and get results in seconds.

A notebook is a web-based development environment. With notebooks, you get a simple interactive development experience combined with the advanced data stream processing capabilities provided by Apache Flink. Studio notebooks use notebooks powered by Apache Zeppelin, and uses Apache Flink as the stream processing engine. Studio notebooks seamlessly combine these technologies to make advanced analytics on data streams accessible to developers of all skill sets.

Apache Zeppelin provides your Studio notebooks with a complete suite of analytics tools, including the following:

- Data Visualization
- Exporting data to files
- Controlling the output format for easier analysis

Using the notebook

1. Go to the AWS Management Console and choose **Amazon Kinesis** under the list of services.
2. On the left-hand navigation page, choose **Analytics applications** and then choose **Studio notebooks**.
3. Verify that the **KinesisDataAnalyticsStudio** notebook is running.
4. Choose the notebook and then choose **Open in Apache Zeppelin**.
5. Download the Data Producer Zeppelin Notebook file which you will use to read and load data into the Kinesis Stream.
6. Import the `Data Producer` Zeppelin Notebook. Make sure to modify input `STREAM_NAME` and `REGION` in the notebook code. The input stream name can be found in the CloudFormation stack output.
7. Execute **Data Producer** notebook by choosing the **Run this paragraph** button to insert sample data into the input Kinesis Data Stream.

8. While the sample data loads, download MaskPhoneNumber-Interactive notebook, which will read input data, anonymize phone numbers from the input stream and store anonymized data into the output stream.
9. Import the `MaskPhoneNumber-interactive` Zeppelin notebook.
10. Execute each paragraph in the notebook.

    a. In paragraph 1, you import a User Defined Function to anonymize phone numbers.

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

    b. In the next paragraph, you create an in-memory table to read input stream data. Make sure the stream name and AWS Region are correct.

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

    DROP TABLE IF EXISTS customer_reviews;

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

    c. Check if data is loaded into the in-memory table.

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

    d. Invoke the user defined function to anonymize the phone number.

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

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

e. Now that the phone numbers are masked, create a view with a masked number.

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

DROP VIEW IF EXISTS sentiments_view;

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

f. Verify the data.

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

g. Create in-memory table for the output Kinesis Stream. Make sure stream name and AWS Region are correct.

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

DROP TABLE IF EXISTS customer_reviews_stream_table;

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

h. Insert updated records in the target Kinesis Stream.

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

```
INSERT INTO customer_reviews_stream_table
SELECT customer_id, product, review, phoneNumber
FROM sentiments_view
```

i. View and verify data from the target Kinesis Stream.

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

Promoting a notebook as an application

Now that you have tested your notebook code interactively, you will deploy the code as a streaming application with durable state. You will need to first modify Application configuration to specify a location for your code in Amazon S3.

1. On the AWS Management Console, choose your notebook and in **Deploy as application configuration - optional**, choose **Edit**.
2. Under **Destination for code in Amazon S3**, choose the Amazon S3 bucket that was created by the CloudFormation scripts. The process may take a few minutes.
3. You won't be able to promote the note as is. If you try, you will an error as `Select` statements are not supported. To avert this issue, download the MaskPhoneNumber-Streaming Zeppelin Notebook.
4. Import `MaskPhoneNumber-streaming` Zeppelin Notebook.
5. Open the note and choose **Actions for KinesisDataAnalyticsStudio**.
6. Choose **Build MaskPhoneNumber-Streaming and export to S3**. Make sure to rename **Application Name** and include no special characters.
7. Choose **Build and Export**. This will take few minutes to setup Streaming Application.
8. Once the build is complete, choose **Deploy using AWS console**.
9. On the next page, review settings and make sure to choose the correct IAM role. Next, choose **Create streaming application**.
10. After few minutes, you would see message that the streaming application was created successfully.

For more information on deploying applications with durable state and limits, see Deploying as an application with durable state.

Cleanup

Optionally, you can now uninstall the CloudFormation stack. This will remove all the services which you set up in previously.

What Is Amazon Kinesis Data Analytics for SQL Applications?

With Amazon Kinesis Data Analytics for SQL Applications, 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 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 Data Firehose (Amazon S3, Amazon Redshift, Amazon OpenSearch Service, and Splunk), 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:

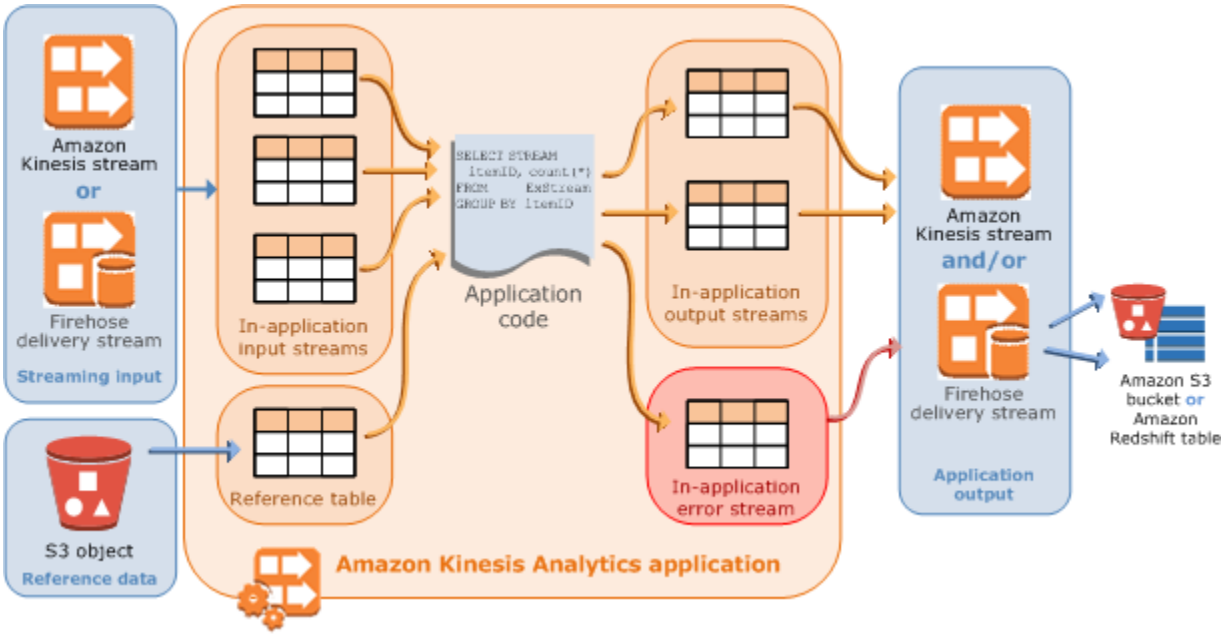
1. **Read the How It Works section of this guide.** This section introduces various Kinesis Data Analytics components that you work with to create an end-to-end experience. For more information, see Amazon Kinesis Data Analytics for SQL Applications: How It Works.
2. **Try the Getting Started exercises.**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

Amazon Kinesis Data Analytics for SQL Applications: How It Works

> **Note**
>
> After September 12, 2023, you will not able to create new applications using Kinesis Data Firehose as a source if you do not already use Kinesis Data Analytics for SQL. For more information, see Limits.

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 Kinesis Data Analytics API. Kinesis Data Analytics provides API operations to manage applications. For a list of API operations, see Actions.

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, add or delete an output configuration, or update application code, Kinesis Data Analytics updates the current application version ID. Kinesis Data Analytics also maintains timestamps 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 Firehose 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 and Configuring Application Input.

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

  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.

- **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. You can also 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. You can then apply additional queries to process these streams. For more information, see Application Code.

- **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 the application output to persist data in the in-application streams that hold your application output (also referred to as in-application output streams) to external destinations. External destinations can be a Firehose delivery stream or a Kinesis data stream. Note the following about these destinations:
  - You can configure a Firehose delivery stream to write results to Amazon S3, Amazon Redshift, or Amazon OpenSearch Service (OpenSearch Service).
  - 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.

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 is 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.

- 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 if you experience an application interruption. For more information, see Delivery Model for Persisting Application Output to an External Destination.

Topics

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 for SQL Applications: How It Works. The sections in this topic describe the application input sources.

Topics

- Using the Schema Discovery Feature on Streaming Data
- Using the Schema Discovery Feature on Static Data
- Preprocessing Data Using a Lambda Function
- Parallelizing Input Streams for Increased Throughput

Configuring a Streaming Source

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

- A Kinesis data stream
- A Firehose delivery stream

> **Note**
>
> After September 12, 2023, you will not able to create new applications using Kinesis Data Firehose as a source if you do not already use Kinesis Data Analytics for SQL. Existing customers using Kinesis Data Analytics for SQL applications with `KinesisFirehoseInput` can continue to add applications with `KinesisFirehoseInput` within an existing account using Kinesis Data Analytics. If you are an existing customer and wish to create a new account with Kinesis Data Analytics for SQL applications with `KinesisFirehoseInput`, you can create a case via the service limit increase form. For more information, see the AWS Support Center. We recommend always testing any new applications before promoting to production.

> **Note**
>
> If the Kinesis data stream is encrypted, Kinesis Data Analytics accesses the data in the encrypted stream seamlessly with no further configuration needed. Kinesis Data Analytics does not store unencrypted data read from Kinesis Data Streams. For more information, see What Is Server-Side Encryption For Kinesis Data Streams?.

Kinesis Data Analytics continuously polls the streaming source for new data and ingests it in in-application streams according to the input configuration.

> **Note**
>
> Adding a Kinesis Stream as your application's input does not affect the data in the stream. If another resource such as a Firehose delivery stream also accessed the same Kinesis stream, both the Firehose delivery stream and the Kinesis Data Analytics application would receive the same data. Throughput and throttling might be affected, however.

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 Kinesis Data Analytics can assume to access the stream on your behalf.
- **In-application stream name prefix** – When you start the application, 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. 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, Kinesis Data Analytics maps the streaming source to one in-application stream named *prefix*_001.

  There is a limit on the rate that you can insert rows in an in-application stream. Therefore, Kinesis Data Analytics supports multiple such in-application streams so that you can bring records into your application at a much faster rate. If you find that 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. You also describe how each record on the stream maps to columns in the in-application stream that is created. This is where you provide column names and data types.

Note

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 same casing (matching lowercase and uppercase letters exactly). For more information about identifiers, see Identifiers in the *Amazon Managed Service for Apache Flink 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 CreateApplication and AddApplicationInput. The following is the input configuration part of the `Createapplication` API request body:

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

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

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 Amazon 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:

```
Ticker, SalePrice, OrderId

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

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

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

You can add an application reference data source either with the API or with the console. Amazon Kinesis Data Analytics provides the following API actions to manage reference data sources:

- AddApplicationReferenceDataSource

- UpdateApplication

For information about adding reference data using the console, see Example: Adding Reference Data to a Kinesis Data Analytics Application.

Note the following:

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

Suppose that you want to refresh the data after Kinesis Data Analytics creates the in-application reference table. Perhaps you updated the Amazon S3 object, or you want to use a different Amazon S3 object. In this case, you can either explicitly call UpdateApplication, or choose **Actions**, **Synchronize reference data table** in the console. 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. If the object size exceeds the limit, 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 Kinesis Data Analytics can assume to read the object on your behalf.
- **In-application reference table name** – 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.

```
{
    "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

> **Note**
>
> After September 12, 2023, you will not able to create new applications using Kinesis Data Firehose as a source if you do not already use Kinesis Data Analytics for SQL. For more information, see Limits.

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.

Amazon Kinesis Data Analytics uses JSONPath expressions in the application's source schema to identify data elements in a streaming source that contains JSON-format data.

For more information about how to map streaming data to your application's input stream, see the section called "Mapping Streaming Source Elements to SQL Input Columns".

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 the following JSON record:

```
{
  "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 `$.customername` 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

You can access the data in a JSON array in the following ways:

- Retrieve all the elements in the array as a single row.
- Retrieve each element in the array as a separate row.

Retrieve All Elements in an Array in a Single Row

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

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

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

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

The preceding expression returns the following from the example JSON record used in this section.

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

Retrieve All Elements in an Array in Separate Rows

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.

23284
63122
77284

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.

John Doe	23284
John Doe	63122
John Doe	77284

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.

```
$.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.

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

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

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

Other Considerations

Additional considerations for working with JSONPath are as follows:

- If no arrays are accessed by an individual element in the JSONPath expressions in the application schema, then a single row is created in the application's input stream for each JSON record processed.
- When an array is flattened (that is, its elements are returned as individual rows), 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

> **Note**
>
> After September 12, 2023, you will not able to create new applications using Kinesis Data Firehose as a source if you do not already use Kinesis Data Analytics for SQL. For more information, see Limits.

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. 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.

For more information about how to configure input streams, see Configuring Application Input.

Mapping JSON Data to SQL Columns

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

- To map elements to columns using the console, see Working with the Schema Editor.
- To map elements to columns using the Kinesis Data Analytics API, see the following section.

To map JSON elements to columns in the in-application input stream, you need a schema with 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 Kinesis Data Analytics API CreateApplication action. To create the in-application stream, specify a schema to transform your data into a schematized version used in SQL. The CreateApplication 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 object in the SourceSchema `RecordColumns` array. The RecordColumn object has the following schema:

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

The fields in the RecordColumn 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": "$.CHANGE"
        },
        {
            "SqlType": "DECIMAL(5,2)",
            "Name": "PRICE",
            "Mapping": "$.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 about 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.
- 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.

Related Topics

- Configuring Application Input
- Data Types
- Working with the Schema Editor
- CreateApplication
- RecordColumn
- SourceSchema

Using the Schema Discovery Feature on Streaming Data

> **Note**
>
> After September 12, 2023, you will not able to create new applications using Kinesis Data Firehose as a source if you do not already use Kinesis Data Analytics for SQL. For more information, see Limits.

Providing an input schema that describes how records on the streaming input map to an in-application stream can be cumbersome and error prone. You can use the DiscoverInputSchema 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.

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, 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: Transforming DateTime Values.

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

Source Stream Column Name	Discovery-Suggested Column Name
USER	COL_USER
USER@DOMAIN	COL_USERDOMAIN
@@	COL_0

Schema Discovery Issues

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

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

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 examples, see Example: Transforming DateTime Values.

Using the Schema Discovery Feature on Static Data

> **Note**
>
> After September 12, 2023, you will not able to create new applications using Kinesis Data Firehose as a source if you do not already use Kinesis Data Analytics for SQL. For more information, see Limits.

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. Suppose that 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 either the console, or the DiscoverInputSchema API with the `S3Configuration` parameter specified.

Running Schema Discovery Using the Console

To run discovery on a static file using the console, do the following:

1. Add a reference data object to an S3 bucket.
2. Choose **Connect reference data** in the application's main page in the Kinesis Data Analytics console.
3. Provide the bucket, path and IAM role data for accessing the Amazon S3 object containing the reference data.
4. Choose **Discover schema**.

For more information on how to add reference data and discover schema in the console, see Example: Adding Reference Data to a Kinesis Data Analytics Application.

Running Schema Discovery Using the API

To run discovery on a static file using the API, 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 Amazon 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.

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) in the Getting Started section.
2. Create a file named `data.csv` with the following contents:

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

3. Sign in to the Amazon S3 console at https://console.aws.amazon.com/s3/.
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/. 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 Amazon 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 looks 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
\r\n2001,1,AK,TotalElectricPowerIndustry,Coal,ShortTons,47615\r
\n2001,1,AK,ElectricGeneratorsElectricUtilities,Coal,ShortTons,16535\r
\n2001,1,AK,CombinedHeatandPowerElectricPower,Coal,ShortTons,22890\r
\n2001,1,AL,TotalElectricPowerIndustry,Coal,ShortTons,3020601\r
\n2001,1,AL,ElectricGeneratorsElectricUtilities,Coal,ShortTons,2987681"
    ],
    "ParsedInputRecords": [
        [
            null,
```

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

```
            "2001",
            "1",
            "AL",
            "ElectricGeneratorsElectricUtilities",
            "Coal",
            "ShortTons",
            "2987681"
        ]
    ]
}
```

Preprocessing Data Using a Lambda Function

> **Note**
>
> After September 12, 2023, you will not able to create new applications using Kinesis Data Firehose as a source if you do not already use Kinesis Data Analytics for SQL. For more information, see Limits.

If the data in your stream needs format conversion, transformation, enrichment, or filtering, you can preprocess the data using an AWS Lambda function. You can do this 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 Amazon 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. Sign in to the AWS Management Console and open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. On the **Connect to a Source** page for your application, choose **Enabled** in the **Record pre-processing with AWS Lambda** section.
3. To use a Lambda function that you have already created, choose the function in the **Lambda function** drop-down list.
4. To create a new Lambda function from one of the Lambda preprocessing templates, choose the template from the drop-down list. Then choose **View <template name> in Lambda** to edit the function.
5. 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 in the *AWS Lambda Developer Guide*.
6. 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:

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

Lambda Preprocessing Metrics

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

Using AWS Lambda with the Kinesis Producer Library

The Kinesis Producer Library (KPL) aggregates small user-formatted records into larger records up to 1 MB to make better use of Amazon Kinesis Data Streams throughput. The Kinesis Client Library (KCL) for Java supports deaggregating these records. However, you must use a special module to deaggregate the records when you use AWS Lambda as the consumer of your streams.

To get the necessary project code and instructions, see the Kinesis Producer Library Deaggregation Modules for AWS Lambda on GitHub. You can use the components in this project to process KPL serialized data within AWS Lambda in Java, Node.js, and Python. You can also use these components as part of a multi-lang KCL application.

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 data stream or 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 data stream or a Firehose delivery stream.

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

Kinesis Data Firehose Request Data Model

Field	Description
`invocationId`	The Lambda invocation Id (random GUID).

Field	Description
applicationArn	Kinesis Data Analytics application Amazon Resource Name (ARN)
streamArn	Delivery stream ARN
records	

Field	Description	
recordId	record ID (random GUID)	
kinesisFirehoseRecordMetadata	<table><tr><th>Field</th><th>Description</th></tr><tr><td>approximateArrivalTimestamp</td><td>Delivery stream record approximate arrival time</td></tr></table>	
data	Base64-encoded source record payload	

The following example shows input from a Firehose delivery stream:

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

```
}
```

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

Kinesis Streams Request Data Model

Field	Description
`invocationId`	The Lambda invocation Id (random GUID).
`applicationArn`	Kinesis Data Analytics application ARN
`streamArn`	Delivery stream ARN
records	

Field	Description
`recordId`	record ID based off of Kinesis record sequence number
`kinesisStreamRecordMetadata`	

Field	Description
`sequenceNumber`	Sequence number from the Kinesis stream record
`partitionKey`	Partition key from the Kinesis stream record
`shardId`	`ShardId` from the Kinesis stream record
`approximateArrival`	Delivery stream record approximate arrival time

Field	Description
Field	Description
Field	Description
Timestan	
data	Base64-encoded source record payload

The following example shows input from a Kinesis data stream:

```
{
  "invocationId": "00540a87-5050-496a-84e4-e7d92bbaf5e2",
  "applicationArn": "arn:aws:kinesisanalytics:us-east-1:12345678911:application/lambda-
test",
  "streamArn": "arn:aws:kinesis:us-east-1:AAAAAAAAAAAA:stream/lambda-test",
  "records": [
    {
      "recordId": "49572672223665514422805246926656954630972486059535892482",
      "data": "aGVsbG8gd29ybGQ=",
      "kinesisStreamRecordMetadata":{
            "shardId" :"shardId-000000000003",
            "partitionKey":"7400791606",

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

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

records

Field	Description
`recordId`	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.
`result`	The status of the data transformation of the record. The possible values are: • Ok: The record was transformed successfully. Kinesis Data Analytics ingests the record for SQL processing. • `Dropped`: 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. • `ProcessingFailed` : The record could not be transform ed. 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. The data payload field is optional for a `ProcessingFailed` record.
`data`	The transformed data payload, after base64-encoding. Each data payload can contain multiple JSON documents if the application ingestion data format is JSON. Or each 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.

The following example shows output from a Lambda function:

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

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.

For data preprocessing failures, Kinesis Data Analytics 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.

Creating Lambda Functions for Preprocessing

Your Amazon Kinesis Data Analytics application can use Lambda functions for preprocessing records as they are ingested into the application. Kinesis Data Analytics provides the following templates on the console to use as a starting point for preprocessing your data.

Topics

Creating a Preprocessing Lambda Function in Node.js

The following templates for creating preprocessing Lambda function in Node.js are available on the Kinesis Data Analytics console:

Lambda Blueprint	Language and version	Description
General Kinesis Data Analytics Input Processing	Node.js 6.10	A Kinesis Data Analytics record preprocessor 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.
Compressed Input Processing	Node.js 6.10	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 processin g status.

Creating a Preprocessing Lambda Function in Python

The following templates for creating preprocessing Lambda function in Python are available on the console:

Lambda Blueprint	Language and version	Description
General Kinesis Analytics Input Processing	Python 2.7	A Kinesis Data Analytics record preprocessor 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.
KPL Input Processing	Python 2.7	A Kinesis Data Analytics record processor that receives Kinesis Producer Library (KPL) aggregates of JSON or CSV records as input and returns disaggregated records with a processing status.

Creating a Preprocessing Lambda Function in Java

To create a Lambda function in Java for preprocessing records, use the Java events classes.

The following code demonstrates a sample Lambda function that preprocesses records using Java:

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

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

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

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

```
            context.getLogger().log("record aat is :" +
 record.kinesisStreamRecordMetadata.approximateArrivalTimestamp);
             // Add your record.data pre-processing logic here.

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

}
```

Creating a Preprocessing Lambda Function in .NET

To create a Lambda function in .NET for preprocessing records, use the .NET events classes.

The following code demonstrates a sample Lambda function that preprocesses records using C#:

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

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

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

```
                // Add your record preprocessig logic here.

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

For more information about creating Lambda functions for preprocessing and destinations in .NET, see [Amazon.Lambda.KinesisAnalyticsEvents].

Parallelizing Input Streams for Increased Throughput

> **Note**
>
> After September 12, 2023, you will not able to create new applications using Kinesis Data Firehose as a source if you do not already use Kinesis Data Analytics for SQL. For more information, see Limits.

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 for SQL Applications: How It Works.

In almost all cases, Amazon Kinesis Data Analytics scales your application to handle the capacity of the Kinesis streams or 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 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 about accessing multiple in-application streams in your code, see Accessing Separate In-Application Streams in Your Amazon Kinesis Data Analytics Application.

Although Kinesis Data Streams and 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 stream include a `shard_id` field that can be used to identify the source shard for the record.
- The records from a Firehose delivery stream don't include a field that identifies the record's source shard or partition. This is because 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 `InputBytes` and `MillisBehindLatest` metrics in Amazon CloudWatch.

If the `InputBytes` metric is greater that 100 MB/sec (or you anticipate that it will be greater than this rate), this can cause an increase in `MillisBehindLatest` and increase the impact of application issues. To address this, we recommend making the following language choices for your application:

- Use multiple streams and Kinesis Data Analytics for SQL applications if your application has scaling needs beyond 100 MB/second.
- Use Kinesis Data Analytics for Java Applications if you want to use a single stream and application.

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. You set this number after an application is created using UpdateApplication.

> **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 AWS CLI, see Step 2: Set Up the AWS Command Line Interface (AWS CLI).

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 about `CreateApplication`, see CreateApplication.

```
{
   "ApplicationCode": "<The SQL code the new application will run on the input
 stream>",
   "ApplicationDescription": "<A friendly description for the new application>",
   "ApplicationName": "<The name for the new application>",
   "Inputs": [
```

```
    {
      "InputId": "ID for the new input stream",
      "InputParallelism": {
        "Count": 2
    }],
   "Outputs": [ ... ],
 }]
}
```

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 about `Update_Application`, see UpdateApplication.

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

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

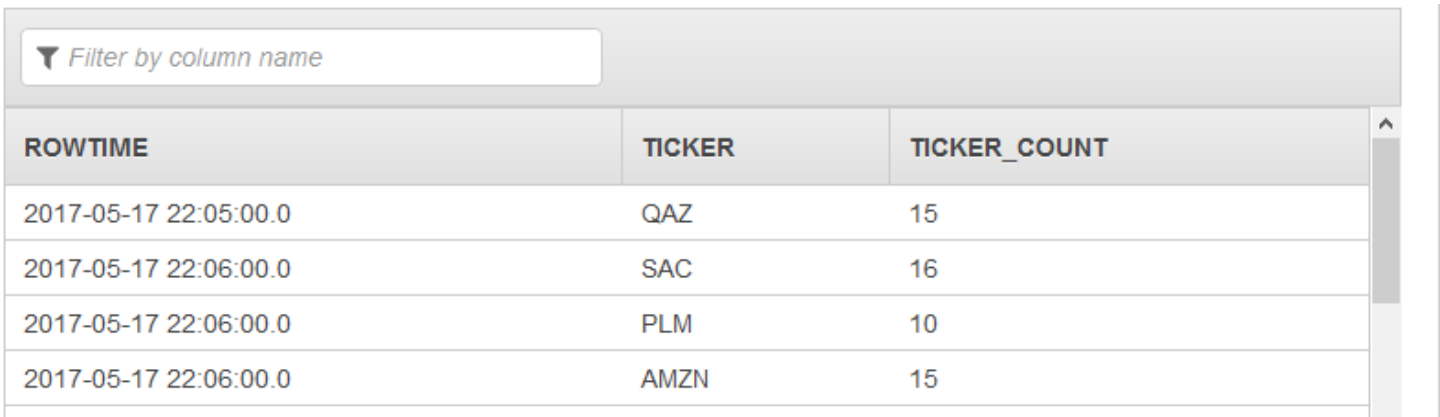
To use multiple in-application input streams in your application, you must explicitly select from the different streams. The following code example demonstrates how to query multiple input streams in the application created in the Getting Started tutorial.

In the following example, each source stream is first aggregated using COUNT before being combined into a single in-application stream called `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.

> **Note**
>
> To run this example and get results from both in-application input streams, update both the number of shards in your source stream and the `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 `in_application_stream001` similar to the following:



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.

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 for SQL Applications: How It Works.

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:

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

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

Note

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 Managed Service for Apache Flink 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.

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 a Kinesis Data Analytics Application.

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
- 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 an 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 Firehose 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.

> **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.

  Kinesis Data Analytics looks for the in-application stream that you specified in the output configuration. (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 Firehose delivery stream, or a Lambda function. You provide the Amazon Resource Name (ARN) of the stream or function. You also provide an IAM role that Kinesis Data Analytics can assume to write to the stream or function on your behalf. You describe the record format (JSON, CSV) to Kinesis Data Analytics to use when writing to the external destination.

If 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 issue is not resolved, your application eventually stops processing new data. You can monitor Kinesis Data 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.

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

```
        "Name": "string"
    }

]
```

Creating a Firehose Delivery Stream Output

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

```
"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.

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

Using a Lambda Function as Output

Using AWS 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
- Amazon 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.

Topics

- Lambda as Output Event Input Data Model and Record Response Model
- Lambda Output Invocation Frequency
- Adding a Lambda Function for Use as an Output
- Common Lambda as Output Failures
- Creating Lambda Functions for Application Destinations

Lambda as Output Permissions

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

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

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 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 an 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).

Field	Description
`invocationId`	The Lambda invocation ID (random GUID).
`applicationArn`	The Kinesis Data Analytics application Amazon Resource Name (ARN).

records

Field	Description
`recordId`	record ID (random GUID)
`lambdaDeliveryRecordMetadata`	<table><tr><th>Field</th><th>Description</th></tr><tr><td>`retryHir`</td><td>Number of delivery retries</td></tr></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 an output function (with record IDs) must be acknowledged with either `Ok` or `DeliveryFailed`, and it must contain the following parameters. Otherwise, Kinesis Data Analytics treats them as a delivery failure.

records

Field	Description
`recordId`	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.
`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 data 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, the Lambda function is invoked once every 60 seconds.
- If records are emitted to the destination in-application stream within the application as a continuous query or a sliding window, the Lambda destination function is invoked about once per second.

> **Note**
>
> Per-Lambda function invoke request payload size limits apply. Exceeding those limits results 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 a Kinesis Data Analytics application.

1. Sign in to the AWS Management Console and open the Managed Service for Apache Flink 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 and version, or choose **Create new**.
6. If you are creating a new Lambda function, do the following:
   a. Choose one of the templates provided. For more information, Creating Lambda Functions for Application Destinations.
   b. The **Create Function** page opens in a new browser tab. In the **Name** box, 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. On the Kinesis Data Analytics console, in the **Lambda function** list, choose the Lambda function that you just created. Choose **$LATEST** for the Lambda function version.
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.

For 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.

Creating Lambda Functions for Application Destinations

Your Kinesis Data Analytics application can use AWS Lambda functions as an output. Kinesis Data Analytics provides templates for creating Lambda functions to use as a destination for your applications. Use these templates as a starting point for post-processing output from your application.

Topics

- Creating a Lambda Function Destination in Node.js
- Creating a Lambda Function Destination in Python

- Creating a Lambda Function Destination in Java
- Creating a Lambda Function Destination in .NET

Creating a Lambda Function Destination in Node.js

The following template for creating a destination Lambda function in Node.js is available on the console:

Lambda as Output Blueprint	Language and Version	Description
`kinesis-analytics-output`	Node.js 12.x	Deliver output records from a Kinesis Data Analytics application to a custom destination.

Creating a Lambda Function Destination in Python

The following templates for creating a destination Lambda function in Python are available on the console:

Lambda as Output Blueprint	Language and Version	Description
`kinesis-analytics-output-sns`	Python 2.7	Deliver output records from a Kinesis Data Analytics application to Amazon SNS.
`kinesis-analytics-output-ddb`	Python 2.7	Deliver output records from a Kinesis Data Analytics application to Amazon DynamoDB.

Creating a Lambda Function Destination in Java

To create a destination Lambda function in Java, use the Java events classes.

The following code demonstrates a sample destination Lambda function using Java:

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

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

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

        event.records.stream().forEach(record -> {
            context.getLogger().log("recordId is : " + record.recordId);
            context.getLogger().log("record retryHint is :" +
 record.lambdaDeliveryRecordMetadata.retryHint);
            // Add logic here to transform and send the record to final destination of
 your choice.
            response.records.add(new Record(record.recordId,
 KinesisAnalyticsOutputDeliveryResponse.Result.Ok));
        });
        return response;
    }

}
```

Creating a Lambda Function Destination in .NET

To create a destination Lambda function in .NET, use the .NET events classes.

The following code demonstrates a sample destination Lambda function using C#:

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

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

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

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

                var deliveredRecord = new KinesisAnalyticsOutputDeliveryResponse.Record
                {
                    RecordId = record.RecordId,
                    Result = KinesisAnalyticsOutputDeliveryResponse.OK
                };
                response.Records.Add(deliveredRecord);
            }
            return response;
        }
    }
```

For more information about creating Lambda functions for pre-processing and destinations in .NET, see [Amazon.Lambda.KinesisAnalyticsEvents].

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, Kinesis Data Analytics takes internal checkpoints. These checkpoints are points in time when output records have been delivered to the destinations without 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. Kinesis Data Analytics writes the output to the configured destinations, such as a Kinesis data stream or a Firehose delivery stream. However, your application can be interrupted occasionally, for example:

- You choose to stop your application and restart it later.
- You delete the IAM role that Kinesis Data Analytics needs to write your application output to the configured destination. Without the IAM role, Kinesis Data Analytics doesn't have any permissions to write to the external destination on your behalf.
- A network outage or other internal service failure causes your application to stop running momentarily.

When your application restarts, Kinesis Data Analytics ensures that it continues to process and write output from a point before or equal to when the failure occurred. This helps ensure that it doesn't miss delivering any application output to the configured destinations.

Suppose that 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 must 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. 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.

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

> **Note**
>
> Your Kinesis Data Analytics application can't access or modify the error stream programmatically because the error stream is created using the system account. You must use the error output to determine what errors your application might encounter. You then write your application's SQL code to handle anticipated error conditions.

Error Stream Schema

The error stream has the following schema:

Field	*Data Type*	*Notes*
ERROR_TIME	TIMESTAMP	The time when the error occurred
ERROR_LEVEL	VARCHAR(10)	
ERROR_NAME	VARCHAR(32)	
MESSAGE	VARCHAR(4096)	
DATA_ROWTIME	TIMESTAMP	The row time of the incoming record
DATA_ROW	VARCHAR(49152)	The hex-encoded data in the original row. You can use standard libraries to hex decode this value, or use web

		resources such as this Hex to String Converter.
PUMP_NAME	VARCHAR(128)	The originating pump, as defined with CREATE PUMP

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 64. For instructions on how to request an increase to this limit, see **To request a limit increase** in Amazon Service Limits.

Using Tagging

This section describes how to add key-value metadata tags to Kinesis Data Analytics applications. These tags can be used for the following purposes:

- Determining billing for individual Kinesis Data Analytics applications. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management Guide*.
- Controlling access to application resources based on tags. For more information, see Controlling Access Using Tags in the *User Guide*.
- User-defined purposes. You can define application functionality based on the presence of user tags.

Note the following information about tagging:

- The maximum number of application tags includes system tags. The maximum number of user-defined application tags is 50.
- If an action includes a tag list that has duplicate `Key` values, the service throws an `InvalidArgumentException`.

This topic contains the following sections:

Adding Tags when an Application is Created

You add tags when creating an application using the `tags` parameter of the CreateApplication action.

The following example request shows the `Tags` node for a `CreateApplication` request:

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

Adding or Updating Tags for an Existing Application

You add tags to an application using the TagResource action. You cannot add tags to an application using the UpdateApplication action.

To update an existing tag, add a tag with the same key of the existing tag.

The following example request for the `TagResource` action adds new tags or updates existing tags:

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

```
      {
         "Key": "ExistingKeyOfTagToUpdate",
         "Value": "NewValueForExistingTag"
      }
   ]
}
```

Listing Tags for an Application

To list existing tags, you use the ListTagsForResource action.

The following example request for the `ListTagsForResource` action lists tags for an application:

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

Removing Tags from an Application

To remove tags from an application, you use the UntagResource action.

The following example request for the `UntagResource` action removes tags from an application:

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

Getting Started with Amazon Kinesis Data Analytics for SQL Applications

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

Topics

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Step 1: Set Up an 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
2. Create an IAM User

Sign Up for AWS

When you sign up for Amazon Web Services, 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://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

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 with IAM
- IAM User Guide

Next Step

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

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

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, and then set up the AWS CLI later when you need it.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:
   - Getting Set Up with the AWS Command Line Interface
   - Configuring the AWS Command Line Interface
2. Add a named profile for the administrator user in the AWS CLI config file. You use this profile when executing the AWS CLI commands. For more information about named profiles, see Named Profiles in the *AWS Command Line Interface User Guide*.

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

   For a list of available AWS Regions, see Regions and Endpoints in the *Amazon Web Services General Reference*.
3. Verify the setup by entering the following help command at the command prompt:

   ```
   aws help
   ```

Next Step

Step 3: Create Your Starter Amazon Kinesis Data Analytics Application

Step 3: Create Your Starter Amazon Kinesis Data Analytics Application

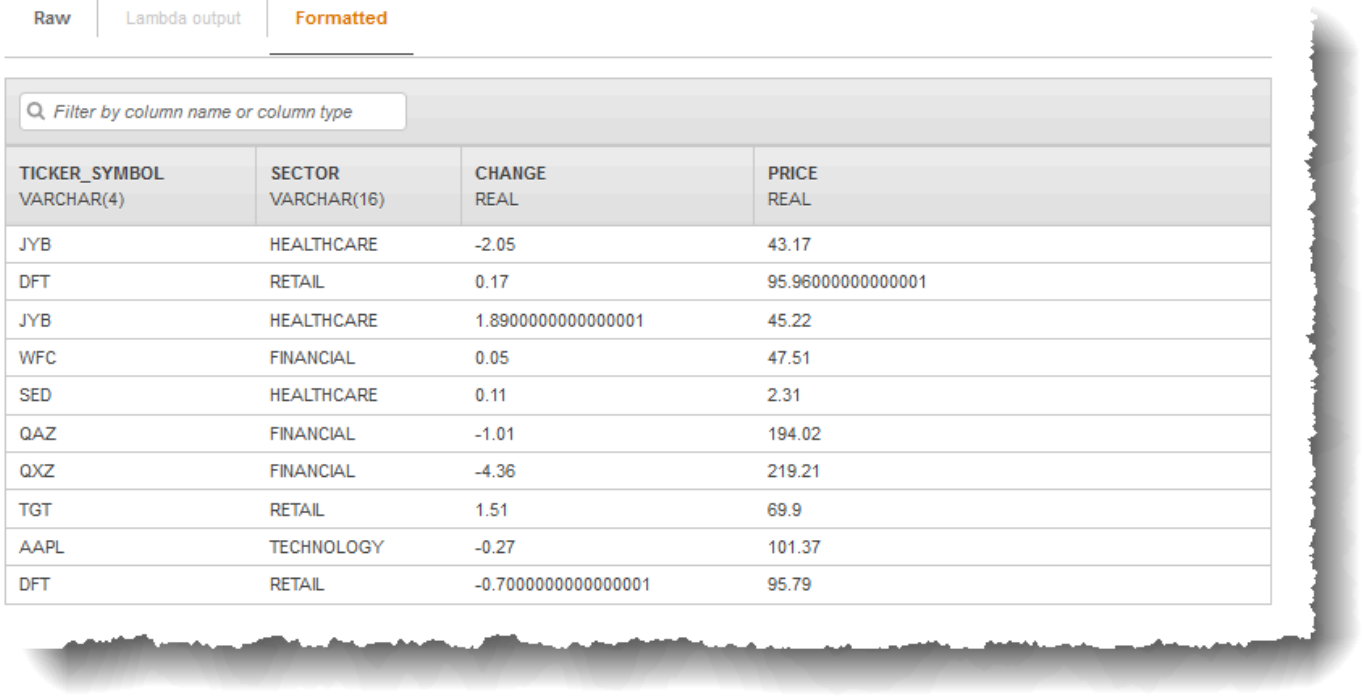
By following the steps in this section, you can create your first Kinesis Data Analytics application using the console.

> **Note**
>
> We suggest that you review Amazon Kinesis Data Analytics for SQL Applications: How It Works 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. 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:

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

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

   The application code queries the in-application stream SOURCE_SQL_STREAM_001. 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.

   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: Creating Simple Alerts.

> **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

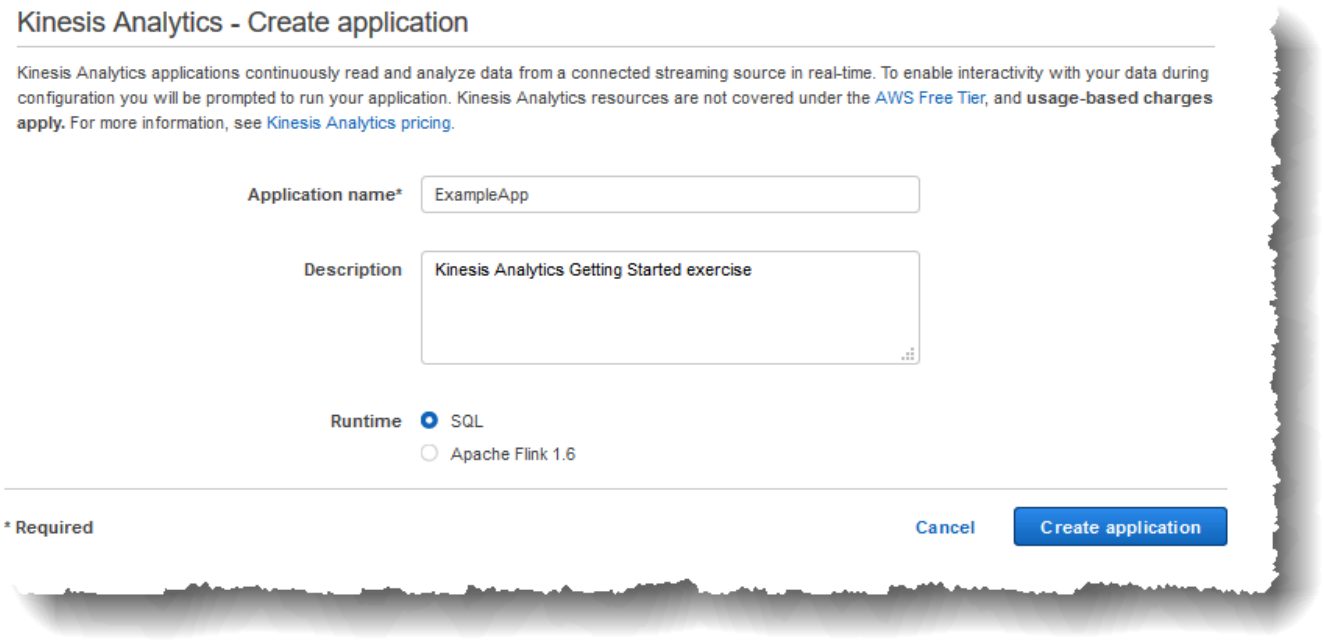
Step 3.1: Create an Application

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 Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**.
3. On the **Create application** page, type an application name, type a description, choose **SQL** for the application's **Runtime** setting, and then choose **Create application**.



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

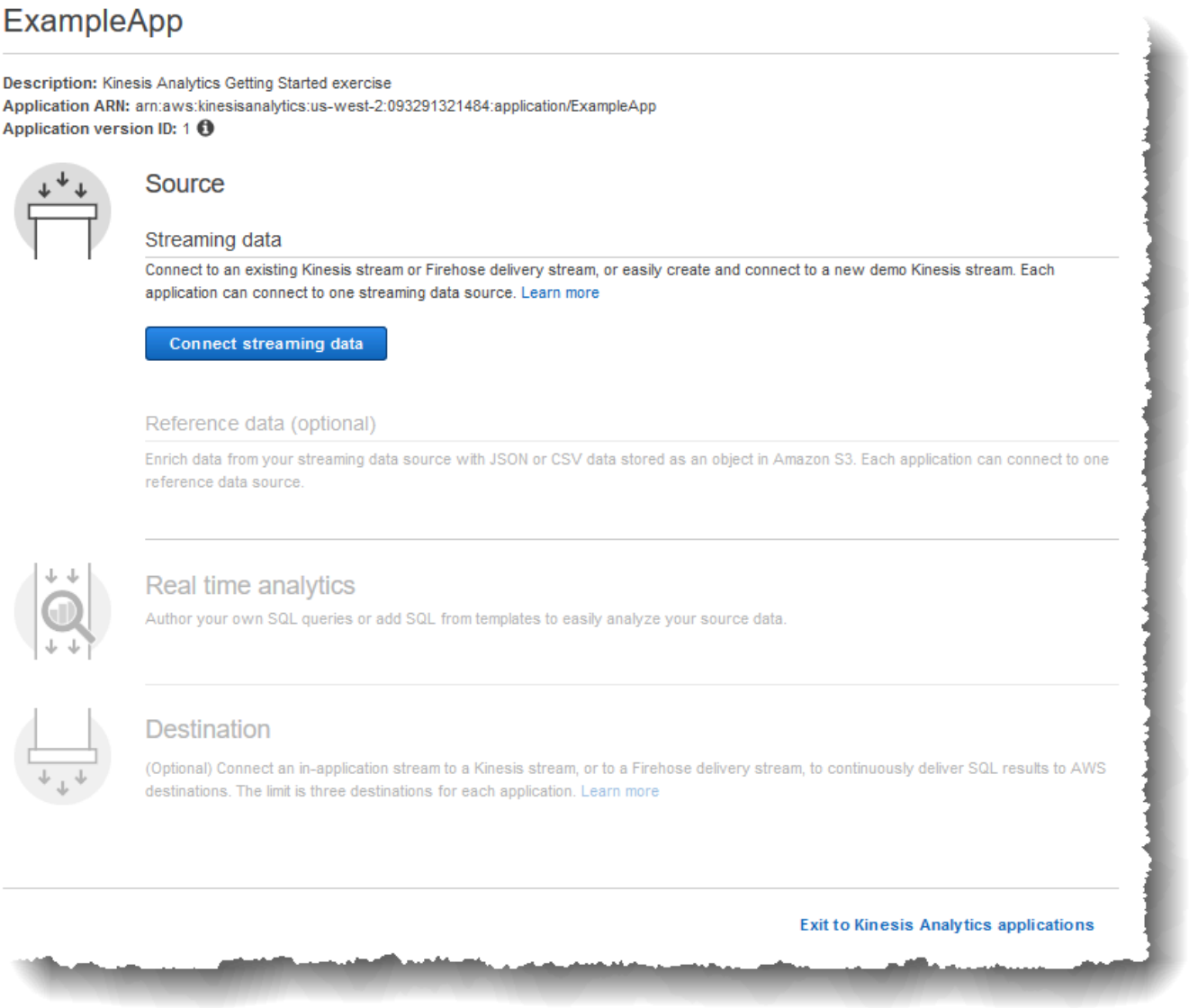
Step 3.2: Configure Input

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 streaming data**.



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 Preprocessing Data Using a Lambda Function.

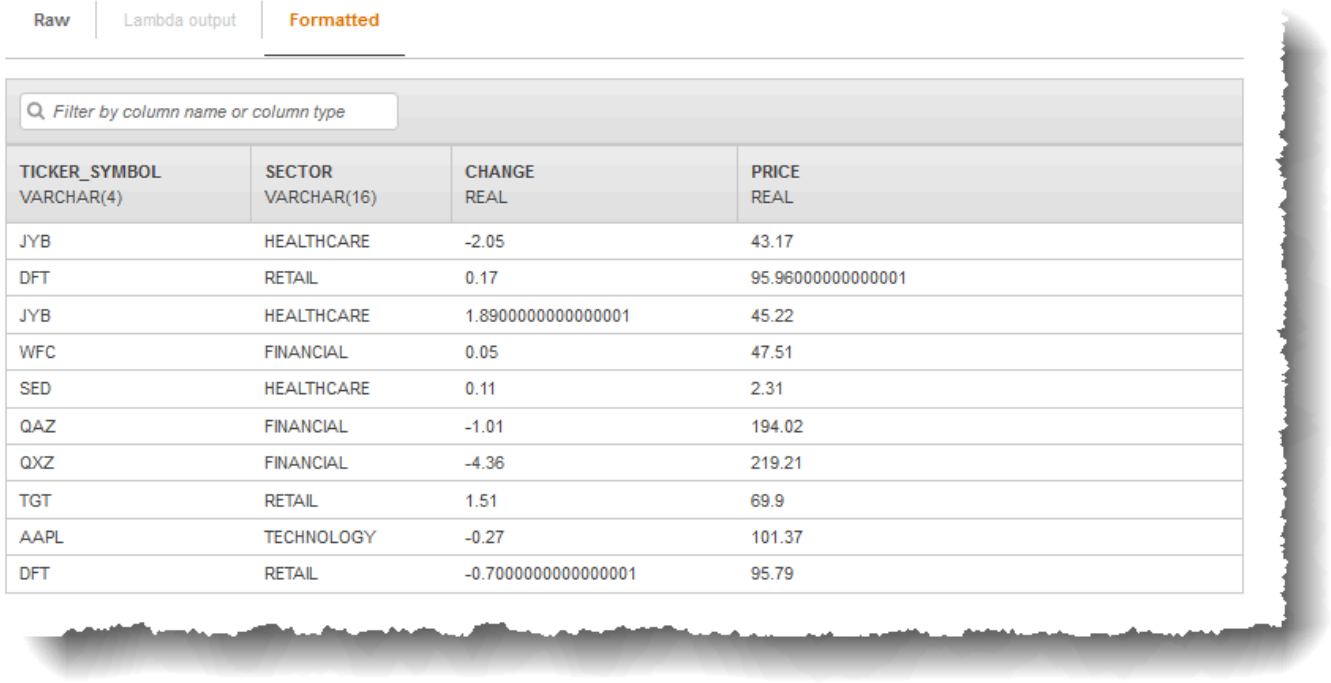
After you provide all the information on this page, the console sends an update request (see UpdateApplication) to add the input configuration the application.

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 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.
- 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 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.

  If you choose **Rediscover schema**, you can request the console to run DiscoverInputSchema 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.

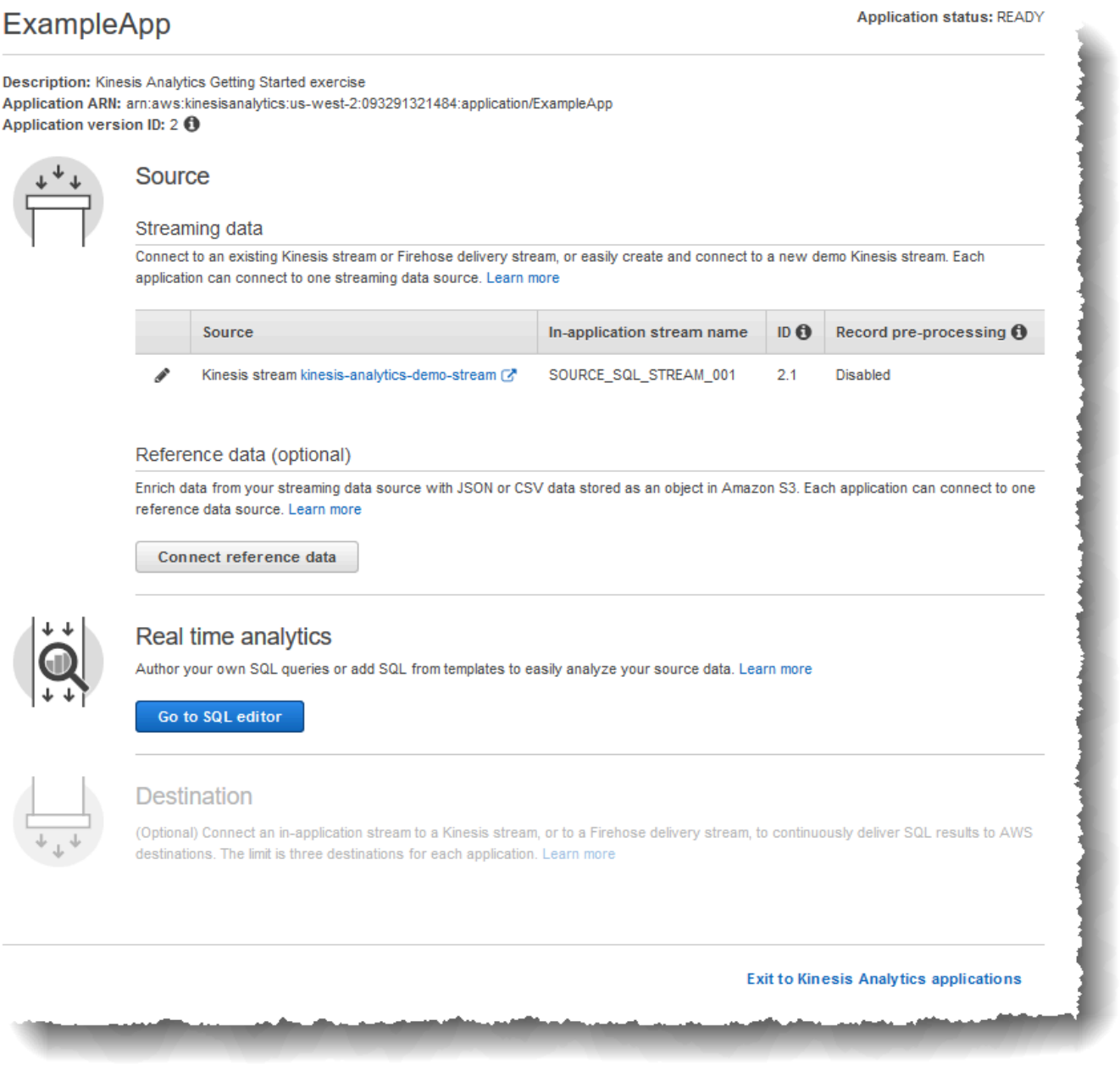
Next Step

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

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 "ExampleApp"?** dialog box, choose **Yes, start application**.

   The console sends a request to start the application (see StartApplication), 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:

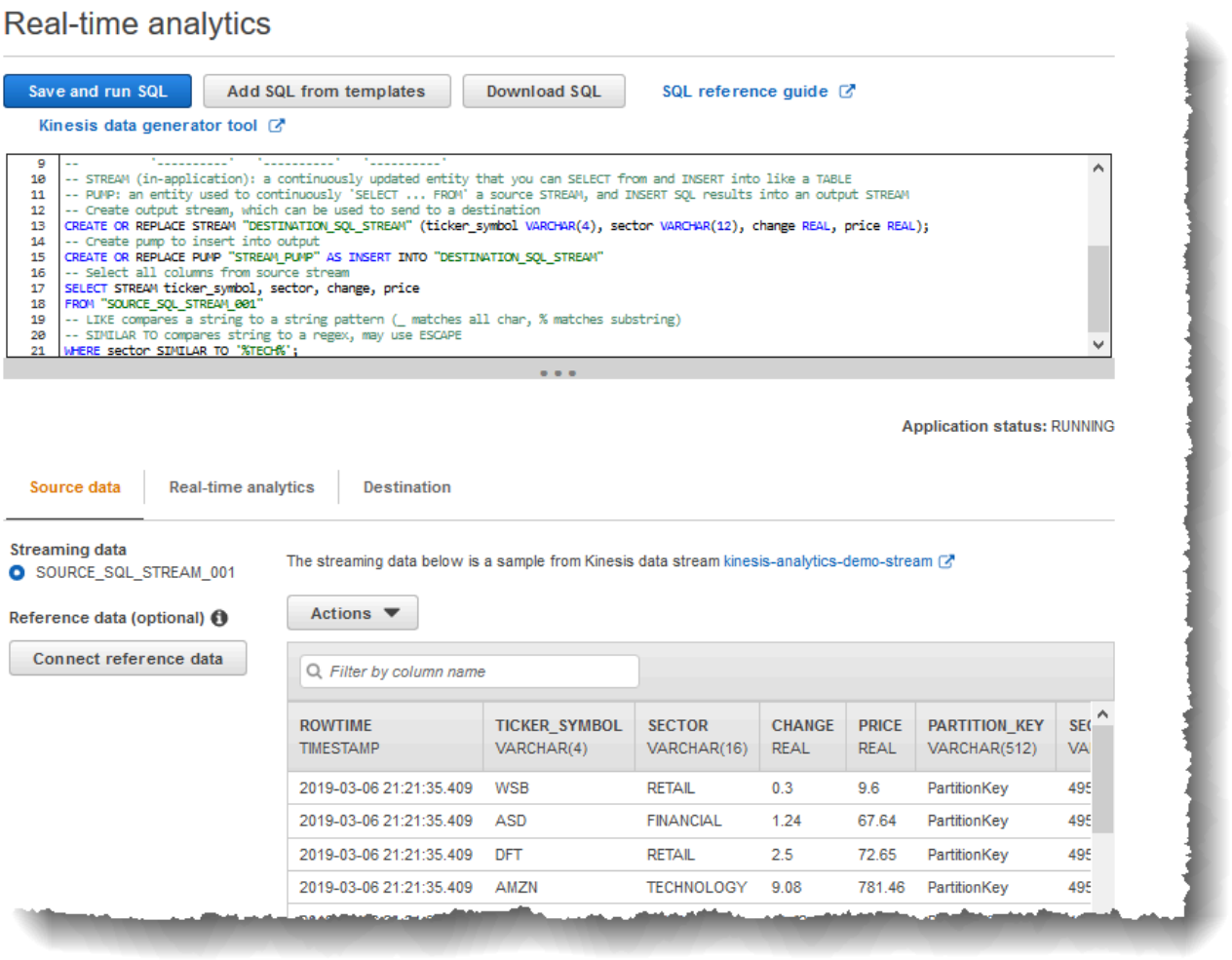
   - It creates the in-application stream `DESTINATION_SQL_STREAM`.
   - It creates a pump `STREAM_PUMP`, and uses it to select rows from `SOURCE_SQL_STREAM_001` and insert them in the `DESTINATION_SQL_STREAM`.

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 timestamp columns:

  - **ROWTIME** – Each row in an in-application stream has a special column called `ROWTIME`. This column is the timestamp 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 timestamp 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 timestamp 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` timestamp 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

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:

```
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 in to these separate streams.

   Append the following SQL statements to your application code:

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

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

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

CREATE OR REPLACE PUMP "TGT_PUMP" AS
```

```
INSERT INTO "TGT_STREAM"
   SELECT STREAM ticker_symbol, change, price
   FROM   "SOURCE_SQL_STREAM_001"
   WHERE  ticker_symbol SIMILAR TO '%TGT%';
```

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

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.

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

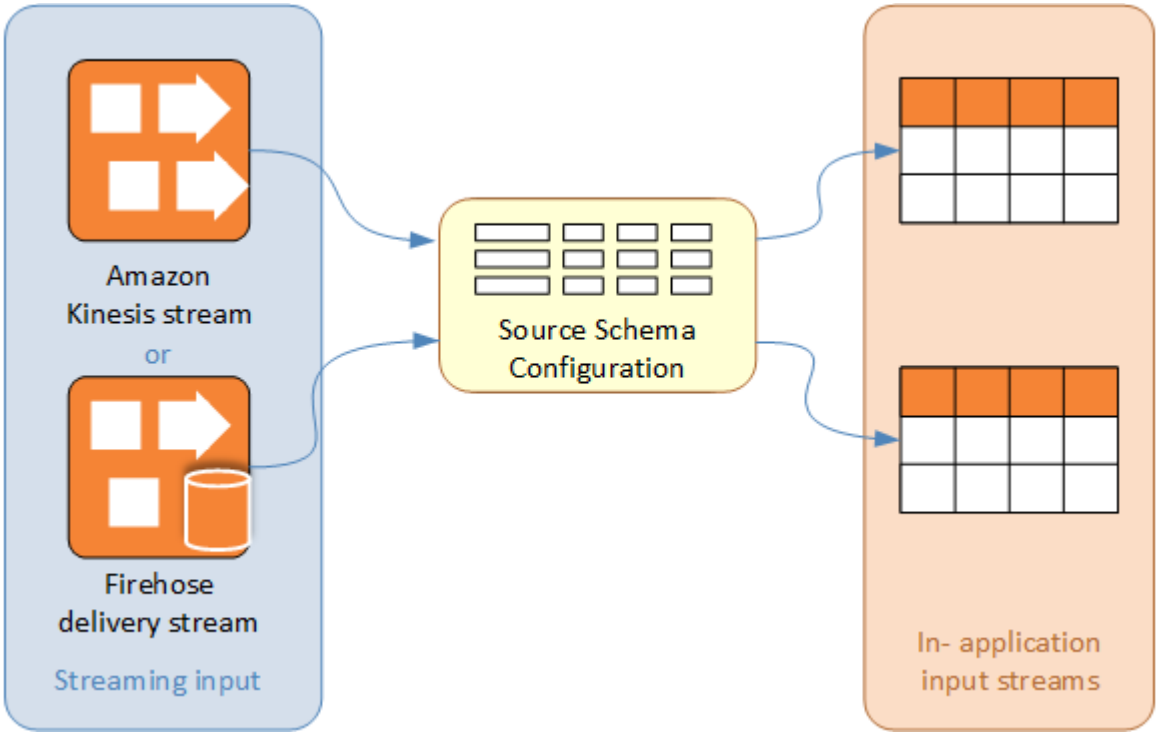
Following, you can find information about how to edit an inferred schema and how to edit SQL code for Amazon Kinesis Data Analytics. You do so by working with the schema editor and SQL editor that are part of the Kinesis Data Analytics console.

> **Note**
>
> To access or sample data in the console, your login user's role must have the `kinesisanalytics:GetApplicationState` permission. For more information about Kinesis Data Analytics application permissions, see Overview of Managing Access.

Topics

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:

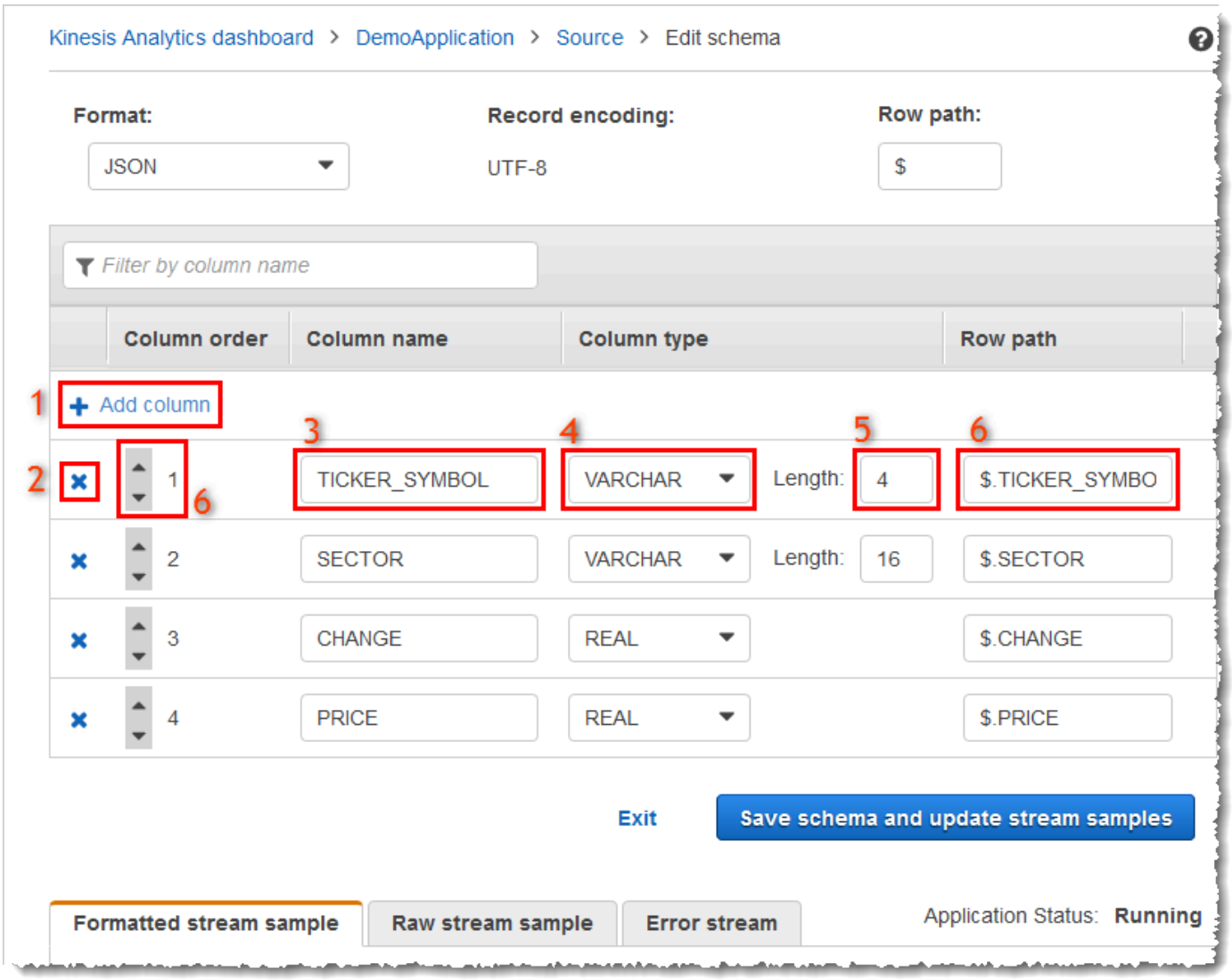
- A JSONPath expression for JSON input streams. JSONPath is a tool for querying JSON data.

- 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. 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:

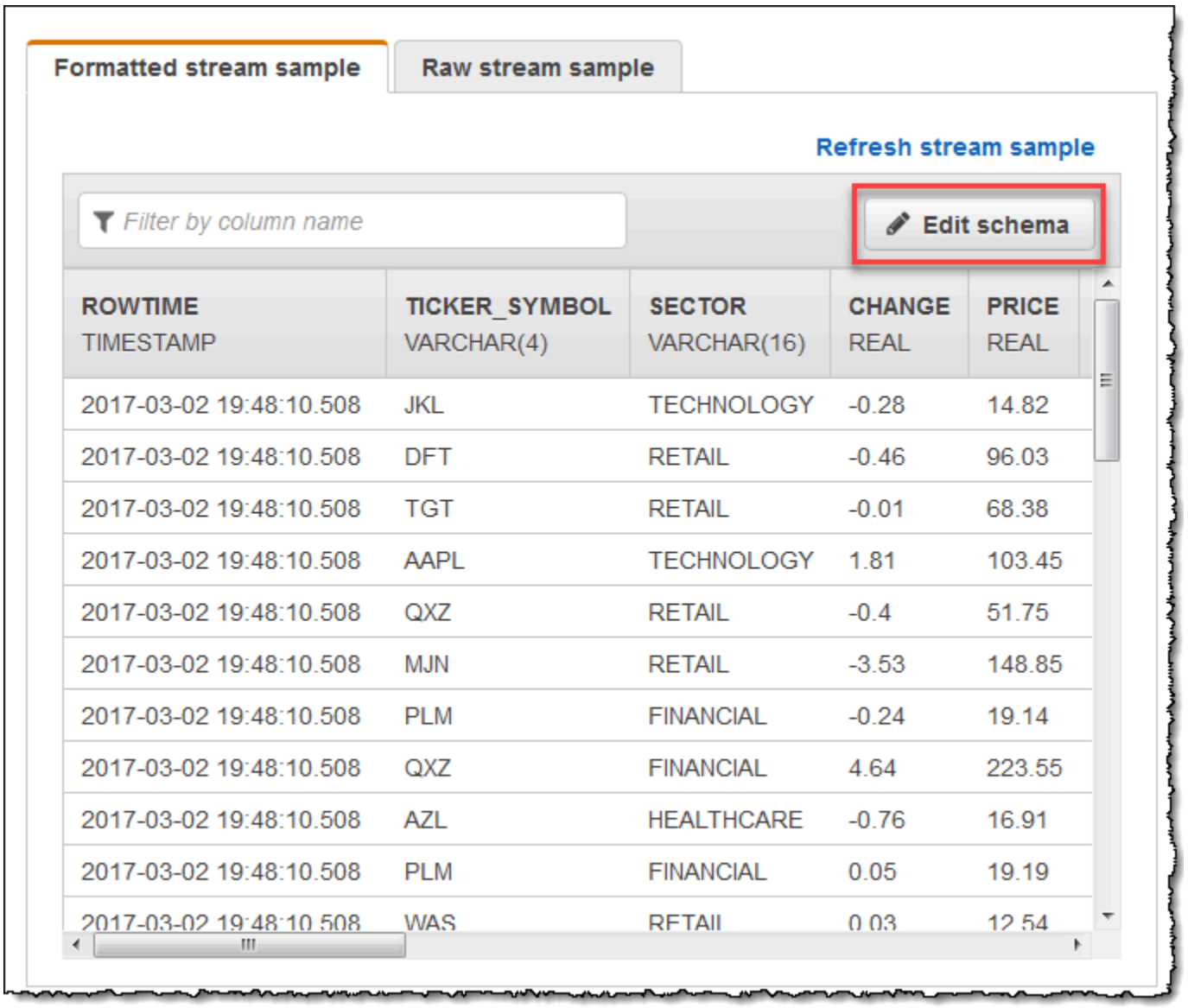
- Add a column (1): You might need to add a data column if a data item is not detected automatically.
- Delete a column (2): You can exclude data from the source stream if your application doesn't require it. This exclusion doesn't affect the data in the source stream. If data is excluded, that data 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, enter 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

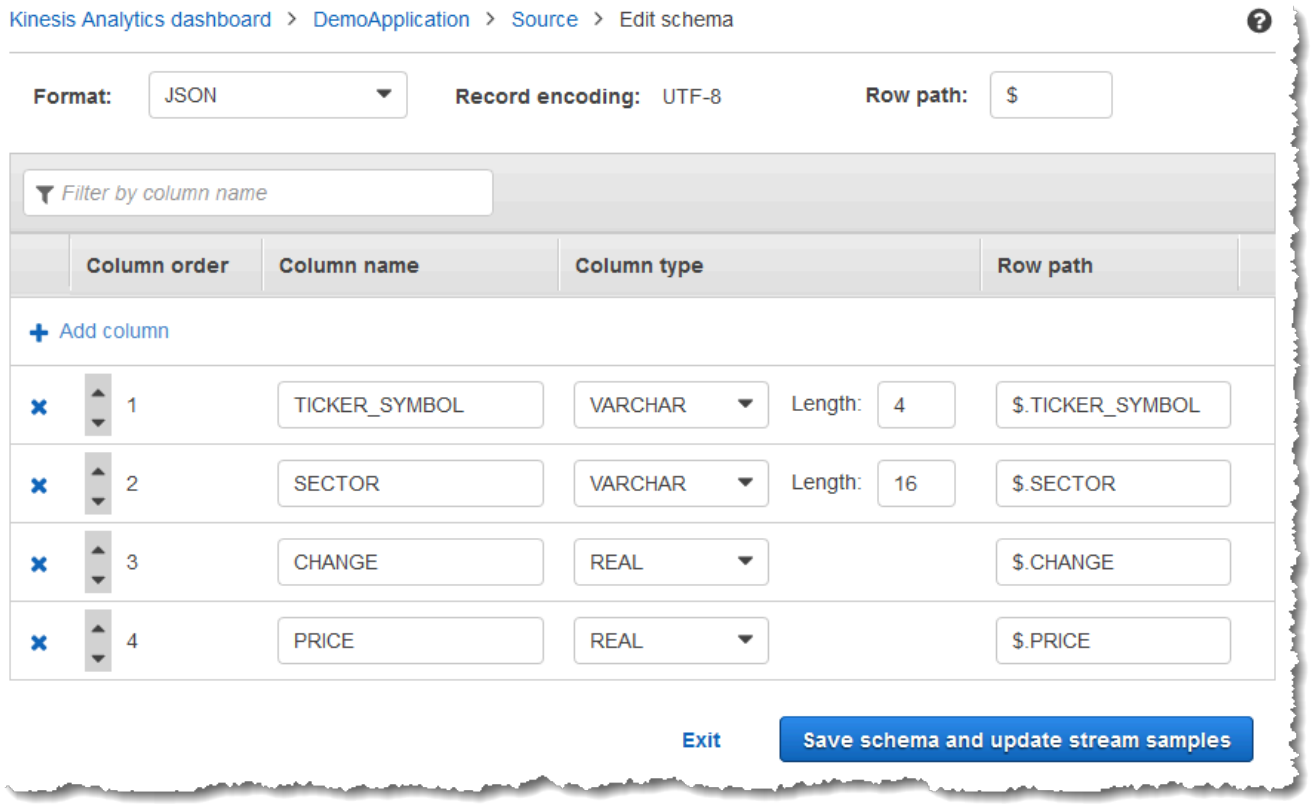
If you need to edit a schema for a streaming source, follow these steps.

To edit the schema for a streaming source

1. On the **Source** page, choose **Edit schema**.



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 the 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.

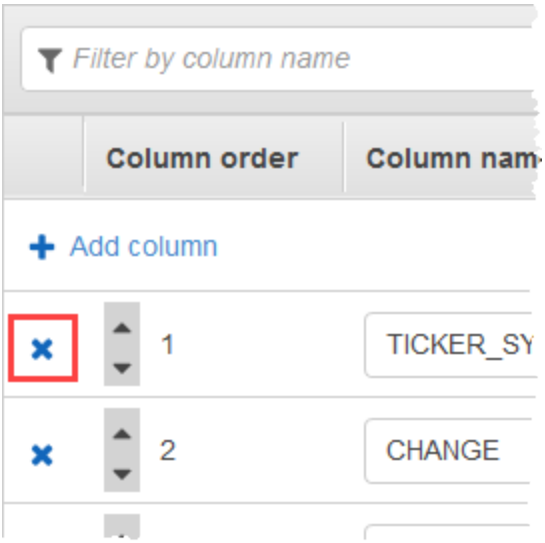
- For **Column type**, type an SQL data type.

  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`.

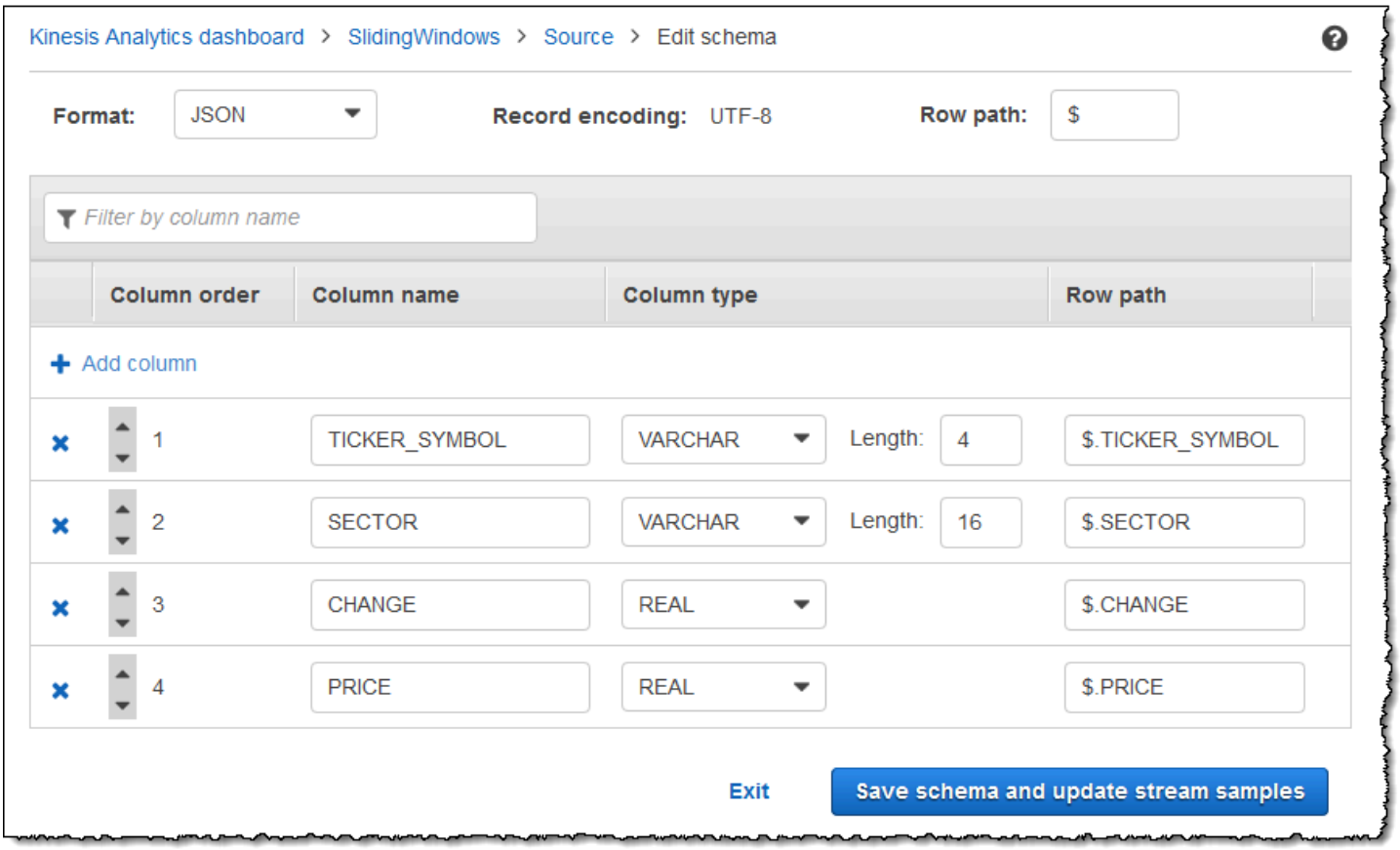
2. To delete a column, choose the **x** icon next to the column number.



3. To rename a column, enter 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, enter 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 the column order, choose the up and down arrows next to the column name.

   For a new column, provide the following information:

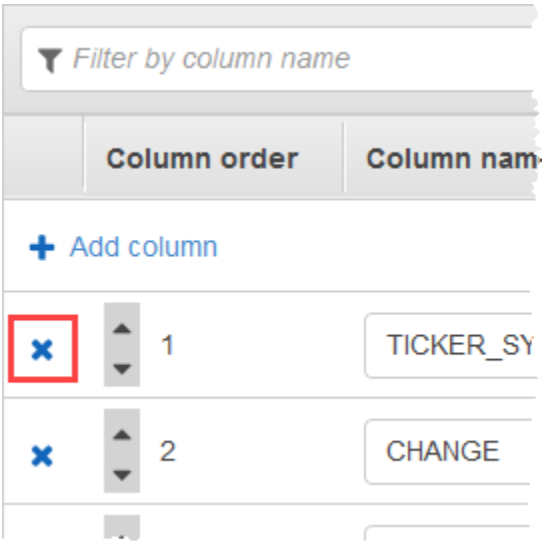
   - For **Column name**, enter a name.

     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**, enter a SQL data type.

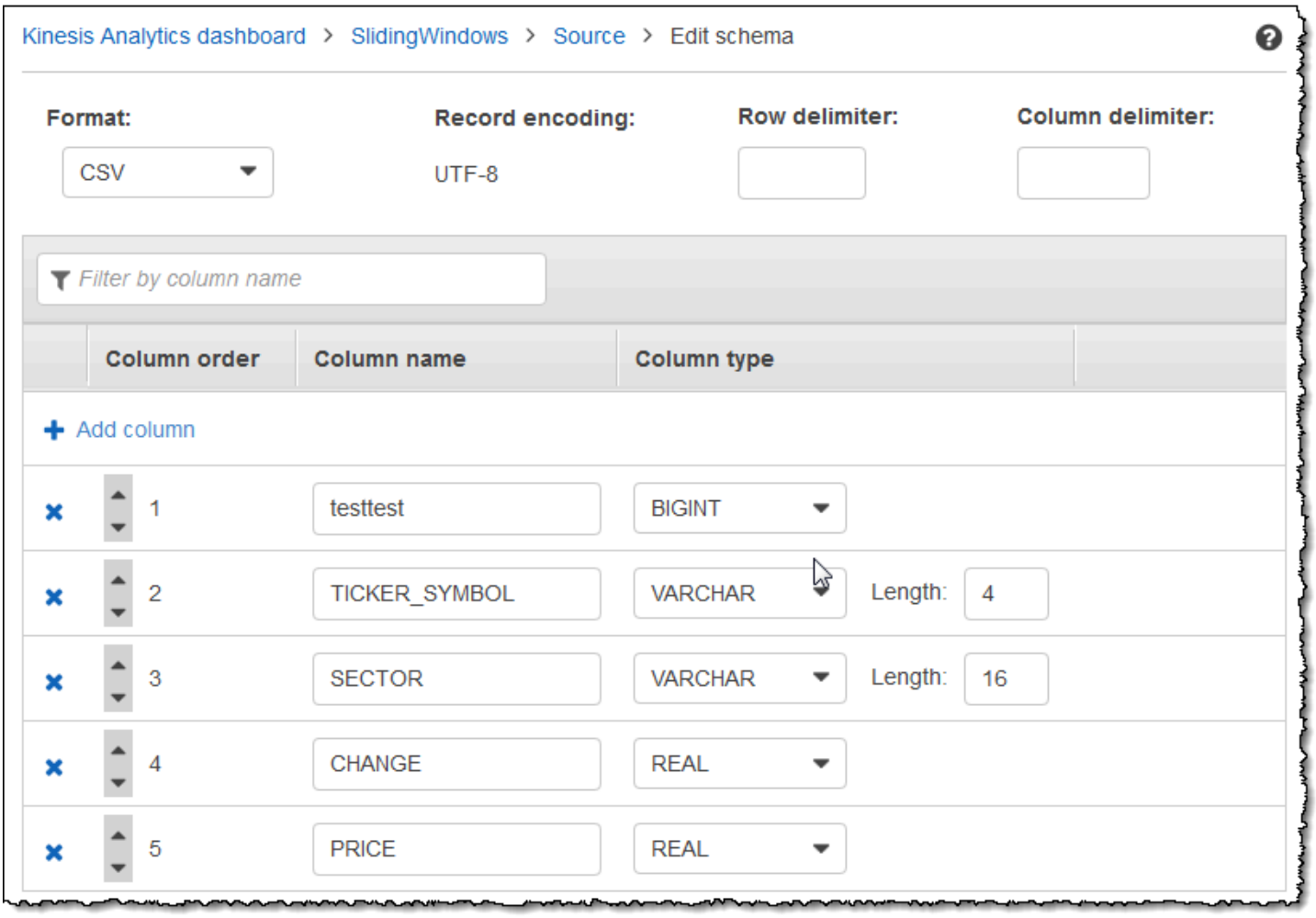
     A column type can be any supported SQL data type. If the new data type is CHAR, VARBINARY, or VARCHAR, specify a data length for **Length**. For more information, see Data Types.

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



5. To rename a column, enter 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.

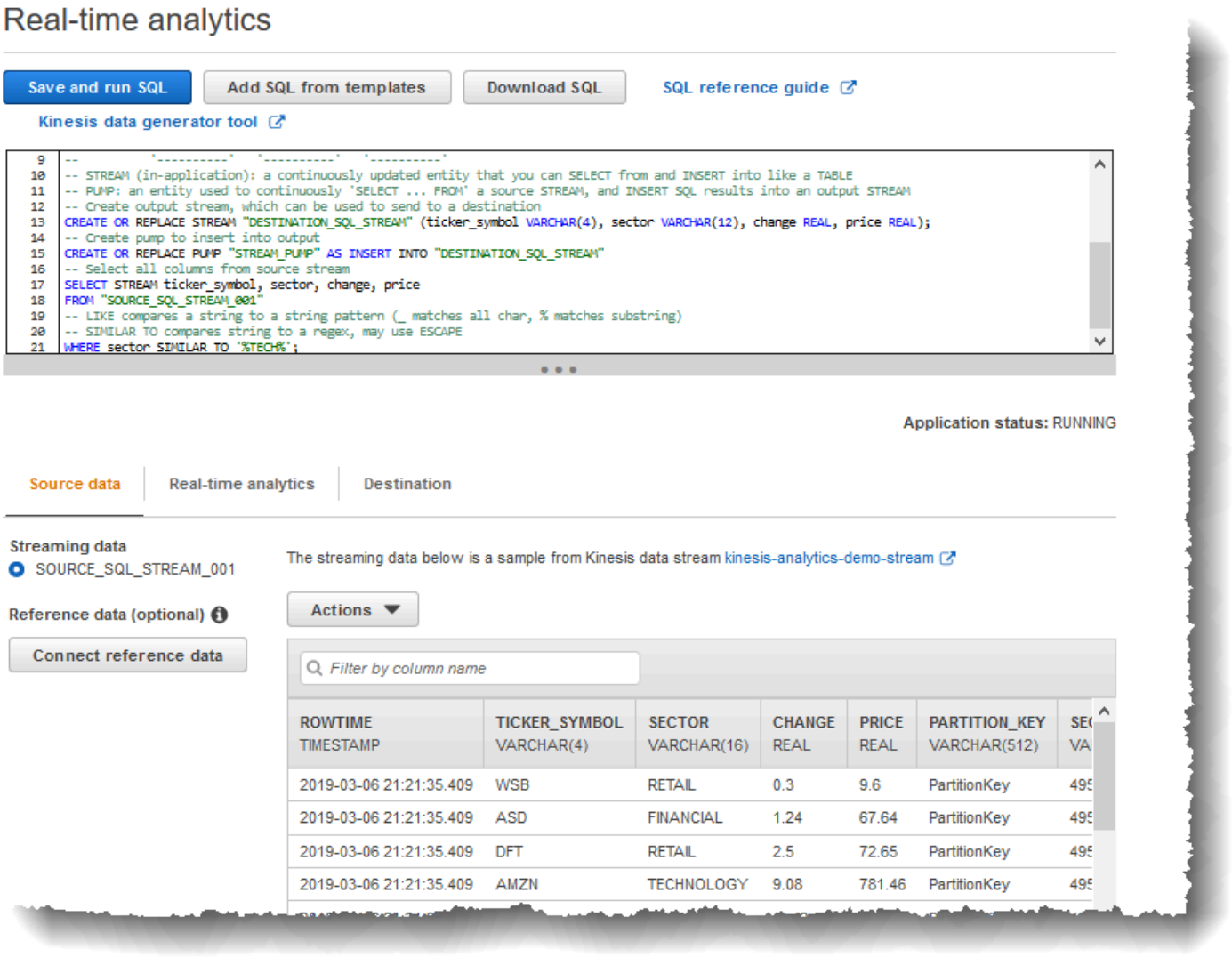
The modified schema appears in the editor and looks similar to the following.



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, enter P in the **Filter by column name** box.

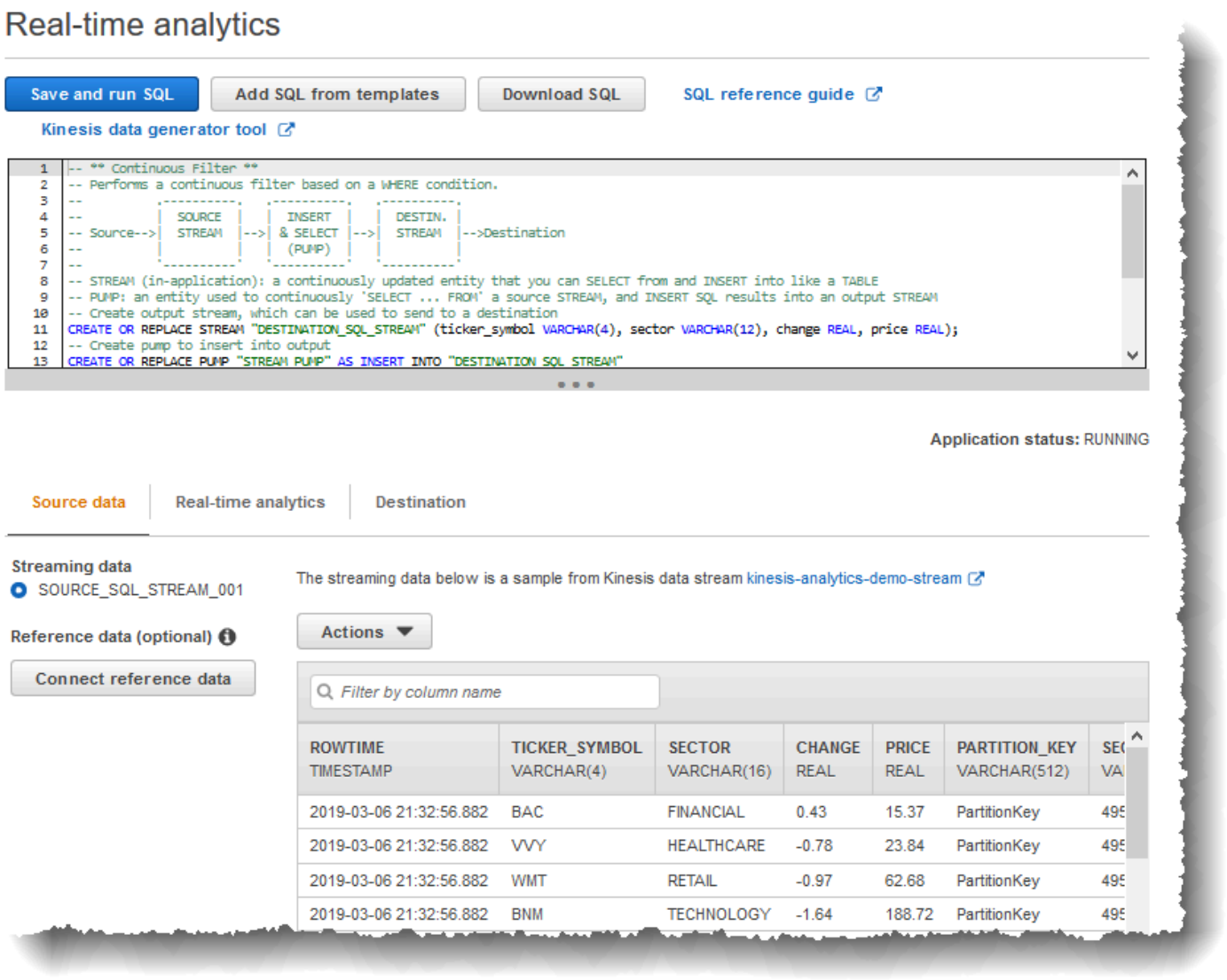
Working with the SQL Editor

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 Kinesis Data Analytics for SQL examples.



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.



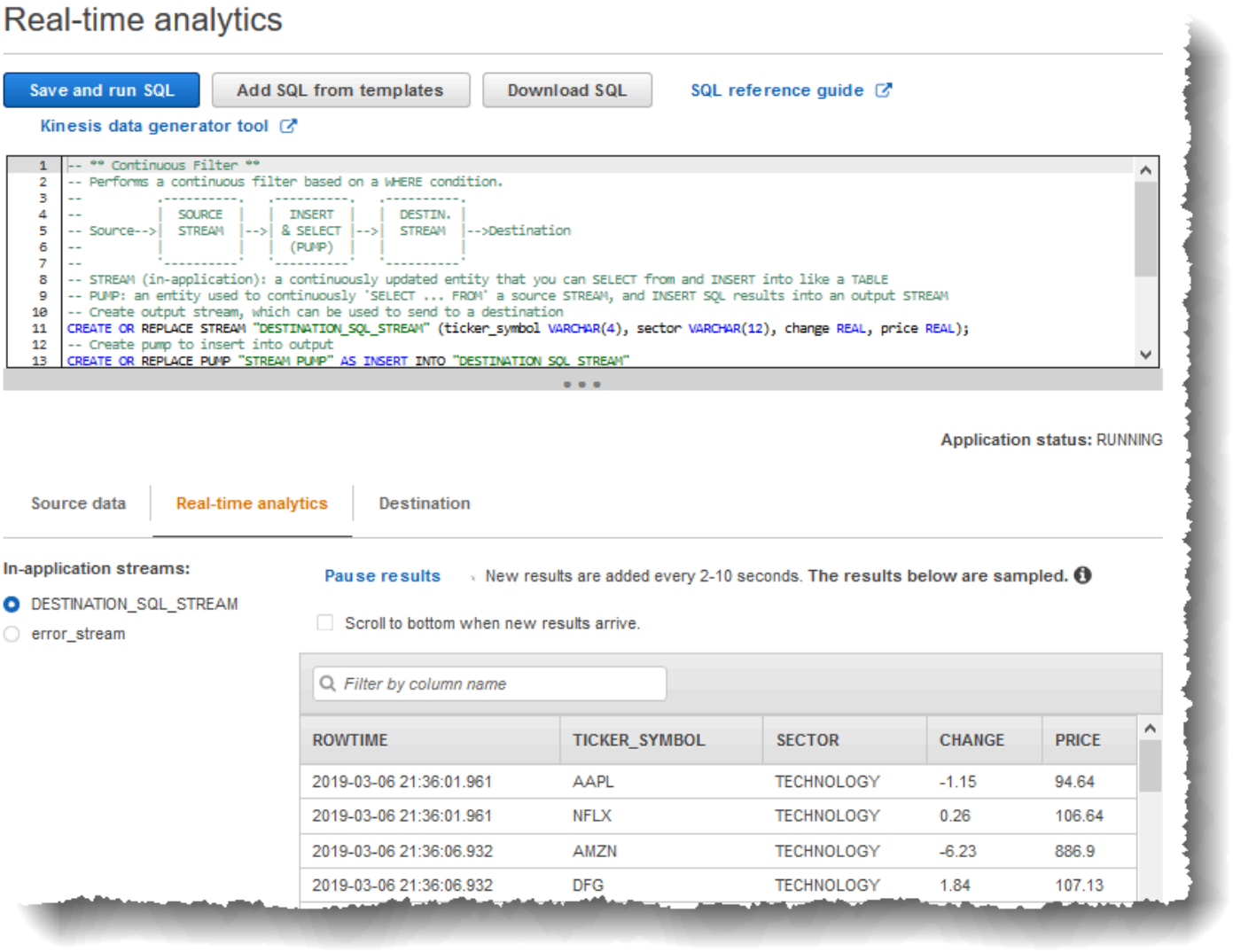
Amazon Kinesis Data Analytics provides the following timestamp columns, so that you don't need to provide explicit mapping in your input configuration:

- **ROWTIME** – Each row in an in-application stream has a special column called `ROWTIME`. This column is the timestamp 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_Timestamp` column. It is the approximate arrival timestamp 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 timestamp values are useful in windowed queries that are time-based. For more information, see Windowed Queries.

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 the 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.

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

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 Amazon Kinesis data streams and Firehose delivery streams. In-application streams exist only in the context of an Amazon Kinesis Data Analytics application. Kinesis data streams and Firehose delivery streams exist independent of your application. You can configure them as a streaming source in your application input configuration or as a destination in output configuration.

You can also create more 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 that the input configuration of your application creates an in-application stream named `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
FROM "INPUTSTREAM";
```

You can have multiple writers insert into an in-application stream, and there can be multiple readers selected from the stream. Think of an in-application stream as implementing a publish/subscribe messaging paradigm. In this paradigm, the data row, including the 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 you query streams, most SQL statements are bound using a row-based or time-based window. For more information, see Windowed Queries.

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

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.

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

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

Understanding Various Times in Streaming Analytics

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

- **Event time** – The timestamp when the event occurred. This is also sometimes called the *client-side time*. It is often desirable to use this time in analytics because it is the time when an event occurred. However, many event sources, such as mobile phones and web clients, do not have

reliable clocks, which can lead to inaccurate times. In addition, connectivity issues can lead to records appearing on a stream not in the same order the events occurred.

- **Ingest time** – The timestamp of when record was added to the streaming source. Amazon Kinesis Data Streams includes a field called `APPROXIMATE_ARRIVAL_TIME` 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 does not accurately reflect the event time.) This `ROWTIME` is accurate in relation to the wall clock, but it might not be the time when the event actually occurred.

Using each of these times in windowed queries that are time-based has advantages and disadvantages. We recommend that 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 that you want to associate 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. 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 (the ingest timestamp shows an earlier value compared to the ROWTIME value) are also processed successfully.

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

```
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.

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).

```
ROWTIME                  INGEST_TIME     TICKER_SYMBOL  SYMBOL_COUNT
```

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

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

You can combine the results for a final accurate count per minute by pushing the results to a downstream database. For example, you can configure the application output to persist the results to a Firehose delivery stream that can write to an Amazon Redshift table. After results are in an Amazon Redshift table, you can query the 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 named `SOURCE_SQL_STREAM_001`. It continuously receives stock prices from a demo stream (a Kinesis data stream). The schema is as follows:

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

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

```
SELECT STREAM TICKER_SYMBOL, PRICE
```

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

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

To test the query

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

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

Windowed Queries

SQL queries in your application code execute continuously over in-application streams. An in-application stream represents unbounded data that flows 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. (The timestamp for a new row is greater than or equal to the 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.

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

You can specify a query to process records in a tumbling window, sliding window, or stagger window manner, depending on your application needs. Kinesis Data Analytics supports the following window types:

- Stagger Windows: A query that aggregates data using keyed time-based windows that open as data arrives. The keys allow for multiple overlapping windows. This is the recommended way to aggregate data using time-based windows, because Stagger Windows reduce late or out-of-order data compared to Tumbling windows.
- Tumbling Windows: A query that aggregates data using distinct time-based windows that open and close at regular intervals.
- Sliding Windows: A query that aggregates data continuously, using a fixed time or rowcount interval.

Stagger Windows

Using *stagger windows* is a windowing method that is suited for analyzing groups of data that arrive at inconsistent times. It is well suited for any time-series analytics use case, such as a set of related sales or log records.

For example, VPC Flow Logs have a capture window of approximately 10 minutes. But they can have a capture window of up to 15 minutes if you're aggregating data on the client. Stagger windows are ideal for aggregating these logs for analysis.

Stagger windows address the issue of related records not falling into the same time-restricted window, such as when tumbling windows were used.

Partial Results with Tumbling Windows

There are certain limitations with using Tumbling Windows for aggregating late or out-of-order data.

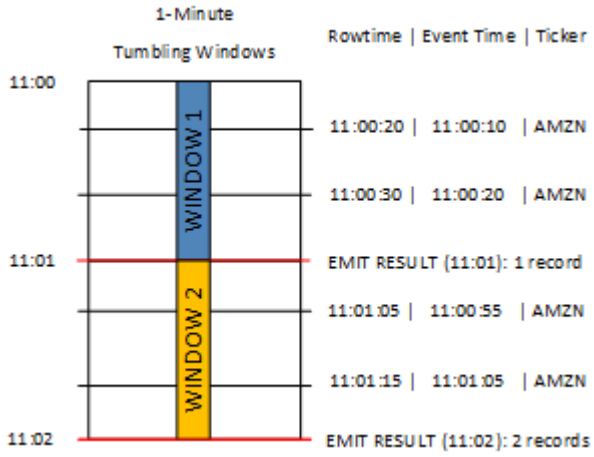
If tumbling windows are used to analyze groups of time-related data, the individual records might fall into separate windows. So then the partial results from each window must be combined later to yield complete results for each group of records.

In the following tumbling window query, records are grouped into windows by row time, event time, and ticker symbol:

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
```

```
    TICKER_SYMBOL VARCHAR(4),
    EVENT_TIME timestamp,
    TICKER_COUNT     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        TICKER_SYMBOL,
        FLOOR(EVENT_TIME TO MINUTE),
        COUNT(TICKER_SYMBOL) AS TICKER_COUNT
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR(EVENT_TIME TO MINUTE),
 STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE);
```

In the following diagram, an application is counting the number of trades it receives, based on when the trades happened (event time) with one minute of granularity. The application can use a tumbling window for grouping data based on row time and event time. The application receives four records that all arrive within one minute of each other. It groups the records by row time, event time, and ticker symbol. Because some of the records arrive after the first tumbling window ends, the records do not all fall within the same one-minute tumbling window.



The preceding diagram has the following events.

ROWTIME	EVENT_TIME	TICKER_SYMBOL
11:00:20	11:00:10	AMZN
11:00:30	11:00:20	AMZN

ROWTIME	EVENT_TIME	TICKER_SYMBOL
11:01:05	11:00:55	AMZN
11:01:15	11:01:05	AMZN

The result set from the tumbling window application looks similar to the following.

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

In the result set preceding, three results are returned:

- A record with a `ROWTIME` of 11:01:00 that aggregates the first two records.
- A record at 11:02:00 that aggregates just the third record. This record has a `ROWTIME` within the second window, but an `EVENT_TIME` within the first window.
- A record at 11:02:00 that aggregates just the fourth record.

To analyze the complete result set, the records must be aggregated in the persistence store. This adds complexity and processing requirements to the application.

Complete Results with Stagger Windows

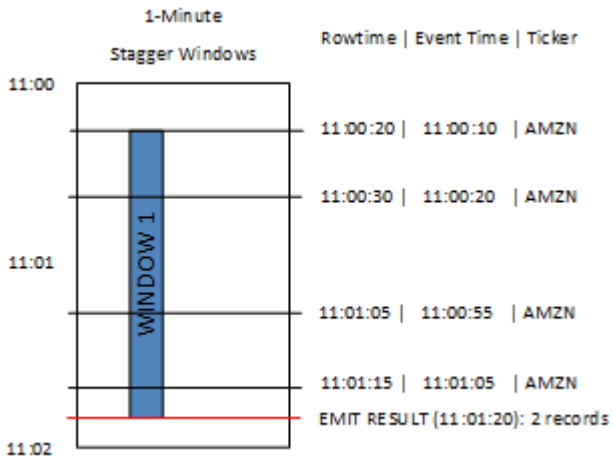
To improve the accuracy of analyzing time-related data records, Kinesis Data Analytics offers a new window type called *stagger windows*. In this window type, windows open when the first event matching the partition key arrives, and not on a fixed time interval. The windows close based on the age specified, which is measured from the time when the window opened.

A stagger window is a separate time-restricted window for each key grouping in a window clause. The application aggregates each result of the window clause inside its own time window, rather than using a single window for all results.

In the following stagger window query, records are grouped into windows by event time and ticker symbol:

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

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

In the following diagram, events are aggregated by event time and ticker symbol into stagger windows.



The preceding diagram has the following events, which are the same events as the tumbling window application analyzed:

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

The result set from the stagger window application looks similar to the following.

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

The returned record aggregates the first three input records. The records are grouped by one-minute stagger windows. The stagger window starts when the application receives the first AMZN record (with a `ROWTIME` of 11:00:20). When the 1-minute stagger window expires (at 11:01:20), a record with the results that fall within the stagger window (based on `ROWTIME` and `EVENT_TIME`) is written to the output stream. Using a stagger window, all of the records with a `ROWTIME` and `EVENT_TIME` within a one-minute window are emitted in a single result.

The last record (with an `EVENT_TIME` outside the one-minute aggregation) is aggregated separately. This is because `EVENT_TIME` is one of the partition keys that is used to separate the records into result sets, and the partition key for `EVENT_TIME` for the first window is `11:00`.

The syntax for a stagger window is defined in a special clause, `WINDOWED BY`. This clause is used instead of the `GROUP BY` clause for streaming aggregations. The clause appears immediately after the optional `WHERE` clause and before the `HAVING` clause.

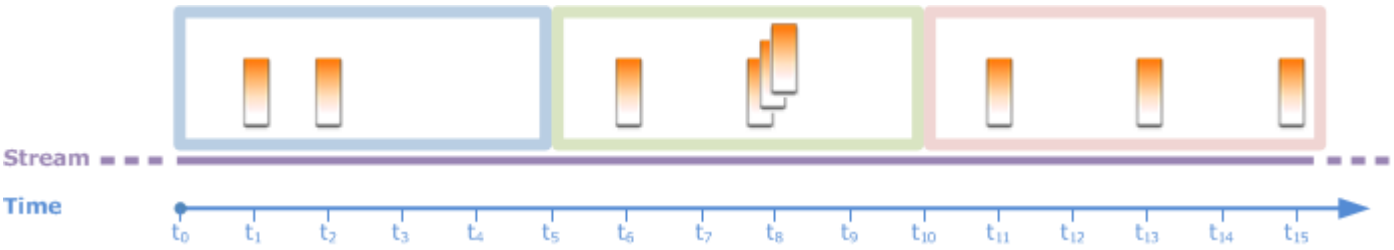
The stagger window is defined in the `WINDOWED BY` clause and takes two parameters: partition keys and window length. The partition keys partition the incoming data stream and define when the window opens. A stagger window opens when the first event with a unique partition key

appears on the stream. The stagger window closes after a fixed time period defined by the window length. The syntax is shown in the following code example:

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

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. It is 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 that you want to find aggregate (min, max) prices for each ticker over a one-minute window. You can use the following query.

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

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. However, 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, for example, 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:

```
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"
```

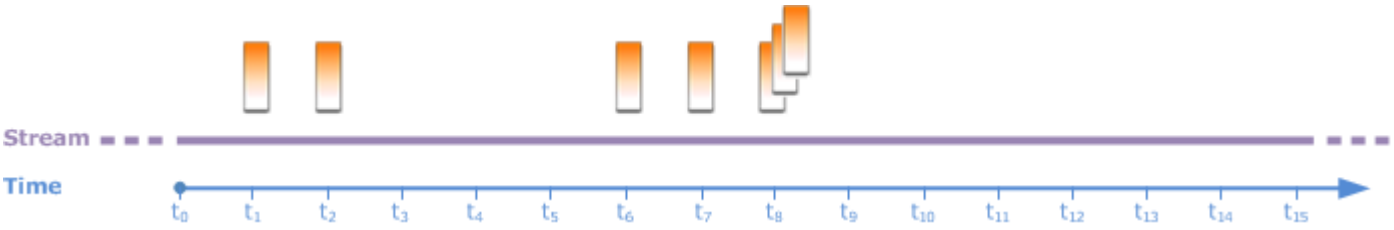
```
SELECT STREAM Ticker_Symbol,
              MIN(Price) AS Min_Price,
              MAX(Price) AS Max_Price
FROM    "SOURCE_SQL_STREAM_001"
GROUP BY Ticker_Symbol,
         STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

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. This example assumes 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.



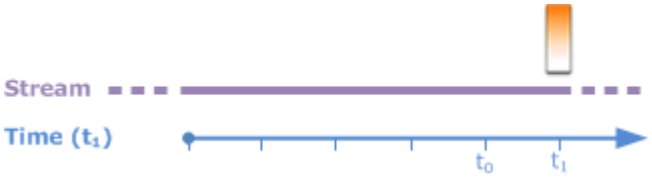
Keep the following in mind:

- The example assumes 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.

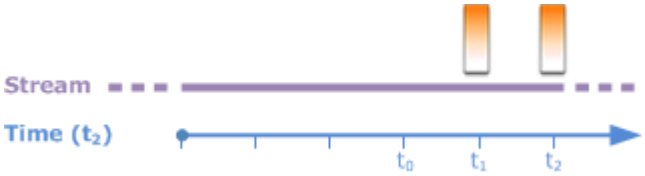
Suppose that the query starts executing at t_0. Then the following occurs:

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.



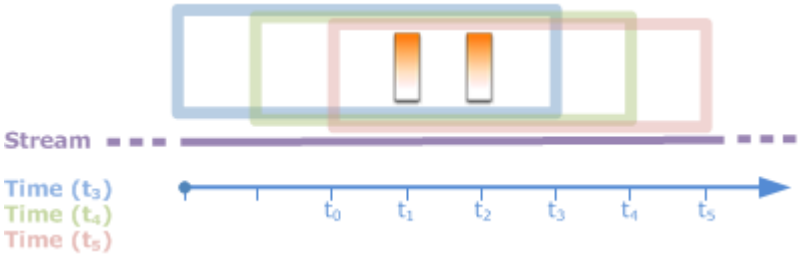
3. At time t_2, another record appears, and the query emits count 2.



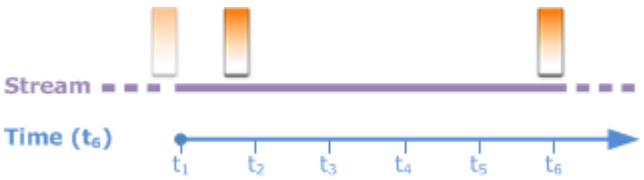
4. The 5-second window slides with time:

   - At t_3, the sliding window t_3 to t_0
   - At t4 (sliding window t_4 to t_0)
   - At t_5 the sliding window t_5–t_0

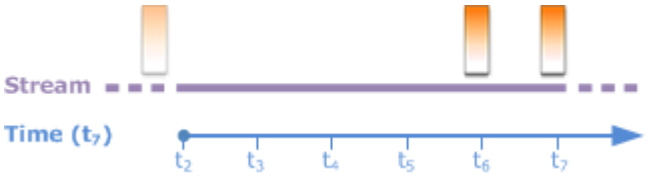
   At all of these times, the 5-second window has the same records—there are no new records. Therefore, the query doesn't emit any output.



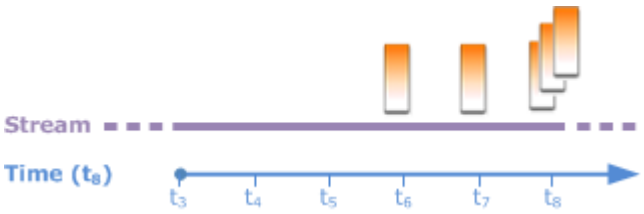
5. At time t_6, the 5-second window is (t_6 to t_1). The query detects one new record at t_6 so it emits output 2. The record at t_1 is no longer in the window and doesn't count.



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. This is because the source data must be read from the stream again.

The following example queries 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.

```
(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.

```
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.

```
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.

```
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.

```
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.

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 that you have the following in-application streams:

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

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

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

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

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

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

In this example, your query joins both the `OrderStream` and `TradeStream`. However, because we want only trades placed one minute after the orders, the query defines the 1-minute window over the `TradeStream`. For information about windowed queries, see Sliding Windows.

```
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:

```
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 that you want stock orders for which there are no trades within one minute of stock order being placed, and trades reported within the same window but for some other orders. This is example of an *outer join*.

```
SELECT STREAM
    ROWTIME,
```

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

Kinesis Data Analytics for SQL examples

This section provides examples of creating and working with applications in Amazon Kinesis Data Analytics. They include example code and step-by-step instructions to help you create Kinesis Data Analytics applications and test your results.

Before you explore these examples, we recommend that you first review Amazon Kinesis Data Analytics for SQL Applications: How It Works and Getting Started with Amazon Kinesis Data Analytics for SQL Applications.

Topics

Examples: Transforming Data

There are times when your application code must preprocess incoming records before performing any analytics in Amazon Kinesis Data Analytics. This can happen for various reasons, such as when records don't conform to the supported record formats, resulting in unnormalized columns in the 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.

Preprocessing Streams with Lambda

For information about preprocessing streams with AWS Lambda, see Preprocessing Data Using a Lambda Function.

Topics

- Example: Transforming DateTime Values
- Example: Transforming Multiple Data Types

Examples: Transforming String Values

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

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

  ```
  { "Col_A" : "string",
    "Col_B" : "string",
    "Col_C" : "string",
    "Col_D" : "string",
  ```

```
  "Col_E_Unstructured" : "value,value,value,value"}

{ "Col_A" : "string",
  "Col_B" : "string",
  "Col_C" : "string",
  "Col_D" : "string",
  "Col_E_Unstructured" : "value,value,value,value"}
```

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

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

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

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

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

- **String operators** – Operators such as `LIKE` and `SIMILAR` are useful in comparing strings. For more information, see String Operators in the *Amazon Managed Service for Apache Flink SQL Reference*.
- **SQL functions** – The following functions are useful when manipulating individual strings. For more information, see String and Search Functions in the *Amazon Managed Service for Apache Flink SQL Reference*.
  - `CHAR_LENGTH` – Provides the length of a string.
  - `INITCAP` – Returns a converted version of the input string such that the first character of each space-delimited word is uppercase, and all other characters are lowercase.
  - `LOWER/UPPER` – Converts a string to lowercase or uppercase.

- `OVERLAY` – Replaces a portion of the first string argument (the original string) with the second string argument (the replacement string).
- `POSITION` – Searches for a string within another string.
- `REGEX_REPLACE` – Replaces a substring with an alternative substring.
- `SUBSTRING` – Extracts a portion of a source string starting at a specific position.
- `TRIM` – Removes instances of the specified character from the beginning or end of the source string.

- **SQL extensions** – These are useful for working with unstructured strings such as logs and URIs. For more information, see Log Parsing Functions in the *Amazon Managed Service for Apache Flink SQL Reference*.
  - `FAST_REGEX_LOG_PARSER` – Works similar to the regex parser, but it takes several shortcuts to ensure faster results. For example, the fast regex parser stops at the first match it finds (known as *lazy semantics*).
  - `FIXED_COLUMN_LOG_PARSE` – Parses fixed-width fields and automatically converts them to the given SQL types.
  - `REGEX_LOG_PARSE` – Parses a string based on default Java regular expression patterns.
  - `SYS_LOG_PARSE` – Parses entries commonly found in UNIX/Linux system logs.
  - `VARIABLE_COLUMN_LOG_PARSE` – Splits an input string into fields separated by a delimiter character or a delimiter string.
  - `W3C_LOG_PARSE` – Can be used for quickly formatting Apache logs.

For examples using these functions, see the following topics:

Topics

- Example: Extracting a Portion of a String (SUBSTRING Function)
- Example: Replacing a Substring using Regex (REGEX_REPLACE Function)
- Example: Parsing Log Strings Based on Regular Expressions (REGEX_LOG_PARSE Function)
- Example: Parsing Web Logs (W3C_LOG_PARSE Function)
- Example: Split Strings into Multiple Fields (VARIABLE_COLUMN_LOG_PARSE Function)

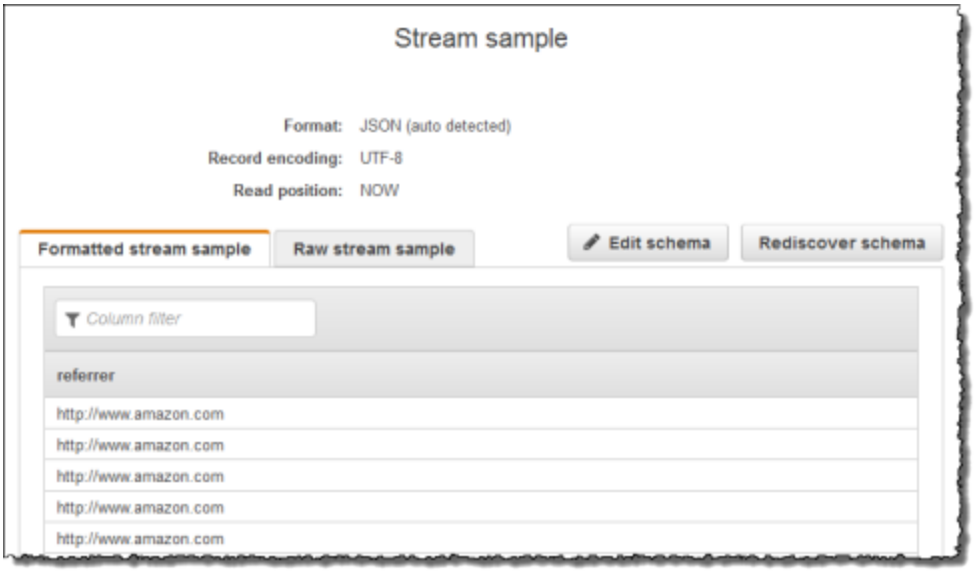
Example: Extracting a Portion of a String (SUBSTRING Function)

This example uses the SUBSTRING function to transform a string in Amazon Kinesis Data Analytics. The SUBSTRING function extracts a portion of a source string starting at a specific position. For more information, see SUBSTRING in the *Amazon Managed Service for Apache Flink SQL Reference*.

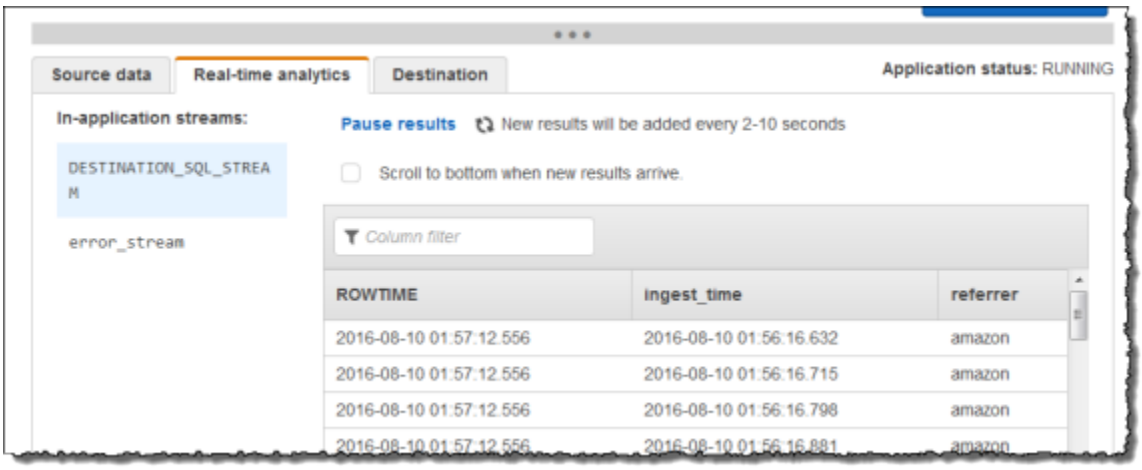
In this example, you write the following records to an Amazon Kinesis data stream.

```
{ "REFERRER" : "http://www.amazon.com" }
{ "REFERRER" : "http://www.amazon.com"}
{ "REFERRER" : "http://www.amazon.com"}
...
```

You then create an Kinesis Data Analytics application on the console, using 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 (REFERRER), as shown.



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



Topics

- Step 1: Create a Kinesis Data Stream
- Step 2: Create the Kinesis Data Analytics Application

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and create a stream with one shard. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
4. Run the following Python code to populate sample log records. This simple code continuously writes the same log record to the stream.

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {"REFERRER": "http://www.amazon.com"}
```

```
def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Kinesis Data Analytics Application

Next, create an Kinesis Data Analytics application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**, type an application name, and choose **Create application**.
3. On the application details page, choose **Connect streaming data**.
4. On the **Connect to source** page, do the following:
   a. Choose the stream that you created in the preceding section.
   b. Choose the option to create an IAM role.
   c. Choose **Discover schema**. Wait for the console to show the inferred schema and samples records used to infer the schema for the in-application stream created. The inferred schema has only one column.
   d. Choose **Save and continue**.
5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.
6. In the SQL editor, write the application code, and verify the results as follows:
   a. Copy the following application code and paste it into the editor.

```
-- CREATE OR REPLACE STREAM for cleaned up referrer
```

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "ingest_time" TIMESTAMP,
    "referrer" VARCHAR(32));

CREATE OR REPLACE PUMP "myPUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM
         "APPROXIMATE_ARRIVAL_TIME",
         SUBSTRING("referrer", 12, (POSITION('.com' IN "referrer") -
 POSITION('www.' IN "referrer") - 4))
      FROM "SOURCE_SQL_STREAM_001";
```

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

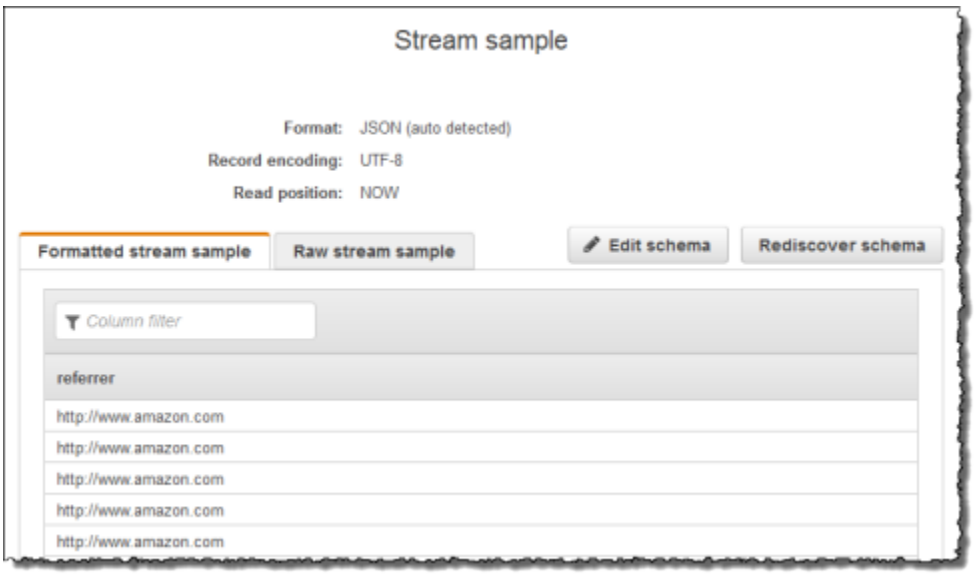
Example: Replacing a Substring using Regex (REGEX_REPLACE Function)

This example uses the REGEX_REPLACE function to transform a string in Amazon Kinesis Data Analytics. REGEX_REPLACE replaces a substring with an alternative substring. For more information, see REGEX_REPLACE in the *Amazon Managed Service for Apache Flink SQL Reference*.

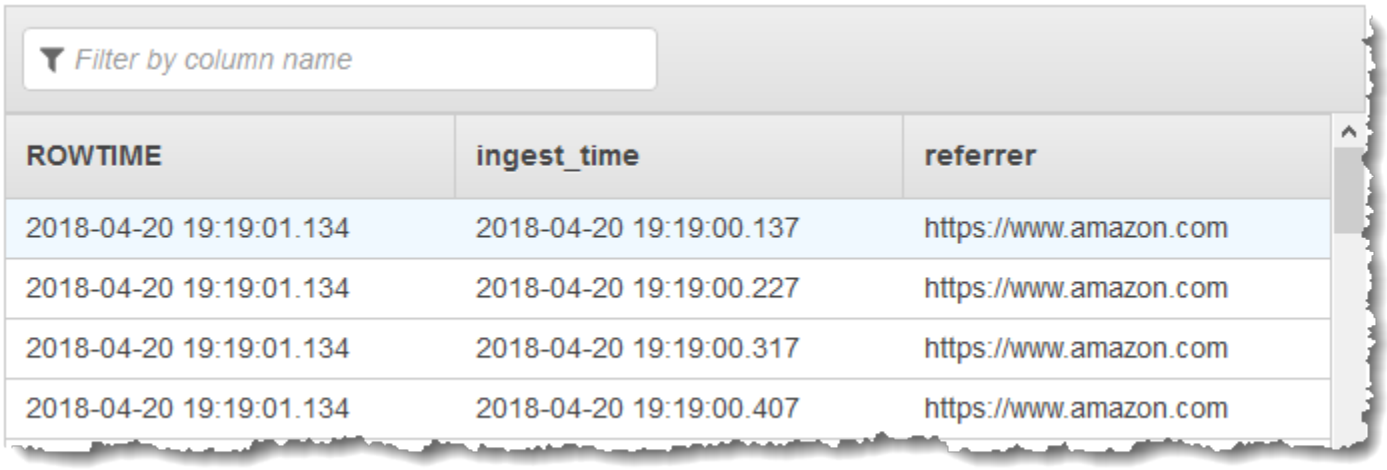
In this example, you write the following records to an Amazon Kinesis data stream:

```
{ "REFERRER" : "http://www.amazon.com" }
{ "REFERRER" : "http://www.amazon.com"}
{ "REFERRER" : "http://www.amazon.com"}
...
```

You then create an Kinesis Data Analytics application on 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 (REFERRER) as shown.



Then, you use the application code with the REGEX_REPLACE function to convert the URL to use `https://` instead of `http://`. You insert the resulting data into another in-application stream, as shown following:



Topics

- Step 1: Create a Kinesis Data Stream
- Step 2: Create the Kinesis Data Analytics Application

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and create a stream with one shard. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
4. Run the following Python code to populate the sample log records. This simple code continuously writes the same log record to the stream.

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {"REFERRER": "http://www.amazon.com"}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Kinesis Data Analytics Application

Next, create an Kinesis Data Analytics application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**, type an application name, and choose **Create application**.

3. On the application details page, choose **Connect streaming data**.
4. On the **Connect to source** page, do the following:

   a. Choose the stream that you created in the preceding section.

   b. Choose the option to create an IAM role.

   c. Choose **Discover schema**. Wait for the console to show the inferred schema and samples records used to infer the schema for the in-application stream created. The inferred schema has only one column.

   d. Choose **Save and continue**.

5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.
6. In the SQL editor, write the application code and verify the results as follows:

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

```
-- CREATE OR REPLACE STREAM for cleaned up referrer
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "ingest_time" TIMESTAMP,
    "referrer" VARCHAR(32));

CREATE OR REPLACE PUMP "myPUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM
         "APPROXIMATE_ARRIVAL_TIME",
         REGEX_REPLACE("REFERRER", 'http://', 'https://', 1, 0)
      FROM "SOURCE_SQL_STREAM_001";
```

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

Example: Parsing Log Strings Based on Regular Expressions (REGEX_LOG_PARSE Function)

This example uses the REGEX_LOG_PARSE function to transform a string in Amazon Kinesis Data Analytics. REGEX_LOG_PARSE parses a string based on default Java regular expression patterns. For more information, see REGEX_LOG_PARSE in the *Amazon Managed Service for Apache Flink SQL Reference*.

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

```
{"LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] \"GET /index.php HTTP/1.1\"
 200 125 \"-\" \"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0\""}
{"LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] \"GET /index.php HTTP/1.1\"
 200 125 \"-\" \"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0\""}
{"LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] \"GET /index.php HTTP/1.1\"
 200 125 \"-\" \"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0\""}
...
```

You then create an Kinesis Data Analytics application on 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 (LOGENTRY), as shown following.

ROWTIME TIMESTAMP	LOGENTRY VARCHAR(256)
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"

Then, you use the application code with the `REGEX_LOG_PARSE` function to parse the log string to retrieve the data elements. You insert the resulting data into another in-application stream, as shown in the following screenshot:

Filter by column name

ROWTIME	LOGENTRY	MATCH1	MATCH2
2018-05-09 18:16:11.616	203.0.113.24 - - [25/Mar	203.0.113.24 - - [25/Mar	125 "-" "Mozilla/5.0 [
2018-05-09 18:16:11.616	203.0.113.24 - - [25/Mar	203.0.113.24 - - [25/Mar	125 "-" "Mozilla/5.0 [
2018-05-09 18:16:11.616	203.0.113.24 - - [25/Mar	203.0.113.24 - - [25/Mar	125 "-" "Mozilla/5.0 [
2018-05-09 18:16:11.616	203.0.113.24 - - [25/Mar	203.0.113.24 - - [25/Mar	125 "-" "Mozilla/5.0 [

Topics

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and create a stream with one shard. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
4. Run the following Python code to populate sample log records. This simple code continuously writes the same log record to the stream.

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "
        '"GET /index.php HTTP/1.1" 200 125 "-" '
        '"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0"'
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )
```

```
if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Kinesis Data Analytics Application

Next, create an Kinesis Data Analytics application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**, and specify an application name.
3. On the application details page, choose **Connect streaming data**.
4. On the **Connect to source** page, do the following:

   a. Choose the stream that you created in the preceding section.

   b. Choose the option to create an IAM role.

   c. Choose **Discover schema**. Wait for the console to show the inferred schema and samples records used to infer the schema for the in-application stream created. The inferred schema has only one column.

   d. Choose **Save and continue**.

5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.
6. In the SQL editor, write the application code, and verify the results as follows:

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

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (logentry VARCHAR(24), match1
 VARCHAR(24), match2 VARCHAR(24));

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM T.LOGENTRY, T.REC.COLUMN1, T.REC.COLUMN2
    FROM
         (SELECT STREAM LOGENTRY,
             REGEX_LOG_PARSE(LOGENTRY, '(\w.+) (\d.+) (\w.+) (\w.+)') AS REC
             FROM SOURCE_SQL_STREAM_001) AS T;
```

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

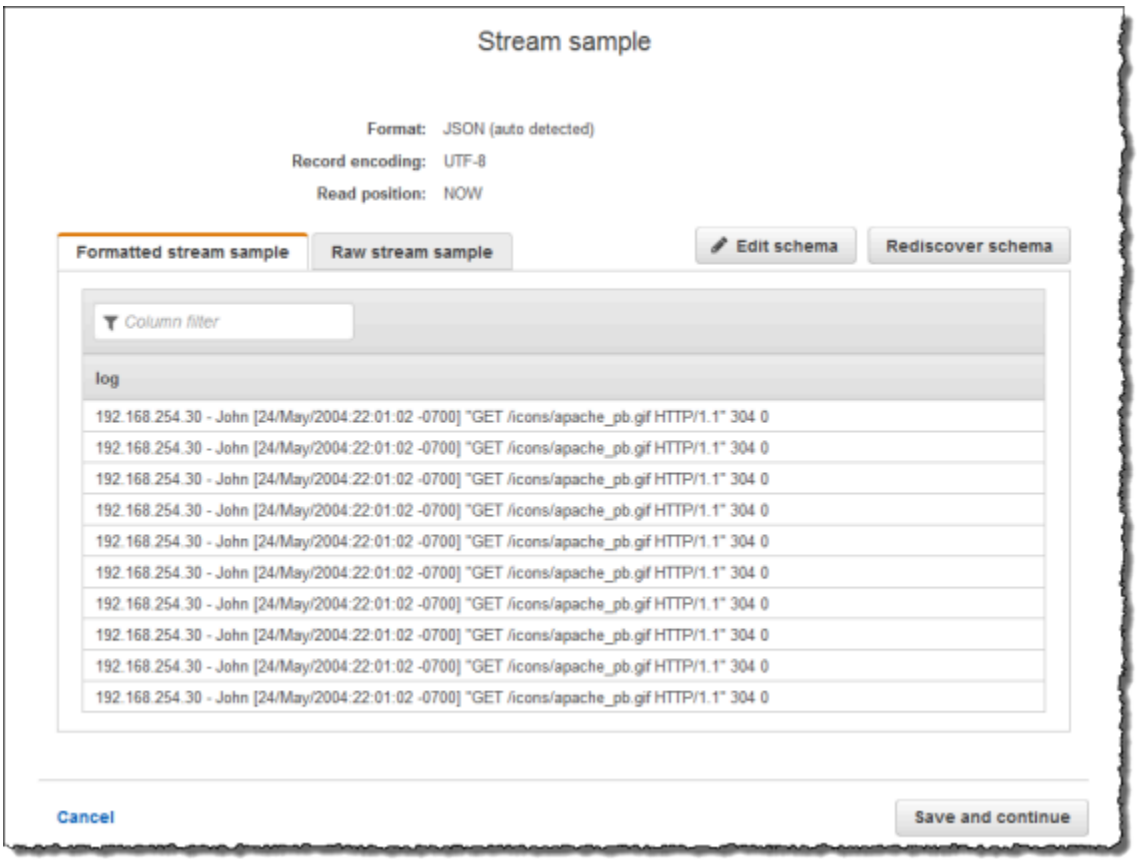
Example: Parsing Web Logs (W3C_LOG_PARSE Function)

This example uses the `W3C_LOG_PARSE` function to transform a string in Amazon Kinesis Data Analytics. You can use `W3C_LOG_PARSE` to format Apache logs quickly. For more information, see W3C_LOG_PARSE in the *Amazon Managed Service for Apache Flink SQL Reference*.

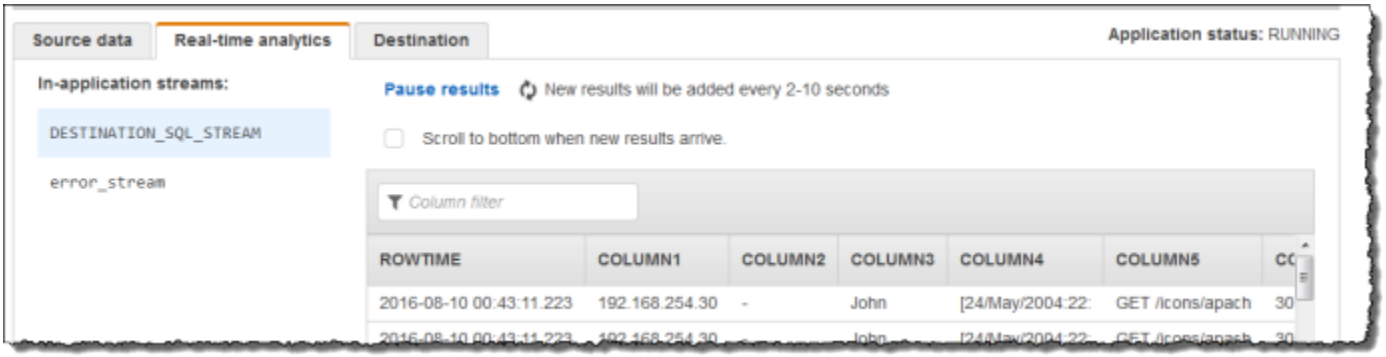
In this example, you write log records to an Amazon Kinesis data stream. Example logs are shown following:

```
{"Log":"192.168.254.30 - John [24/May/2004:22:01:02 -0700] "GET /icons/apache_pba.gif
 HTTP/1.1" 304 0"}
{"Log":"192.168.254.30 - John [24/May/2004:22:01:03 -0700] "GET /icons/apache_pbb.gif
 HTTP/1.1" 304 0"}
{"Log":"192.168.254.30 - John [24/May/2004:22:01:04 -0700] "GET /icons/apache_pbc.gif
 HTTP/1.1" 304 0"}
...
```

You then create an Kinesis Data Analytics application on 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 `W3C_LOG_PARSE` function to parse the log, and create another in-application stream with various log fields in separate columns, as shown following:



Topics

- Step 1: Create a Kinesis Data Stream
- Step 2: Create the Kinesis Data Analytics Application

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and create a stream with one shard. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
4. Run the following Python code to populate the sample log records. This simple code continuously writes the same log record to the stream.

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "log": "192.168.254.30 - John [24/May/2004:22:01:02 -0700] "
        '"GET /icons/apache_pb.gif HTTP/1.1" 304 0'
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Kinesis Data Analytics Application

Create an Kinesis Data Analytics application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**, type an application name, and choose **Create application**.
3. On the application details page, choose **Connect streaming data**.
4. On the **Connect to source** page, do the following:

   a. Choose the stream that you created in the preceding section.

   b. Choose the option to create an IAM role.

   c. Choose **Discover schema**. Wait for the console to show the inferred schema and samples records used to infer the schema for the in-application stream created. The inferred schema has only one column.

   d. Choose **Save and continue**.

5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.
6. In the SQL editor, write the application code, and verify the results as follows:

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

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
column1 VARCHAR(16),
column2 VARCHAR(16),
column3 VARCHAR(16),
column4 VARCHAR(16),
column5 VARCHAR(16),
column6 VARCHAR(16),
column7 VARCHAR(16));

CREATE OR REPLACE PUMP "myPUMP" AS
INSERT INTO "DESTINATION_SQL_STREAM"
        SELECT STREAM
            l.r.COLUMN1,
            l.r.COLUMN2,
            l.r.COLUMN3,
            l.r.COLUMN4,
            l.r.COLUMN5,
            l.r.COLUMN6,
            l.r.COLUMN7
        FROM (SELECT STREAM W3C_LOG_PARSE("log", 'COMMON')
```

```
                FROM "SOURCE_SQL_STREAM_001") AS l(r);
```

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

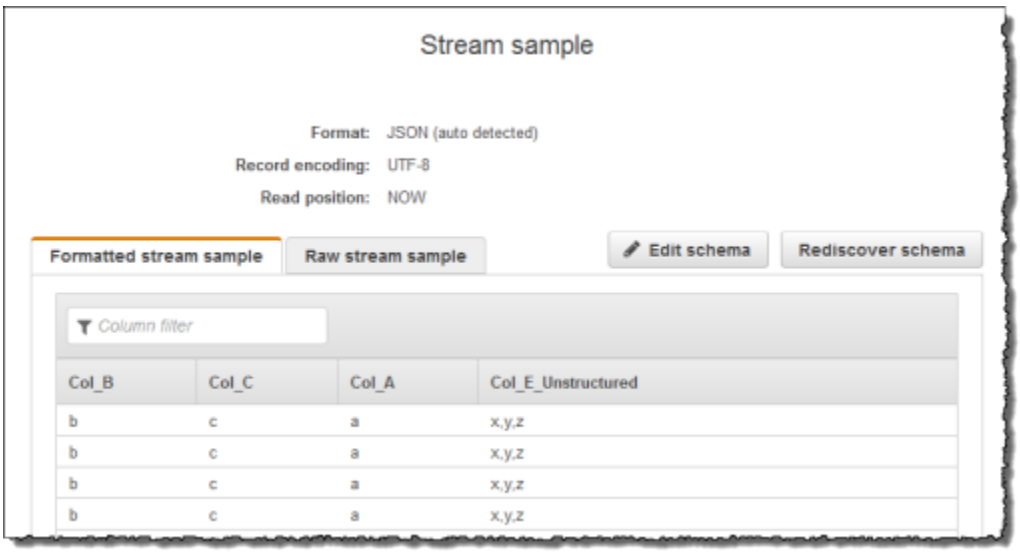
Example: Split Strings into Multiple Fields (VARIABLE_COLUMN_LOG_PARSE Function)

This example uses the `VARIABLE_COLUMN_LOG_PARSE` function to manipulate strings in Kinesis Data Analytics. `VARIABLE_COLUMN_LOG_PARSE` splits an input string into fields separated by a delimiter character or a delimiter string. For more information, see VARIABLE_COLUMN_LOG_PARSE in the *Amazon Managed Service for Apache Flink SQL Reference*.

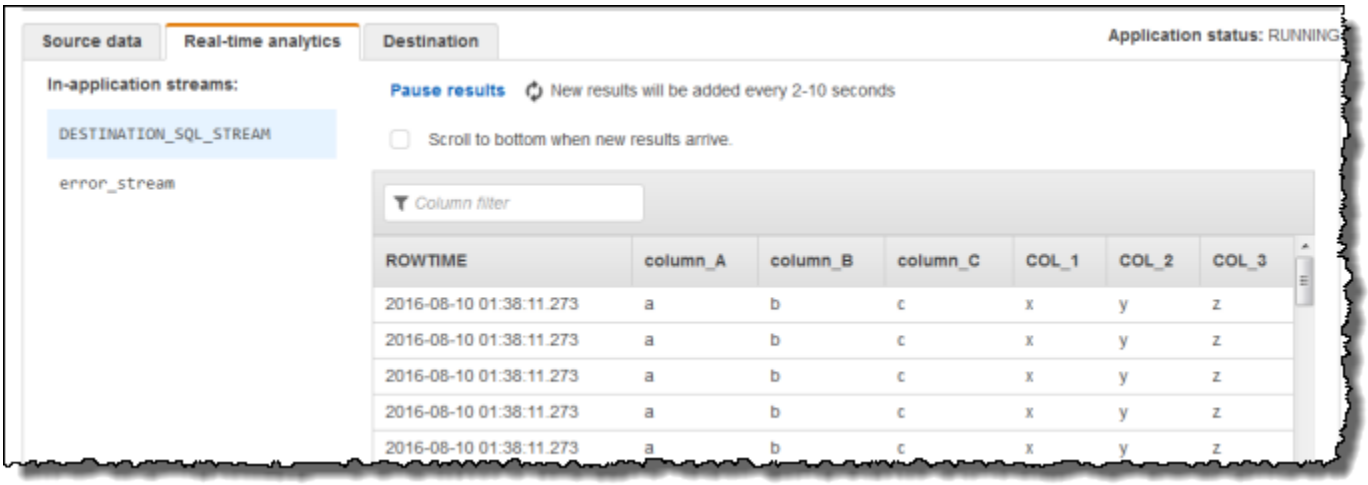
In this example, you write semi-structured records to an Amazon Kinesis data stream. The example records are as follows:

```
{ "Col_A" : "string",
  "Col_B" : "string",
  "Col_C" : "string",
  "Col_D_Unstructured" : "value,value,value,value"}
{ "Col_A" : "string",
  "Col_B" : "string",
  "Col_C" : "string",
  "Col_D_Unstructured" : "value,value,value,value"}
```

You then create an Kinesis Data Analytics application on the console, using the Kinesis stream as the streaming source. The discovery process reads sample records on the streaming source and infers an in-application schema with four columns, as shown following:



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



Topics

- Step 1: Create a Kinesis Data Stream
- Step 2: Create the Kinesis Data Analytics Application

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and create a stream with one shard. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
4. Run the following Python code to populate the sample log records. This simple code continuously writes the same log record to the stream.

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {"Col_A": "a", "Col_B": "b", "Col_C": "c", "Col_E_Unstructured":
 "x,y,z"}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Kinesis Data Analytics Application

Create an Kinesis Data Analytics application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.

2. Choose **Create application**, type an application name, and choose **Create application**.
3. On the application details page, choose **Connect streaming data**.
4. On the **Connect to source** page, do the following:

   a. Choose the stream that you created in the preceding section.

   b. Choose the option to create an IAM role.

   c. Choose **Discover schema**. 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.

   d. Choose **Save and continue**.

5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.
6. In the SQL editor, write application code, and verify the results:

   a. 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;
```

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

Example: Transforming DateTime Values

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 Managed Service for Apache Flink SQL Reference*.

- **SQL Functions** – These include the following. For more information, see Date and Time Functions in the *Amazon Managed Service for Apache Flink SQL Reference*.
  - `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 Kinesis Data Analytics is running (UTC).
  - `LOCALTIMESTAMP` – Returns the current time stamp as defined by the environment on which Kinesis Data Analytics is running (UTC).

- **SQL Extensions** – These include the following. For more information, see Date and Time Functions and Datetime Conversion Functions in the *Amazon Managed Service for Apache Flink SQL Reference*.
  - `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.

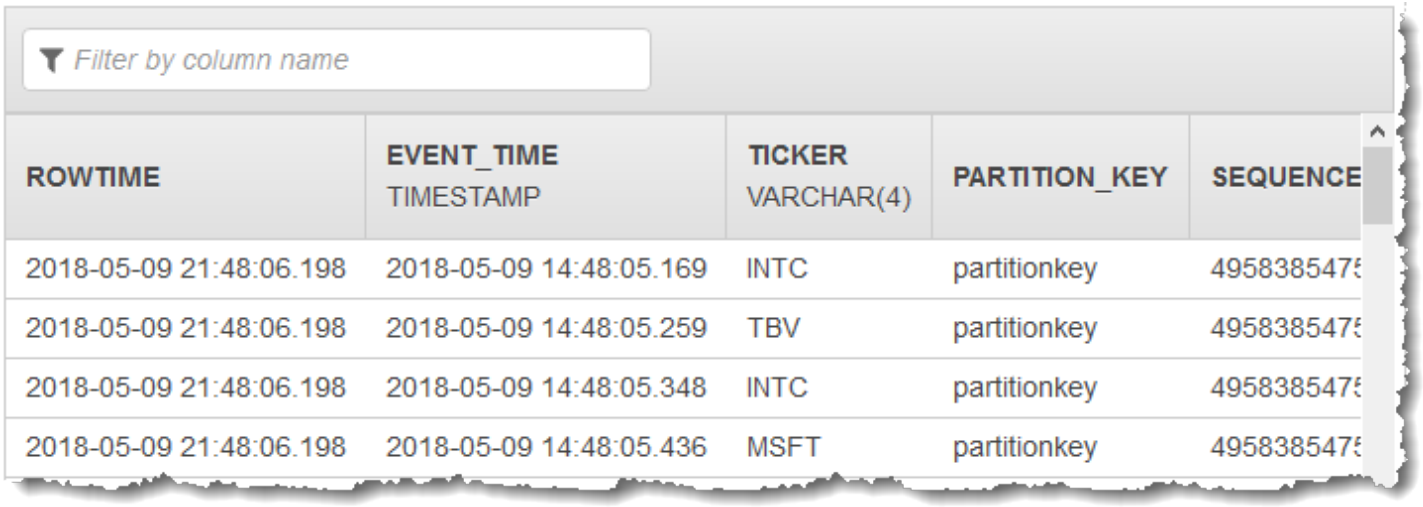
Most of the preceding SQL functions use a format to convert the columns. The format is flexible. For example, you can specify the format `yyyy-MM-dd hh:mm:ss` to convert an input string `2009-09-16 03:15:24` into a time stamp. For more information, Char To Timestamp(Sys) in the *Amazon Managed Service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

Example: Transforming Dates

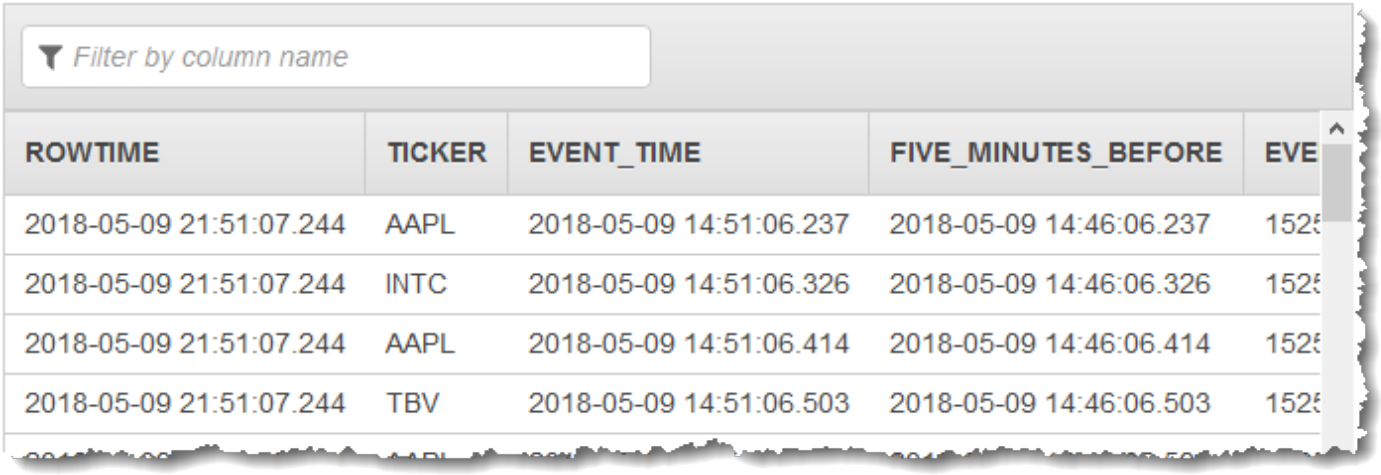
In this example, you write the following records to an Amazon Kinesis data stream.

```
{"EVENT_TIME": "2018-05-09T12:50:41.337510", "TICKER": "AAPL"}
{"EVENT_TIME": "2018-05-09T12:50:41.427227", "TICKER": "MSFT"}
{"EVENT_TIME": "2018-05-09T12:50:41.520549", "TICKER": "INTC"}
{"EVENT_TIME": "2018-05-09T12:50:41.610145", "TICKER": "MSFT"}
{"EVENT_TIME": "2018-05-09T12:50:41.704395", "TICKER": "AAPL"}
...
```

You then create an Kinesis Data Analytics application on 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 two columns (`EVENT_TIME` and `TICKER`) as shown.



Then, you use the application code with SQL functions to convert the `EVENT_TIME` time stamp field in various ways. You then insert the resulting data into another in-application stream, as shown in the following screenshot:



Step 1: Create a Kinesis Data Stream

Create an Amazon Kinesis data stream and populate it with event time and ticker records as follows:

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and create a stream with one shard.
4. Run the following Python code to populate the stream with sample data. This simple code continuously writes a record with a random ticker symbol and the current time stamp to the stream.

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
```

```
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Amazon Kinesis Data Analytics Application

Create an application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**, type an application name, and choose **Create application**.
3. On the application details page, choose **Connect streaming data** to connect to the source.
4. On the **Connect to source** page, do the following:

    a. Choose the stream that you created in the preceding section.

    b. Choose to create an IAM role.

    c. Choose **Discover schema**. Wait for the console to show the inferred schema and the sample records that are used to infer the schema for the in-application stream created. The inferred schema has two columns.

    d. Choose **Edit Schema**. Change the **Column type** of the **EVENT_TIME** column to `TIMESTAMP`.

    e. Choose **Save schema and update stream samples**. After the console saves the schema, choose **Exit**.

    f. Choose **Save and continue**.

5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.
6. In the SQL editor, write the application code and verify the results as follows:

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

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

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM
    TICKER,
    EVENT_TIME,
    EVENT_TIME - INTERVAL '5' MINUTE,
    UNIX_TIMESTAMP(EVENT_TIME),
    TIMESTAMP_TO_CHAR('yyyy-MM-dd hh:mm:ss', EVENT_TIME),
    EXTRACT(SECOND FROM EVENT_TIME)
FROM "SOURCE_SQL_STREAM_001"
```

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

Example: Transforming Multiple Data Types

A common requirement in extract, transform, and load (ETL) applications is to process multiple record types on a streaming source. You can create Kinesis Data Analytics applications to process these kinds of streaming sources. The process is as follows:

1. First, you map the streaming source to an in-application input stream, similar to all other Kinesis Data Analytics applications.
2. Then, in your application code, you write SQL statements to retrieve rows of specific types from the in-application input stream. You then insert them into 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`). These are stock orders and corresponding trades. For each order, there can be zero or more trades. Example records of each type are shown following:

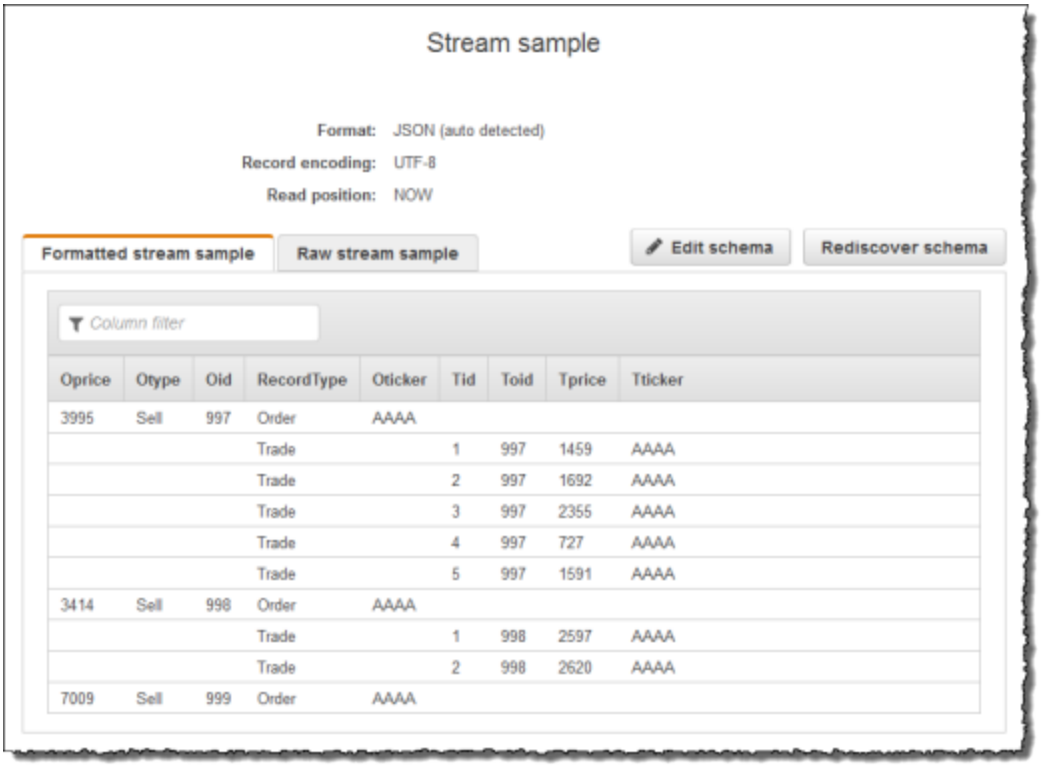
Order record

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

Trade record

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

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

1. Filter order and trade rows into separate in-application streams:

   a. Filter the order records in the `SOURCE_SQL_STREAM_001`, and save the orders in the `Order_Stream`.

```
--Create Order_Stream.
CREATE OR REPLACE STREAM "Order_Stream"
           (
            order_id     integer,
            order_type   varchar(10),
            ticker       varchar(4),
            order_price  DOUBLE,
            record_type  varchar(10)
            );

CREATE OR REPLACE PUMP "Order_Pump" AS
   INSERT INTO "Order_Stream"
      SELECT STREAM oid, otype,oticker, oprice, recordtype
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  recordtype = 'Order';
```

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

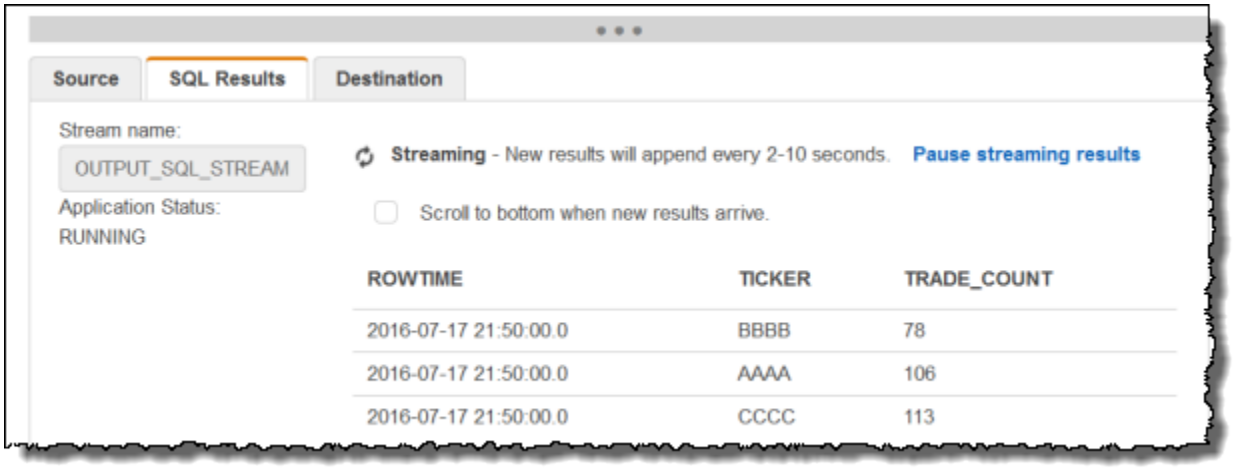
```
--Create Trade_Stream.
CREATE OR REPLACE STREAM "Trade_Stream"
           (trade_id     integer,
            order_id     integer,
            trade_price  DOUBLE,
            ticker       varchar(4),
            record_type  varchar(10)
            );

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';
```

2. 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 OPUT_SQL_STREAM.

CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
            ticker  varchar(4),
            trade_count   integer
            );

CREATE OR REPLACE PUMP "Output_Pump" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM ticker, count(*) as trade_count
      FROM   "Trade_Stream"
      GROUP BY ticker,
               FLOOR("Trade_Stream".ROWTIME TO MINUTE);
```

You see the result, as shown following:



Topics

- Step 1: Prepare the Data
- Step 2: Create the Application

Next Step

Step 1: Prepare the Data

Step 1: Prepare the Data

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 that you create in the next step.

Topics

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. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
- **Using the AWS CLI** – Use the following Kinesis `create-stream` AWS CLI command to create the stream:

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

Step 1.2: Populate the Streaming Source

Run the following Python script to populate sample records on the `OrdersAndTradesStream`. If you created the stream with 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 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
import random
import boto3

STREAM_NAME = "OrdersAndTradesStream"
PARTITION_KEY = "partition_key"


def get_order(order_id, ticker):
    return {
        "RecordType": "Order",
        "Oid": order_id,
        "Oticker": ticker,
        "Oprice": random.randint(500, 10000),
        "Otype": "Sell",
    }


def get_trade(order_id, trade_id, ticker):
    return {
        "RecordType": "Trade",
        "Tid": trade_id,
        "Toid": order_id,
        "Tticker": ticker,
        "Tprice": random.randint(0, 3000),
    }


def generate(stream_name, kinesis_client):
    order_id = 1
    while True:
        ticker = random.choice(["AAAA", "BBBB", "CCCC"])
        order = get_order(order_id, ticker)
        print(order)
        kinesis_client.put_record(
```

```
            StreamName=stream_name, Data=json.dumps(order),
 PartitionKey=PARTITION_KEY
        )
        for trade_id in range(1, random.randint(0, 6)):
            trade = get_trade(order_id, trade_id, ticker)
            print(trade)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(trade),
                PartitionKey=PARTITION_KEY,
            )
        order_id += 1


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Next Step

Step 2: Create the Application

Step 2: Create the Application

In this section, you create an 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. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**. This example uses the application name **`ProcessMultipleRecordTypes`**.
3. On the application details page, choose **Connect streaming data** to connect to the source.
4. On the **Connect to source** page, do the following:
   a. Choose the stream that you created in Step 1: Prepare the Data.
   b. Choose to create an IAM role.
   c. Wait for the console to show the inferred schema and samples records that are 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, start application** 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"
           (
            "order_id"     integer,
            "order_type"   varchar(10),
            "ticker"       varchar(4),
            "order_price"  DOUBLE,
            "record_type"  varchar(10)
            );

CREATE OR REPLACE PUMP "Order_Pump" AS
   INSERT INTO "Order_Stream"
      SELECT STREAM "Oid", "Otype","Oticker", "Oprice", "RecordType"
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  "RecordType" = 'Order';
--*****************************************
--Create Trade_Stream.
CREATE OR REPLACE STREAM "Trade_Stream"
           ("trade_id"     integer,
            "order_id"     integer,
            "trade_price"  DOUBLE,
            "ticker"       varchar(4),
            "record_type"  varchar(10)
            );

CREATE OR REPLACE PUMP "Trade_Pump" AS
   INSERT INTO "Trade_Stream"
      SELECT STREAM "Tid", "Toid", "Tprice", "Tticker", "RecordType"
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  "RecordType" = 'Trade';
--*************************************************************
--do some analytics on the Trade_Stream and Order_Stream.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
            "ticker"  varchar(4),
            "trade_count"   integer
```

```
            );

CREATE OR REPLACE PUMP "Output_Pump" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM "ticker", count(*) as trade_count
      FROM   "Trade_Stream"
      GROUP BY "ticker",
                FLOOR("Trade_Stream".ROWTIME TO MINUTE);
```

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

Next Step

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

Examples: Windows and Aggregation

This section provides examples of Amazon Kinesis Data Analytics applications that use windowed and aggregate queries. (For more information, see Windowed Queries.) Each example provides step-by-step instructions and example code for setting up the Kinesis Data Analytics application.

Topics

- Example: Stagger Window
- Example: Tumbling Window Using ROWTIME
- Example: Tumbling Window Using an Event Timestamp
- Example: Retrieving the Most Frequently Occurring Values (TOP_K_ITEMS_TUMBLING)
- Example: Aggregating Partial Results from a Query

Example: Stagger Window

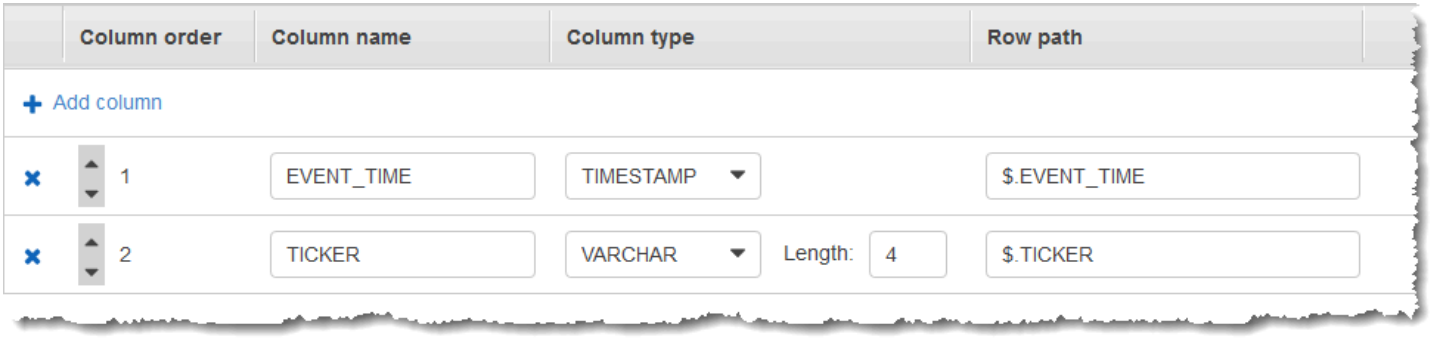
When a windowed query processes separate windows for each unique partition key, starting when data with the matching key arrives, the window is referred to as a *stagger window*. For details, see Stagger Windows. This Amazon Kinesis Data Analytics example uses the EVENT_TIME and TICKER

columns to create stagger windows. The source stream contains groups of six records with identical EVENT_TIME and TICKER values that arrive within in a one-minute period, but not necessarily with the same minute value (for example, `18:41:xx`).

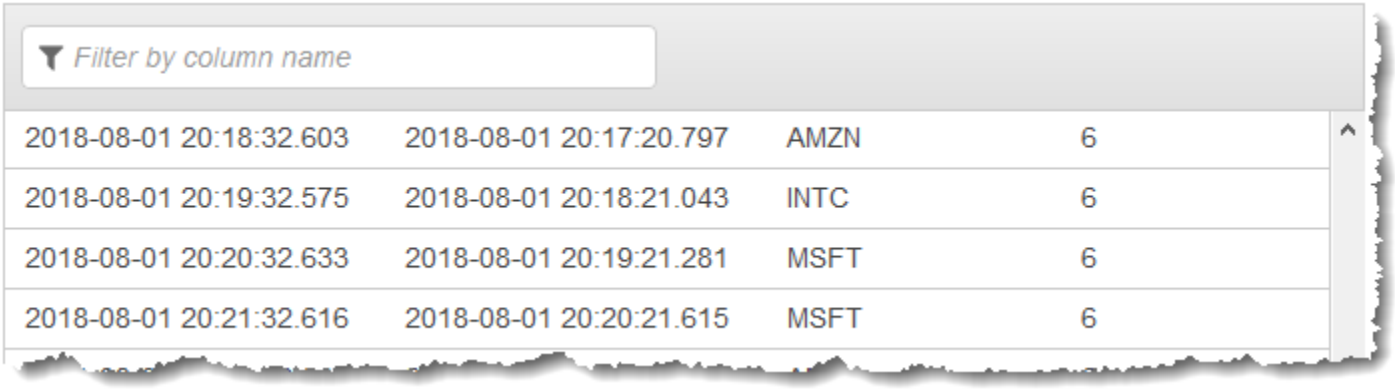
In this example, you write the following records to a Kinesis data stream at the following times. The script does not write the times to the stream, but the time that the record is ingested by the application is written to the `ROWTIME` field:

```
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:17:30
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:17:40
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:17:50
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:18:00
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:18:10
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:18:21
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:18:31
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:18:41
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:18:51
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:19:01
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:19:11
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:19:21
...
```

You then create a Kinesis Data Analytics application in the AWS Management 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 two columns (`EVENT_TIME` and `TICKER`) as shown following.



You use the application code with the `COUNT` function to create a windowed aggregation of the data. Then you insert the resulting data into another in-application stream, as shown in the following screenshot:



In the following procedure, you create a Kinesis Data Analytics application that aggregates values in the input stream in a stagger window based on EVENT_TIME and TICKER.

Topics

- Step 1: Create a Kinesis Data Stream
- Step 2: Create the Kinesis Data Analytics Application

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and then create a stream with one shard. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
4. To write records to a Kinesis data stream in a production environment, we recommend using either the Kinesis Producer Library or Kinesis Data Streams API. For simplicity, this example uses the following Python script to generate records. Run the code to populate the sample ticker records. This simple code continuously writes a group of six records with the same random `EVENT_TIME` and ticker symbol to the stream, over the course of one minute. Keep the script running so that you can generate the application schema in a later step.

```
import datetime
import json
import random
```

```
import time
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    event_time = datetime.datetime.utcnow() - datetime.timedelta(seconds=10)
    return {
        "EVENT_TIME": event_time.isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        # Send six records, ten seconds apart, with the same event time and ticker
        for _ in range(6):
            print(data)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(data),
                PartitionKey="partitionkey",
            )
            time.sleep(10)


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Kinesis Data Analytics Application

Create a Kinesis Data Analytics application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**, type an application name, and choose **Create application**.
3. On the application details page, choose **Connect streaming data** to connect to the source.
4. On the **Connect to source** page, do the following:

a. Choose the stream that you created in the preceding section.

b. Choose **Discover Schema**. Wait for the console to show the inferred schema and samples records that are used to infer the schema for the in-application stream created. The inferred schema has two columns.

c. Choose **Edit Schema**. Change the **Column type** of the **EVENT_TIME** column to `TIMESTAMP`.

d. Choose **Save schema and update stream samples**. After the console saves the schema, choose **Exit**.

e. Choose **Save and continue**.

5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.

6. In the SQL editor, write the application code, and verify the results as follows:

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

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

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

b. Choose **Save and run SQL**.

On the **Real-time analytics** tab, you can see all the in-application streams that the application created and verify the data.

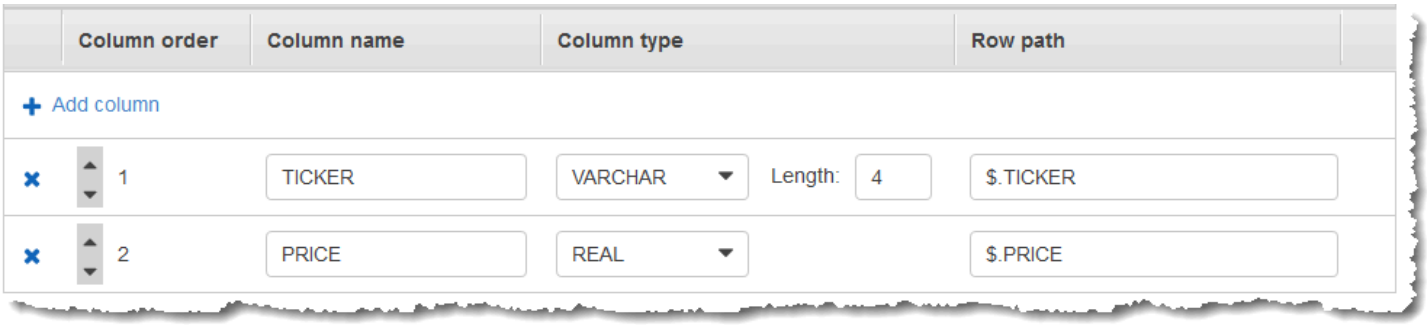
Example: Tumbling Window Using ROWTIME

When a windowed query processes each window in a non-overlapping manner, the window is referred to as a *tumbling window*. For details, see Tumbling Windows (Aggregations Using GROUP BY). This Amazon Kinesis Data Analytics example uses the `ROWTIME` column to create tumbling windows. The `ROWTIME` column represents the time the record was read by the application.

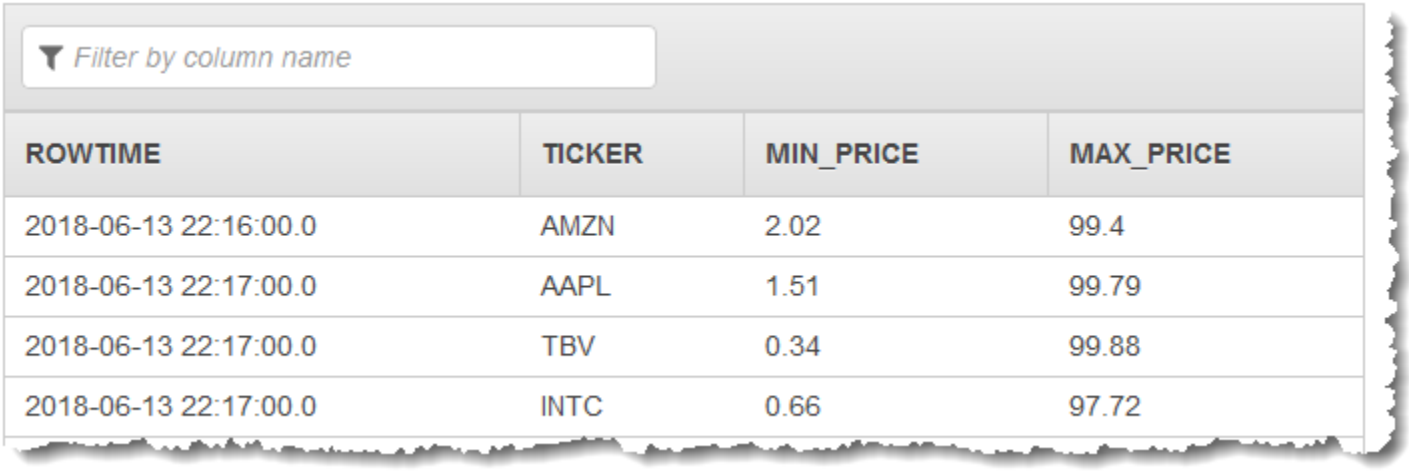
In this example, you write the following records to a Kinesis data stream.

```
{"TICKER": "TBV", "PRICE": 33.11}
{"TICKER": "INTC", "PRICE": 62.04}
{"TICKER": "MSFT", "PRICE": 40.97}
{"TICKER": "AMZN", "PRICE": 27.9}
...
```

You then create a Kinesis Data Analytics application in the AWS Management 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 two columns (`TICKER` and `PRICE`) as shown following.



You use the application code with the `MIN` and `MAX` functions to create a windowed aggregation of the data. Then you insert the resulting data into another in-application stream, as shown in the following screenshot:



In the following procedure, you create a Kinesis Data Analytics application that aggregates values in the input stream in a tumbling window based on ROWTIME.

Topics

- Step 1: Create a Kinesis Data Stream
- Step 2: Create the Kinesis Data Analytics Application

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and then create a stream with one shard. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
4. To write records to a Kinesis data stream in a production environment, we recommend using either the Kinesis Client Library or Kinesis Data Streams API. For simplicity, this example uses the following Python script to generate records. Run the code to populate the sample ticker records. This simple code continuously writes a random ticker record to the stream. Keep the script running so that you can generate the application schema in a later step.

```
import datetime
import json
```

```
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Kinesis Data Analytics Application

Create a Kinesis Data Analytics application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**, enter an application name, and choose **Create application**.
3. On the application details page, choose **Connect streaming data** to connect to the source.
4. On the **Connect to source** page, do the following:

    a. Choose the stream that you created in the preceding section.

b. Choose **Discover Schema**. Wait for the console to show the inferred schema and samples records that are used to infer the schema for the in-application stream created. The inferred schema has two columns.

c. Choose **Save schema and update stream samples**. After the console saves the schema, choose **Exit**.

d. Choose **Save and continue**.

5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.

6. In the SQL editor, write the application code, and verify the results as follows:

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

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (TICKER VARCHAR(4), MIN_PRICE
 REAL, MAX_PRICE REAL);

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

b. Choose **Save and run SQL**.

On the **Real-time analytics** tab, you can see all the in-application streams that the application created and verify the data.

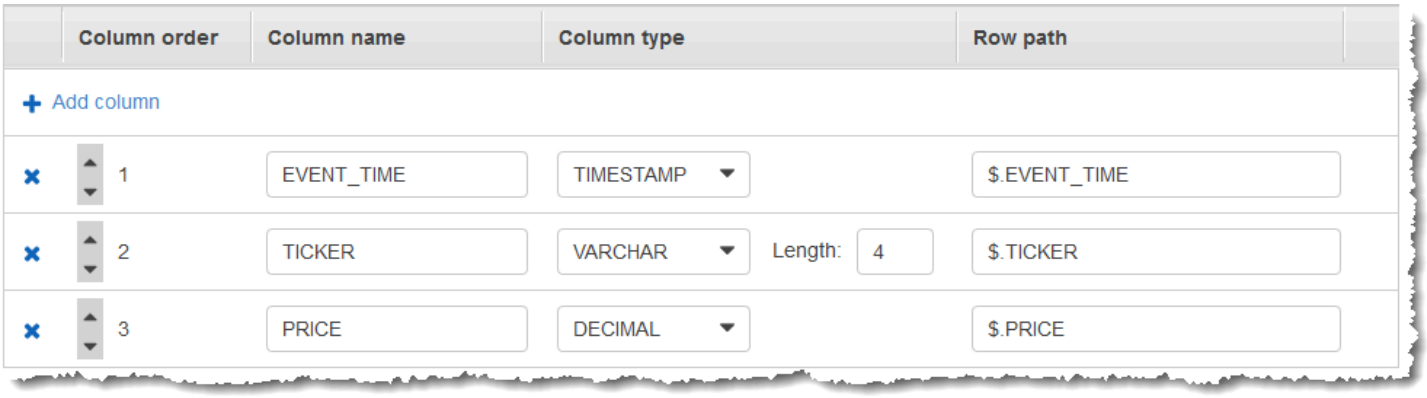
Example: Tumbling Window Using an Event Timestamp

When a windowed query processes each window in a non-overlapping manner, the window is referred to as a *tumbling window*. For details, see Tumbling Windows (Aggregations Using GROUP BY). This Amazon Kinesis Data Analytics example demonstrates a tumbling window that uses an event timestamp, which is a user-created timestamp that is included in the streaming data. It uses this approach rather than just using ROWTIME, which is a timestamp that Kinesis Data Analytics creates when the application receives the record. You would use an event timestamp in the streaming data if you want to create an aggregation based on when an event occurred, rather than when it was received by the application. In this example, the `ROWTIME` value triggers the aggregation every minute, and the records are aggregated by both `ROWTIME` and the included event time.

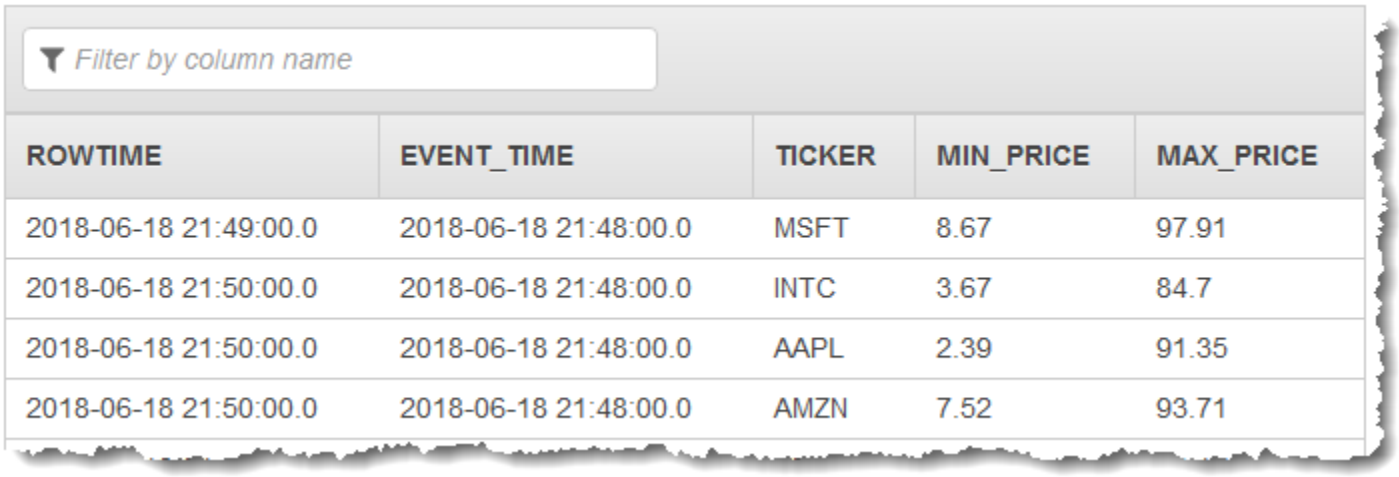
In this example, you write the following records to an Amazon Kinesis stream. The EVENT_TIME value is set to 5 seconds in the past, to simulate processing and transmission lag that might create a delay from when the event occurred, to when the record is ingested into Kinesis Data Analytics.

```
{"EVENT_TIME": "2018-06-13T14:11:05.766191", "TICKER": "TBV", "PRICE": 43.65}
{"EVENT_TIME": "2018-06-13T14:11:05.848967", "TICKER": "AMZN", "PRICE": 35.61}
{"EVENT_TIME": "2018-06-13T14:11:05.931871", "TICKER": "MSFT", "PRICE": 73.48}
{"EVENT_TIME": "2018-06-13T14:11:06.014845", "TICKER": "AMZN", "PRICE": 18.64}
...
```

You then create a Kinesis Data Analytics application in the AWS Management 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 three columns (EVENT_TIME, TICKER, and PRICE) as shown following.



You use the application code with the MIN and MAX functions to create a windowed aggregation of the data. Then you insert the resulting data into another in-application stream, as shown in the following screenshot:



In the following procedure, you create a Kinesis Data Analytics application that aggregates values in the input stream in a tumbling window based on an event time.

Topics

- Step 1: Create a Kinesis Data Stream
- Step 2: Create the Kinesis Data Analytics Application

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and then create a stream with one shard. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
4. To write records to a Kinesis data stream in a production environment, we recommend using either the Kinesis Client Library or Kinesis Data Streams API. For simplicity, this example uses the following Python script to generate records. Run the code to populate the sample ticker records. This simple code continuously writes a random ticker record to the stream. Keep the script running so that you can generate the application schema in a later step.

```
import datetime
import json
```

```
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Kinesis Data Analytics Application

Create a Kinesis Data Analytics application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**, enter an application name, and choose **Create application**.
3. On the application details page, choose **Connect streaming data** to connect to the source.
4. On the **Connect to source** page, do the following:

    a. Choose the stream that you created in the preceding section.

b. Choose **Discover Schema**. Wait for the console to show the inferred schema and samples records that are used to infer the schema for the in-application stream created. The inferred schema has three columns.

c. Choose **Edit Schema**. Change the **Column type** of the **EVENT_TIME** column to `TIMESTAMP`.

d. Choose **Save schema and update stream samples**. After the console saves the schema, choose **Exit**.

e. Choose **Save and continue**.

5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.

6. In the SQL editor, write the application code, and verify the results as follows:

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

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (EVENT_TIME timestamp, TICKER
 VARCHAR(4), min_price REAL, max_price REAL);

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

b. Choose **Save and run SQL**.

On the **Real-time analytics** tab, you can see all the in-application streams that the application created and verify the data.

Example: Retrieving the Most Frequently Occurring Values (TOP_K_ITEMS_TUMBLING)

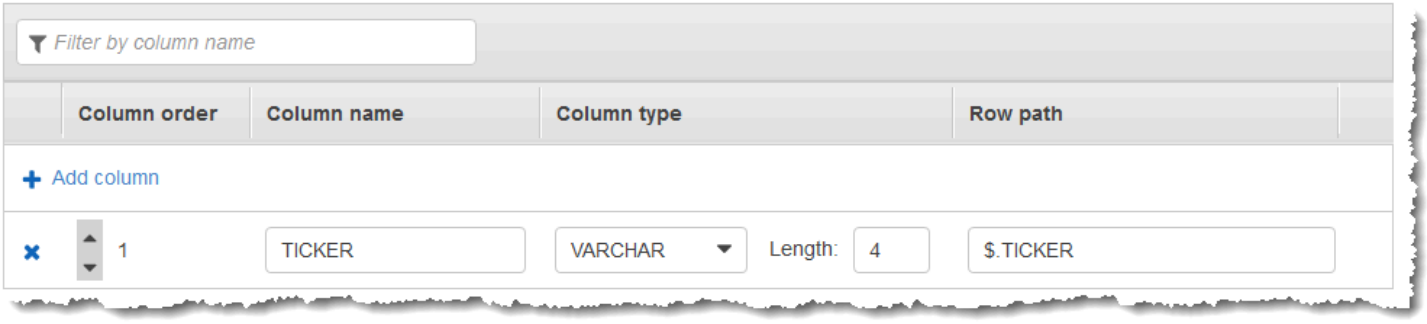
This Amazon Kinesis Data Analytics example demonstrates how to use the `TOP_K_ITEMS_TUMBLING` function to retrieve the most frequently occurring values in a tumbling window. For more information, see `TOP_K_ITEMS_TUMBLING` function in the *Amazon Managed Service for Apache Flink SQL Reference*.

The `TOP_K_ITEMS_TUMBLING` function is useful when aggregating over tens or hundreds of thousands of keys, and you want to reduce your resource usage. The function produces the same result as aggregating with `GROUP BY` and `ORDER BY` clauses.

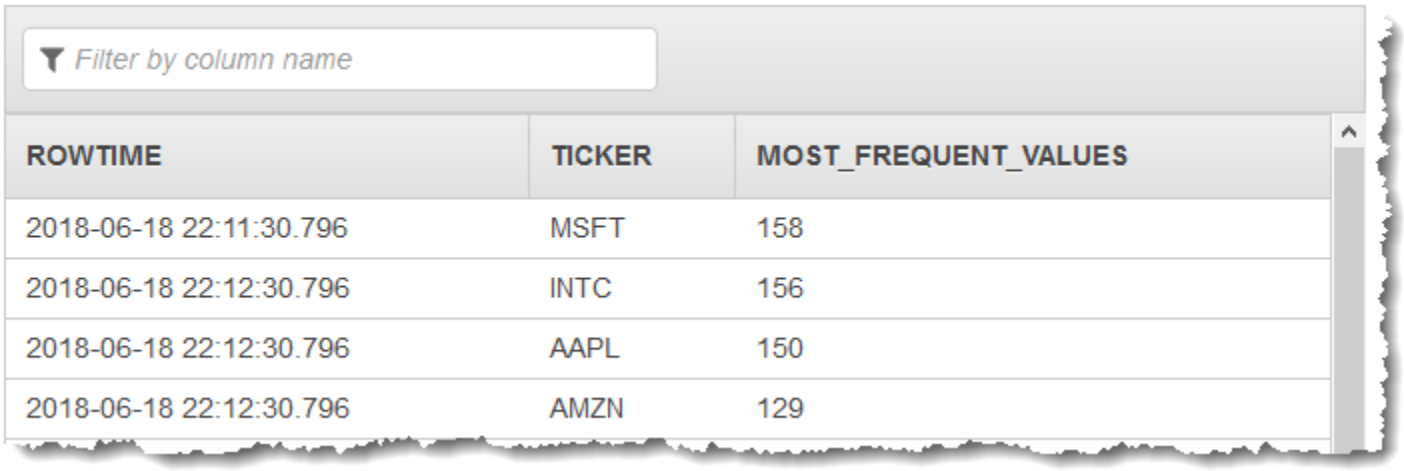
In this example, you write the following records to an Amazon Kinesis data stream:

```
{"TICKER": "TBV"}
{"TICKER": "INTC"}
{"TICKER": "MSFT"}
{"TICKER": "AMZN"}
...
```

You then create a Kinesis Data Analytics application in the AWS Management 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 (`TICKER`) as shown following.



You use the application code with the `TOP_K_VALUES_TUMBLING` function to create a windowed aggregation of the data. Then you insert the resulting data into another in-application stream, as shown in the following screenshot:



In the following procedure, you create a Kinesis Data Analytics application that retrieves the most frequently occurring values in the input stream.

Topics

- Step 1: Create a Kinesis Data Stream
- Step 2: Create the Kinesis Data Analytics Application

Step 1: Create a Kinesis Data Stream

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

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Streams** in the navigation pane.
3. Choose **Create Kinesis stream**, and then create a stream with one shard. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
4. To write records to a Kinesis data stream in a production environment, we recommend using either the Kinesis Client Library or Kinesis Data Streams API. For simplicity, this example uses the following Python script to generate records. Run the code to populate the sample ticker records. This simple code continuously writes a random ticker record to the stream. Leave the script running so that you can generate the application schema in a later step.

```
import datetime
import json
```

```
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Step 2: Create the Kinesis Data Analytics Application

Create a Kinesis Data Analytics application as follows:

1. Open the Managed Service for Apache Flink console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**, type an application name, and choose **Create application**.
3. On the application details page, choose **Connect streaming data** to connect to the source.
4. On the **Connect to source** page, do the following:

    a. Choose the stream that you created in the preceding section.

b. Choose **Discover Schema**. Wait for the console to show the inferred schema and samples records that are used to infer the schema for the in-application stream created. The inferred schema has one column.

c. Choose **Save schema and update stream samples**. After the console saves the schema, choose **Exit**.

d. Choose **Save and continue**.

5. On the application details page, choose **Go to SQL editor**. To start the application, choose **Yes, start application** in the dialog box that appears.

6. In the SQL editor, write the application code, and verify the results as follows:

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

```
CREATE OR REPLACE STREAM DESTINATION_SQL_STREAM (
  "TICKER" VARCHAR(4),
  "MOST_FREQUENT_VALUES" BIGINT
);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM *
        FROM TABLE (TOP_K_ITEMS_TUMBLING(
            CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"),
            'TICKER',         -- name of column in single quotes
            5,                      -- number of the most frequently occurring
 values
            60                      -- tumbling window size in seconds
            )
        );
```

b. Choose **Save and run SQL**.

On the **Real-time analytics** tab, you can see all the in-application streams that the application created and verify the data.

Example: Aggregating Partial Results from a Query

If an Amazon Kinesis data stream contains records that have an event time that does not exactly match ingestion time, a selection of results in a tumbling window contains records that arrived, but did not necessarily occur, within the window. In this case, the tumbling window contains only

a partial set of the results that you want. There are several approaches that you can use to correct this issue:

- Use a tumbling window only, and aggregate partial results in post processing through a database or data warehouse using upserts. This approach is efficient in processing an application. It handles the late data indefinitely for aggregate operators (`sum`, `min`, `max`, and so on). The downside to this approach is that you must develop and maintain additional application logic in the database layer.
- Use a tumbling and sliding window, which produces partial results early, but also continues to produce complete results over the sliding window period. This approach handles late data with an overwrite instead of an upsert so that no additional application logic needs to be added in the database layer. The downside to this approach is that it uses more Kinesis processing units (KPUs) and still produces two results, which might not work for some use cases.

For more information about tumbling and sliding windows, see Windowed Queries.

In the following procedure, the tumbling window aggregation produces two partial results (sent to the `CALC_COUNT_SQL_STREAM` in-application stream) that must be combined to produce a final result. The application then produces a second aggregation (sent to the `DESTINATION_SQL_STREAM` in-application stream) that combines the two partial results.

To create an application that aggregates partial results using an event time

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Analytics** in the navigation pane. Create a Kinesis Data Analytics application as described in the Getting Started with Amazon Kinesis Data Analytics for SQL Applications tutorial.
3. In the SQL editor, replace the application code with the following:

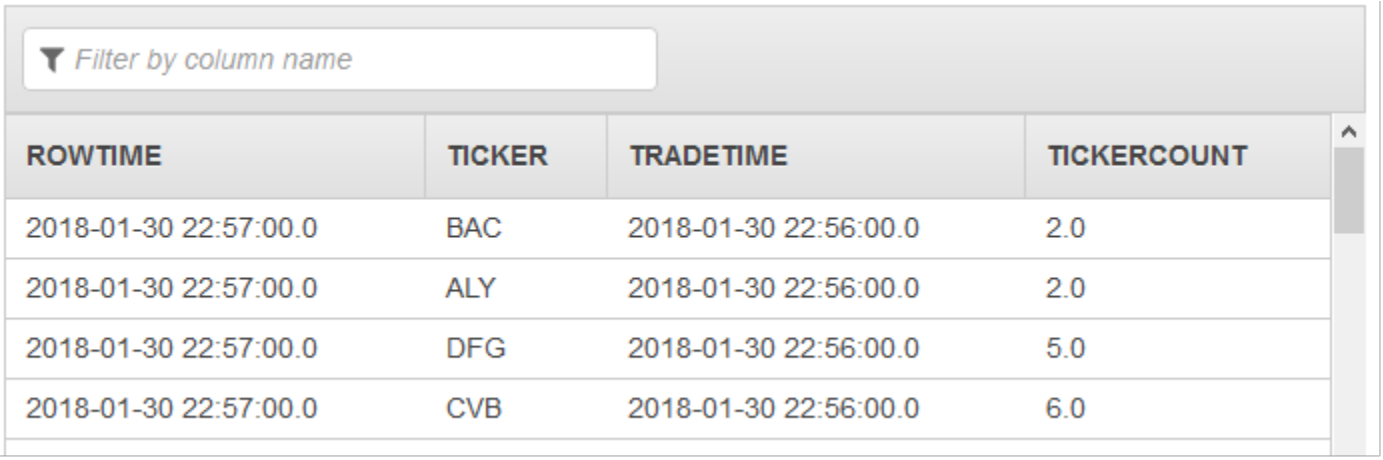
```
CREATE OR REPLACE STREAM "CALC_COUNT_SQL_STREAM"
    (TICKER      VARCHAR(4),
    TRADETIME   TIMESTAMP,
    TICKERCOUNT       DOUBLE);

CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
    (TICKER      VARCHAR(4),
    TRADETIME   TIMESTAMP,
```

```
    TICKERCOUNT       DOUBLE);

CREATE PUMP "CALC_COUNT_SQL_PUMP_001" AS
    INSERT INTO "CALC_COUNT_SQL_STREAM" ("TICKER","TRADETIME", "TICKERCOUNT")
    SELECT STREAM
        "TICKER_SYMBOL",
        STEP("SOURCE_SQL_STREAM_001"."ROWTIME" BY INTERVAL '1' MINUTE) as
 "TradeTime",
        COUNT(*) AS "TickerCount"
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY
        STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE),
        STEP("SOURCE_SQL_STREAM_001"."APPROXIMATE_ARRIVAL_TIME" BY INTERVAL '1'
 MINUTE),
        TICKER_SYMBOL;

CREATE PUMP "AGGREGATED_SQL_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM" ("TICKER","TRADETIME", "TICKERCOUNT")
    SELECT STREAM
        "TICKER",
        "TRADETIME",
        SUM("TICKERCOUNT") OVER W1 AS "TICKERCOUNT"
    FROM "CALC_COUNT_SQL_STREAM"
    WINDOW W1 AS (PARTITION BY "TRADETIME" RANGE INTERVAL '10' MINUTE PRECEDING);
```

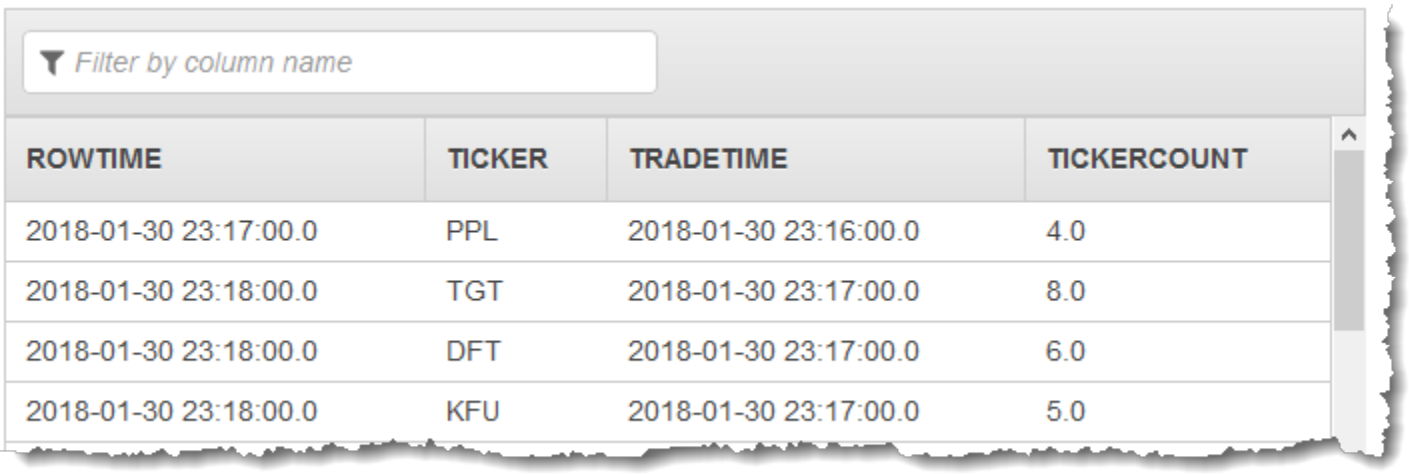
The SELECT statement in the application code filters rows in the SOURCE_SQL_STREAM_001 for stock price changes greater than 1 percent and inserts those rows into another in-application stream CHANGE_STREAM using a pump.

4. Choose **Save and run SQL**.

The first pump outputs a stream to CALC_COUNT_SQL_STREAM similar to the following. Note that the result set is incomplete:



The second pump then outputs a stream to `DESTINATION_SQL_STREAM` that contains the complete result set:



Examples: Joins

This section provides examples of Kinesis Data Analytics applications that use join queries. Each example provides step-by-step instructions for setting up and testing your Kinesis Data Analytics application.

Topics

- Example: Adding Reference Data to a Kinesis Data Analytics Application

Example: Adding Reference Data to a Kinesis Data Analytics Application

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

- Amazon Kinesis Data Analytics for SQL Applications: How It Works
- Configuring Application Input

In this exercise, you add reference data to the application you created in the Kinesis Data Analytics Getting Started exercise. The reference data provides the company name for each ticker symbol; for example:

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

First, complete the steps in the Getting Started exercise to create a starter application. Then follow these steps to set up and add reference data to your application:

1. **Prepare the data**
   - Store the preceding reference data as an object in Amazon Simple Storage Service (Amazon S3).
   - Create an IAM role that Kinesis Data Analytics can assume to read the Amazon S3 object on your behalf.
2. **Add the reference data source to your application.**

   Kinesis Data Analytics reads the Amazon S3 object and creates an in-application reference table that you can query in your application code.
3. **Test the code.**

   In your application code, you write a join query to join the in-application stream with the in-application reference table, to get the company name for each ticker symbol.

Topics

- Step 1: Prepare

- Step 2: Add the Reference Data Source to the Application Configuration
- Step 3: Test: Query the In-Application Reference Table

Step 1: Prepare

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

Store Reference Data as an Amazon S3 Object

In this step, you store the sample reference data as an Amazon S3 object.

1. Open a text editor, add 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 User Guide*.

Create an IAM Role

Next, create an IAM role that Kinesis Data Analytics can assume and read the Amazon S3 object.

1. In AWS Identity and Access Management (IAM), create an IAM role named **`KinesisAnalytics-ReadS3Object`**. To create the role, follow the instructions in Creating a Role for an Amazon Service (AWS Management Console) in the *IAM User Guide*.

   On the IAM console, specify the following:

   - For **Select Role Type**, choose **AWS Lambda**. After creating the role, you will change the trust policy to allow Kinesis Data Analytics (not AWS Lambda) to assume the role.
   - Do not attach any policy on the **Attach Policy** page.
2. Update the IAM role policies:

a. On the IAM console, choose the role that you created.

b. On the **Trust Relationships** tab, update the trust policy to grant 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 Amazon-managed policy called **AmazonS3ReadOnlyAccess**. This grants the role permissions to read an Amazon S3 object. This policy is shown following:

JSON

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

Step 2: Add the Reference Data Source to the Application Configuration

In this step, you add a reference data source to your application configuration. To begin, you need the following information:

- Your S3 bucket name and object key name
- The IAM role Amazon Resource Name (ARN)

1. In the main page for the application, choose **Connect reference data**.
2. In the **Connect reference data source** page, choose the Amazon S3 bucket containing your reference data object, and enter the object's key name.
3. Enter **`CompanyName`** for the **In-application reference table name**.
4. In the **Access to chosen resources** section, choose **Choose from IAM roles that Kinesis Analytics can assume**, and choose the **KinesisAnalytics-ReadS3Object** IAM role you created in the previous section.
5. Choose **Discover schema**. The console detects two columns in the reference data.
6. Choose **Save and close**.

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. The result shows the company name.

1. Replace your application code with the following. The query joins the in-application input stream with the in-application reference table. The application code writes the results to another in-application stream, `DESTINATION_SQL_STREAM`.

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 "Company" varchar(20), sector VARCHAR(12), change DOUBLE, price DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
   SELECT STREAM ticker_symbol, "c"."Company", sector, change, price
   FROM "SOURCE_SQL_STREAM_001" LEFT JOIN "CompanyName" as "c"
   ON "SOURCE_SQL_STREAM_001".ticker_symbol = "c"."Ticker";
```

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

Examples: Machine Learning

This section provides examples of Amazon Kinesis Data Analytics applications that use machine learning queries. Machine learning queries perform complex analysis on data, relying on the history of the data in the stream to find unusual patterns. The examples provide step-by-step instructions to set up and test your Kinesis Data Analytics application.

Topics

Example: Detecting Data Anomalies on a Stream (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 Managed Service for Apache Flink SQL Reference*.

In this exercise, you write application code to assign an anomaly score to records on your application's streaming source. To set up the application, you do the following:

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

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

   The procedure provides a Python script for you to populate the stream. The `heartRate` values are randomly generated, with 99 percent of the records having `heartRate` values between

60 and 100, and only 1 percent of `heartRate` values between 150 and 200. Thus, the records that have `heartRate` values between 150 and 200 are anomalies.

2. **Configure input** – Using the console, you create a Kinesis Data Analytics application and configure the application input by mapping the streaming source to an in-application stream (`SOURCE_SQL_STREAM_001`). When the application starts, Kinesis Data Analytics continuously reads the streaming source and inserts records into the in-application stream.

3. **Specify application code** – The example uses the following application code:

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

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

-- Compute an anomaly score for each record in the input stream
-- using Random Cut Forest
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "TEMP_STREAM"
      SELECT STREAM "heartRate", "rateType", ANOMALY_SCORE
      FROM TABLE(RANDOM_CUT_FOREST(
              CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_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`). You use pumps to insert rows in in-application streams. For more information, see In-Application Streams and Pumps.

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

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

Topics

Next Step

Step 1: Prepare

Step 1: Prepare

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

Topics

Step 1.1: Create the Input and Output Data Streams

In this section, you create two Kinesis streams: `ExampleInputStream` and `ExampleOutputStream`. You can create these streams 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. Choose **Create data stream**. Create a stream with one shard named `ExampleInputStream`. For more information, see Create a Stream in the *Amazon Kinesis Data Streams Developer Guide*.
  3. Repeat the previous step, creating a stream with one shard named `ExampleOutputStream`.

- **To use the AWS CLI**

  1. Use the following Kinesis `create-stream` AWS 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`. This command creates the second stream that the application uses 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 these records to the `ExampleInputStream` stream.

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

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.

```
from enum import Enum
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


class RateType(Enum):
    normal = "NORMAL"
    high = "HIGH"


def get_heart_rate(rate_type):
    if rate_type == RateType.normal:
        rate = random.randint(60, 100)
    elif rate_type == RateType.high:
        rate = random.randint(150, 200)
    else:
        raise TypeError
    return {"heartRate": rate, "rateType": rate_type.value}


def generate(stream_name, kinesis_client, output=True):
    while True:
        rnd = random.random()
        rate_type = RateType.high if rnd < 0.01 else RateType.normal
        heart_rate = get_heart_rate(rate_type)
        if output:
            print(heart_rate)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(heart_rate),
            PartitionKey="partitionkey",
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Next Step

Step 2: Create an Application

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 data stream that you created in the section called "Step 1: Prepare" as the streaming source.
- Use the **Anomaly Detection** template on the console.

To create an application

1. Follow steps 1, 2, and 3 in the Kinesis Data Analytics **Getting Started** exercise (see Step 3.1: Create an Application).

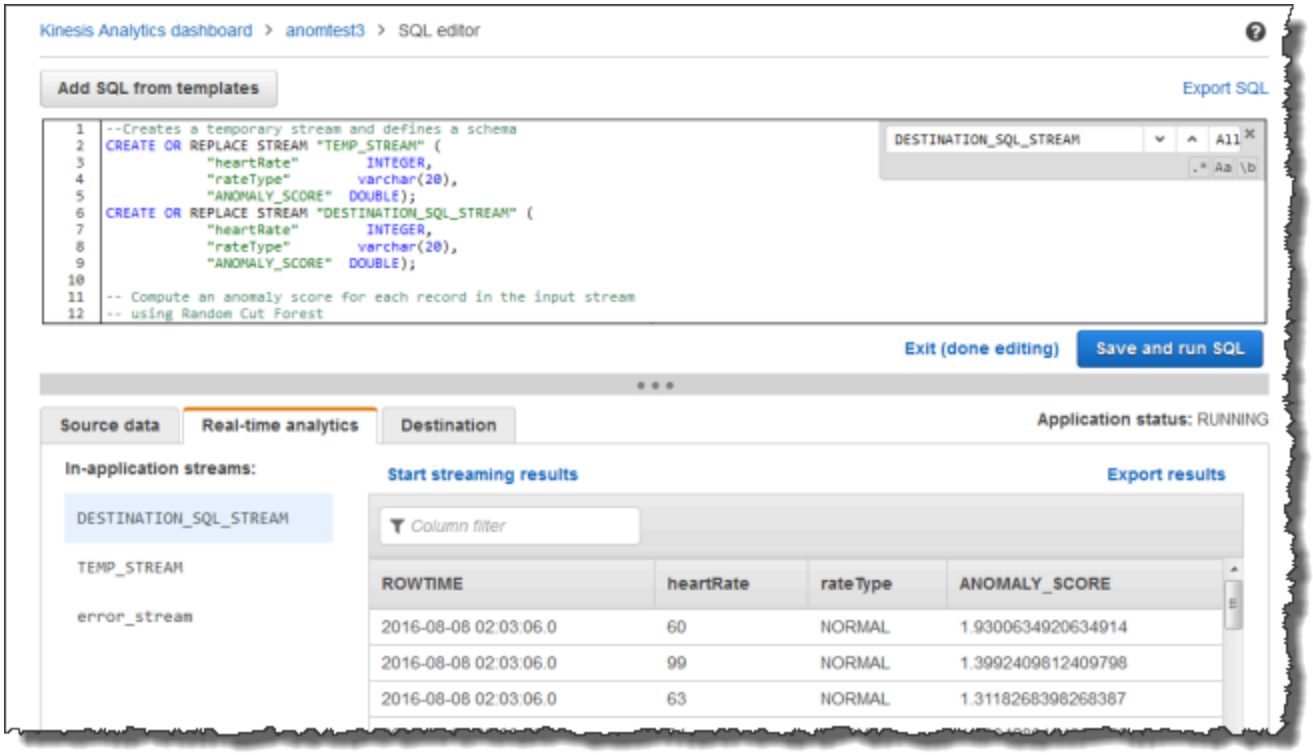
   - In the source configuration, do the following:
     - Specify the streaming source that you created in the preceding section.
     - After the console infers the schema, edit the schema, and set the `heartRate` column type to `INTEGER`.

       Most of the heart rate values are normal, and the discovery process will most likely assign the `TINYINT` type to this column. But a small percentage of the values show a high heart rate. If these high values don't fit in the `TINYINT` type, Kinesis Data Analytics sends these rows to an error stream. Update the data type to `INTEGER` so that it can accommodate all the generated heart rate data.
   - Use the **Anomaly Detection** template on the console. You then update the template code to provide the appropriate column name.
2. Update the application code by providing column names. The resulting application code is shown following (paste this code into the SQL editor):

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

--Creates another stream for application output.
```

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
         "heartRate"        INTEGER,
         "rateType"         varchar(20),
         "ANOMALY_SCORE"    DOUBLE);

-- Compute an anomaly score for each record in the input stream
-- using Random Cut Forest
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "TEMP_STREAM"
      SELECT STREAM "heartRate", "rateType", ANOMALY_SCORE
      FROM TABLE(RANDOM_CUT_FOREST(
              CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_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 the results on the Kinesis Data Analytics console:



Next Step

Step 3: Configure Application Output

Step 3: Configure Application Output

After completing the section called "Step 2: Create an Application", you have application code that is reading heart rate data from a streaming source and assigning an anomaly score to each.

You can now send the application results from the in-application stream to an external destination, which is another Kinesis data stream (`OutputStreamTestingAnomalyScores`). You can analyze the anomaly scores and determine which heart rate is anomalous. You can then extend this application further to generate alerts.

Follow these steps to configure application output:

1. Open the Amazon Kinesis Data Analytics console. In the SQL editor, choose either **Destination** or **Add a destination** in the application dashboard.
2. On the **Connect to destination** page, choose the `OutputStreamTestingAnomalyScores` stream that 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 Preprocessing Data Using a Lambda Function. If you don't set alerts, you can review the records that Kinesis Data Analytics writes to the external destination, which is the Kinesis data stream `OutputStreamTestingAnomalyScores`, as described in Step 4: Verify Output.

Next Step

Step 4: Verify Output

Step 4: Verify Output

After configuring the application output in the section called "Step 3: Configure Application Output", use the following AWS CLI commands to read records in the destination stream that is written by the application:

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

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

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

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

Copy the shard iterator value.

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

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

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

Example: 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 the *Amazon Managed Service for Apache Flink SQL Reference*.

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.

Topics

First Step

Step 1: Prepare the Data

Step 1: Prepare the Data

Before you create an Amazon Kinesis Data Analytics application for this example, 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.

Topics

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. Choose **Data Streams** in the navigation pane. Then 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 that 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.

```
from enum import Enum
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


class PressureType(Enum):
    low = "LOW"
    normal = "NORMAL"
    high = "HIGH"


def get_blood_pressure(pressure_type):
    pressure = {"BloodPressureLevel": pressure_type.value}
    if pressure_type == PressureType.low:
        pressure["Systolic"] = random.randint(50, 80)
        pressure["Diastolic"] = random.randint(30, 50)
    elif pressure_type == PressureType.normal:
        pressure["Systolic"] = random.randint(90, 120)
        pressure["Diastolic"] = random.randint(60, 80)
    elif pressure_type == PressureType.high:
        pressure["Systolic"] = random.randint(130, 200)
        pressure["Diastolic"] = random.randint(90, 150)
    else:
        raise TypeError
    return pressure


def generate(stream_name, kinesis_client):
    while True:
        rnd = random.random()
```

```
        pressure_type = (
            PressureType.low
            if rnd < 0.005
            else PressureType.high
            if rnd > 0.995
            else PressureType.normal
        )
        blood_pressure = get_blood_pressure(pressure_type)
        print(blood_pressure)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(blood_pressure),
            PartitionKey="partitionkey",
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

Next Step

Step 2: Create an Analytics Application

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 that you created as the streaming source in the section called "Step 1: Prepare the Data". You then run application code that uses the `RANDOM_CUT_FOREST_WITH_EXPLANATION` function.

To create an application

1. Open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Analytics** in the navigation pane, and then choose **Create application**.
3. Provide an application name and description (optional), and choose **Create application**.
4. Choose **Connect streaming data**, 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**.

```
--Creates a temporary stream.
CREATE OR REPLACE STREAM "TEMP_STREAM" (
         "Systolic"                 INTEGER,
         "Diastolic"                INTEGER,
         "BloodPressureLevel"       varchar(20),
         "ANOMALY_SCORE"            DOUBLE,
         "ANOMALY_EXPLANATION"      varchar(512));

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

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

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

Next Step

Step 3: Examine the Results

Step 3: Examine the Results

When you run the SQL code for this example, 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:

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

- 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: Detecting Hotspots on a Stream (HOTSPOTS Function)

Amazon Kinesis Data Analytics provides the `HOTSPOTS` function, which can locate and return information about relatively dense regions in your data. For more information, see HOTSPOTS in the *Amazon Managed Service for Apache Flink SQL Reference*.

In this exercise, you write application code to locate hotspots on your application's streaming source. To set up the application, you do the following steps:

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

   ```
   {"x": 7.921782426109737, "y": 8.746265312709893, "is_hot": "N"}
   {"x": 0.722248626528026, "y": 4.648868803193405, "is_hot": "Y"}
   ```

   The example provides a Python script for you to populate the stream. The `x` and `y` values are randomly generated, with some records being clustered around certain locations.

   The `is_hot` field is provided as an indicator if the script intentionally generated the value as part of a hotspot. This can help you evaluate whether the hotspot detection function is working properly.

2. **Create the application** – Using the AWS Management Console, you then create a Kinesis Data Analytics application. Configure the application input by mapping the streaming source to an in-application stream (`SOURCE_SQL_STREAM_001`). When the application starts, Kinesis Data Analytics continuously reads the streaming source and inserts records into the in-application stream.

   In this exercise, you use the following code for the application:

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "x" DOUBLE,
    "y" DOUBLE,
    "is_hot" VARCHAR(4),
    HOTSPOTS_RESULT VARCHAR(10000)
);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT "x", "y", "is_hot", "HOTSPOTS_RESULT"
    FROM TABLE (
        HOTSPOTS(
            CURSOR(SELECT STREAM "x", "y", "is_hot" FROM "SOURCE_SQL_STREAM_001"),
            1000,
            0.2,
            17)
    );
```

The code reads rows in the SOURCE_SQL_STREAM_001, analyzes it for significant hotspots, and writes the resulting data to another in-application stream (DESTINATION_SQL_STREAM). You use pumps to insert rows in in-application streams. For more information, see In-Application Streams and Pumps.

3. **Configure the output** – You configure the application output to send data from the application to an external destination, which is another Kinesis data stream. Review the hotspot scores and determine what scores indicate that a hotspot occurred (and that you need to be alerted). You can use an AWS Lambda function to further process hotspot information and configure alerts.
4. **Verify the output** – The example includes a JavaScript application that reads data from the output stream and displays it graphically, so you can view the hotspots that the application generates in real time.

The exercise uses the US West (Oregon) (us-west-2) to create these streams and your application. If you use any other Region, update the code accordingly.

Topics

- Step 1: Create the Input and Output Streams
- Step 2: Create the Kinesis Data Analytics Application

Step 1: Create the Input and Output Streams

Before you create an Amazon Kinesis Data Analytics application for the Hotspots example, you create two Kinesis data streams. Configure one of the streams as the streaming source for your application, and the other stream as the destination where Kinesis Data Analytics persists your application output.

Topics

Step 1.1: Create the Kinesis Data Streams

In this section, you create two Kinesis data streams: `ExampleInputStream` and `ExampleOutputStream`.

Create these data streams using the console or the AWS CLI.

- To create the data streams using the console:

  1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
  2. Choose **Data Streams** in the navigation pane.
  3. Choose **Create Kinesis stream**, and create a stream with one shard named `ExampleInputStream`.
  4. Repeat the previous step, creating a stream with one shard named `ExampleOutputStream`.

- To create data streams using the AWS CLI:

  - Create streams (`ExampleInputStream` and `ExampleOutputStream`) using the following Kinesis `create-stream` AWS CLI command. To create the second stream, which the application will use to write output, run the same command, changing the stream name to `ExampleOutputStream`.

    ```
    $ aws kinesis create-stream \
    ```

```
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser

$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

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.

```
{"x": 7.921782426109737, "y": 8.746265312709893, "is_hot": "N"}
{"x": 0.722248626580026, "y": 4.648868803193405, "is_hot": "Y"}
```

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. This code does the following:

   - Generates a potential hotspot somewhere in the (X, Y) plane.
   - Generates a set of 1,000 points for each hotspot. Of these points, 20 percent are clustered around the hotspot. The rest are generated randomly within the entire space.
   - The `put-record` command writes the JSON records to the stream.

> **Important**
>
> Do not upload this file to a web server because it contains your AWS credentials.

```
import json
from pprint import pprint
import random
import time
import boto3

STREAM_NAME = "ExampleInputStream"


def get_hotspot(field, spot_size):
    hotspot = {
        "left": field["left"] + random.random() * (field["width"] - spot_size),
        "width": spot_size,
        "top": field["top"] + random.random() * (field["height"] - spot_size),
        "height": spot_size,
    }
    return hotspot


def get_record(field, hotspot, hotspot_weight):
    rectangle = hotspot if random.random() < hotspot_weight else field
    point = {
        "x": rectangle["left"] + random.random() * rectangle["width"],
        "y": rectangle["top"] + random.random() * rectangle["height"],
        "is_hot": "Y" if rectangle is hotspot else "N",
    }
    return {"Data": json.dumps(point), "PartitionKey": "partition_key"}


def generate(
    stream_name, field, hotspot_size, hotspot_weight, batch_size, kinesis_client
):
    """
    Generates points used as input to a hotspot detection algorithm.
    With probability hotspot_weight (20%), a point is drawn from the hotspot;
    otherwise, it is drawn from the base field. The location of the hotspot
    changes for every 1000 points generated.
    """
    points_generated = 0
    hotspot = None
    while True:
```

```
        if points_generated % 1000 == 0:
            hotspot = get_hotspot(field, hotspot_size)
        records = [
            get_record(field, hotspot, hotspot_weight) for _ in range(batch_size)
        ]
        points_generated += len(records)
        pprint(records)
        kinesis_client.put_records(StreamName=stream_name, Records=records)

        time.sleep(0.1)


if __name__ == "__main__":
    generate(
        stream_name=STREAM_NAME,
        field={"left": 0, "width": 10, "top": 0, "height": 10},
        hotspot_size=1,
        hotspot_weight=0.2,
        batch_size=10,
        kinesis_client=boto3.client("kinesis"),
    )
```

Next Step

Step 2: Create the Kinesis Data Analytics Application

Step 2: Create the Kinesis Data Analytics Application

In this section of the Hotspots example, you create an Kinesis Data Analytics application as follows:

- Configure the application input to use the Kinesis data stream you created as the streaming source in Step 1.
- Use the provided application code in the AWS Management Console.

To create an application

1. Create a Kinesis Data Analytics application by following steps 1, 2, and 3 in the Getting Started exercise (see Step 3.1: Create an Application).

   In the source configuration, do the following:

- Specify the streaming source you created in the section called "Step 1: Create Streams".
- After the console infers the schema, edit the schema. Ensure that the x and y column types are set to DOUBLE and that the IS_HOT column type is set to VARCHAR.

2. Use the following application code (you can paste this code into the SQL editor):

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "x" DOUBLE,
    "y" DOUBLE,
    "is_hot" VARCHAR(4),
    HOTSPOTS_RESULT VARCHAR(10000)
);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT "x", "y", "is_hot", "HOTSPOTS_RESULT"
    FROM TABLE (
        HOTSPOTS(
            CURSOR(SELECT STREAM "x", "y", "is_hot" FROM "SOURCE_SQL_STREAM_001"),
            1000,
            0.2,
            17)
    );
```

3. Run the SQL code and review the results.

Filter by column name

ROWTIME	x	y	is_hot	HOTSPOTS_RESULT
2018-03-19 20:19:20.298	3.2902233757560313	1.1460673734716675	N	{"hotspots":[{"density":159.34972933221212,"minValues":[0.4791038226753084,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040
2018-03-19 20:19:21.313	9.758694911135013	9.66632832516424	N	{"hotspots":[{"density":180.8921951354484,"minValues":[0.6623354726891375,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040
2018-03-19 20:19:21.313	8.986657300548824	3.558000293320571	N	{"hotspots":[{"density":180.8921951354484,"minValues":[0.6623354726891375,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040
2018-03-19 20:19:21.313	5.193048038272014	4.94448855569874	Y	{"hotspots":[{"density":180.8921951354484,"minValues":[0.6623354726891375,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040

Next Step

Step 3: Configure the Application Output

Step 3: Configure the Application Output

At this point in the Hotspots example, you have Amazon Kinesis Data Analytics application code discovering significant hotspots from a streaming source and assigning a heat score to each.

You can now send the application result from the in-application stream to an external destination, which is another Kinesis data stream (`ExampleOutputStream`). You can then analyze the hotspot scores and determine what an appropriate threshold is for hotspot heat. You can extend this application further to generate alerts.

To configure the application output

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. In the SQL editor, choose either **Destination** or **Add a destination** in the application dashboard.
3. On the **Add a destination** page, choose **Select from your streams**. Then choose the `ExampleOutputStream` stream that 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`.
4. You can optionally configure AWS Lambda to monitor the `ExampleOutputStream` stream and send you alerts. For more information, see Using a Lambda Function as Output. You can also review the records that Kinesis Data Analytics writes to the external destination, which is the Kinesis stream `ExampleOutputStream`, as described in Step 4: Verify the Application Output.

Next Step

Step 4: Verify the Application Output

Step 4: Verify the Application Output

In this section of the Hotspots example, you set up a web application that displays the hotspot information in a Scalable Vector Graphics (SVG) control.

1. Create a file named `index.html` with the following contents:

```
<!doctype html>
<html lang=en>
<head>
    <meta charset=utf-8>
    <title>hotspots viewer</title>

    <style>
```

```
    #visualization {
      display: block;
      margin: auto;
    }

    .point {
      opacity: 0.2;
    }

    .hot {
      fill: red;
    }

    .cold {
      fill: blue;
    }

    .hotspot {
      stroke: black;
      stroke-opacity: 0.8;
      stroke-width: 1;
      fill: none;
    }
    </style>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.202.0.min.js"></script>
    <script src="https://d3js.org/d3.v4.min.js"></script>
</head>
<body>
<svg id="visualization" width="600" height="600"></svg>
<script src="hotspots_viewer.js"></script>
</body>
</html>
```

2. Create a file in the same directory named `hotspots_viewer.js` with the following contents. Provide your , credentials, and output stream name in the variables provided.

```
// Visualize example output from the Kinesis Analytics hotspot detection algorithm.
// This script assumes that the output stream has a single shard.

// Modify this section to reflect your AWS configuration
var awsRegion = "",        // The  where your Kinesis Analytics application is
 configured.
```

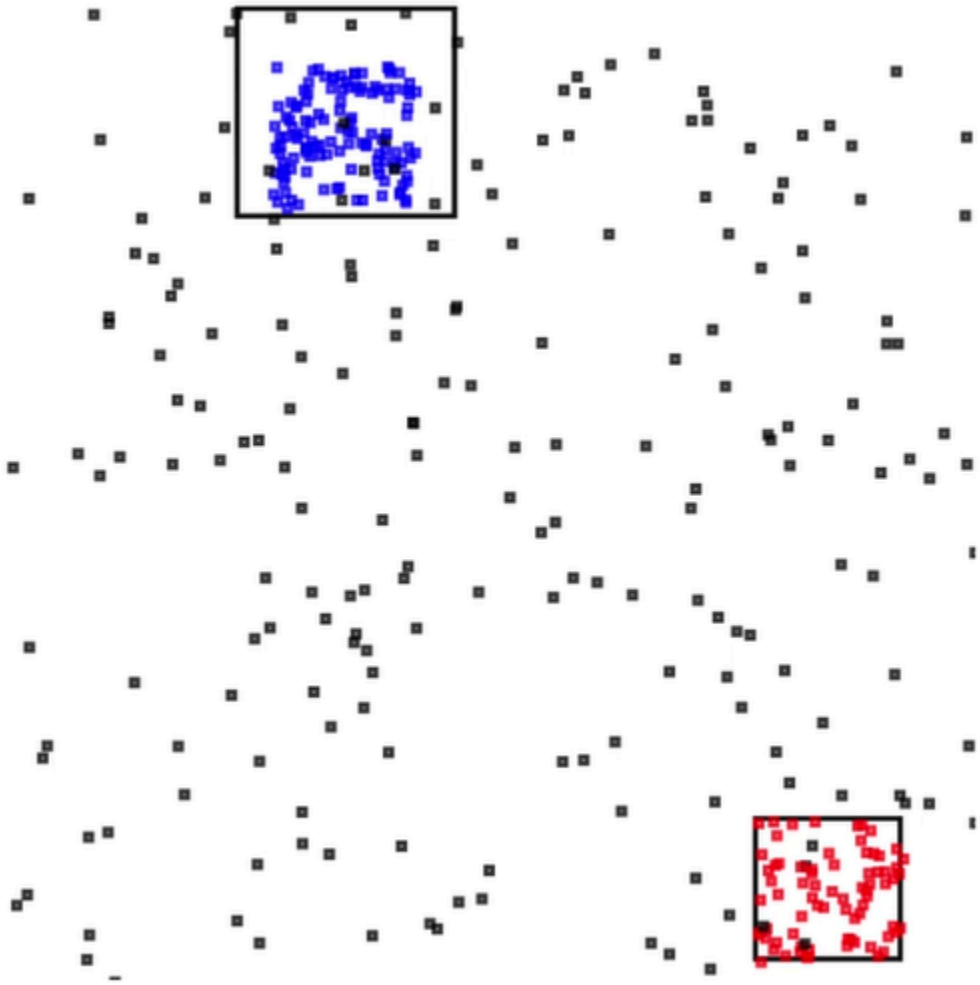
```
    accessKeyId = "",      // Your Access Key ID
    secretAccessKey = "",  // Your Secret Access Key
    outputStream = "";     // The name of the Kinesis Stream where the output from
 the HOTSPOTS function is being written

// The variables in this section should reflect way input data was generated and
 the parameters that the HOTSPOTS
// function was called with.
var windowSize = 1000, // The window size used for hotspot detection
    minimumDensity = 40,  // A filter applied to returned hotspots before
 visualization
    xRange = [0, 10],  // The range of values to display on the x-axis
    yRange = [0, 10];  // The range of values to display on the y-axis

////////////////////////////////////////////////////////////////////////////////
// D3 setup
////////////////////////////////////////////////////////////////////////////////

var svg = d3.select("svg"),
    margin = {"top": 20, "right": 20, "bottom": 20, "left": 20},
    graphWidth = +svg.attr("width") - margin.left - margin.right,
    graphHeight = +svg.attr("height") - margin.top - margin.bottom;

// Return the linear function that maps the segment [a, b] to the segment [c, d].
function linearScale(a, b, c, d) {
    var m = (d - c) / (b - a);
    return function(x) {
        return c + m * (x - a);
    };
}

// helper functions to extract the x-value from a stream record and scale it for
 output
var xValue = function(r) { return r.x; },
    xScale = linearScale(xRange[0], xRange[1], 0, graphWidth),
    xMap = function(r) { return xScale(xValue(r)); };

// helper functions to extract the y-value from a stream record and scale it for
 output
var yValue = function(r) { return r.y; },
    yScale = linearScale(yRange[0], yRange[1], 0, graphHeight),
    yMap = function(r) { return yScale(yValue(r)); };
```

```
// a helper function that assigns a CSS class to a point based on whether it was
 generated as part of a hotspot
var classMap = function(r) { return r.is_hot == "Y" ? "point hot" : "point
 cold"; };

var g = svg.append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

function update(records, hotspots) {

    var points = g.selectAll("circle")
        .data(records, function(r) { return r.dataIndex; });

    points.enter().append("circle")
        .attr("class", classMap)
        .attr("r", 3)
        .attr("cx", xMap)
        .attr("cy", yMap);

    points.exit().remove();

    if (hotspots) {
        var boxes = g.selectAll("rect").data(hotspots);

        boxes.enter().append("rect")
            .merge(boxes)
            .attr("class", "hotspot")
            .attr("x", function(h) { return xScale(h.minValues[0]); })
            .attr("y", function(h) { return yScale(h.minValues[1]); })
            .attr("width", function(h) { return xScale(h.maxValues[0]) -
 xScale(h.minValues[0]); })
            .attr("height", function(h) { return yScale(h.maxValues[1]) -
 yScale(h.minValues[1]); });

        boxes.exit().remove();
    }
}

////////////////////////////////////////////////////////////////////////////////
// Use the AWS SDK to pull output records from Kinesis and update the visualization
////////////////////////////////////////////////////////////////////////////////

var kinesis = new AWS.Kinesis({
    "region": awsRegion,
```

```
    "accessKeyId": accessKeyId,
    "secretAccessKey": secretAccessKey
});

var textDecoder = new TextDecoder("utf-8");

// Decode an output record into an object and assign it an index value
function decodeRecord(record, recordIndex) {
    var record = JSON.parse(textDecoder.decode(record.Data));
    var hotspots_result = JSON.parse(record.HOTSPOTS_RESULT);
    record.hotspots = hotspots_result.hotspots
        .filter(function(hotspot) { return hotspot.density >= minimumDensity});
    record.index = recordIndex
    return record;
}

// Fetch a new records from the shard iterator, append them to records, and update
 the visualization
function getRecordsAndUpdateVisualization(shardIterator, records, lastRecordIndex)
 {
    kinesis.getRecords({
        "ShardIterator": shardIterator
    }, function(err, data) {
        if (err) {
            console.log(err, err.stack);
            return;
        }

        var newRecords = data.Records.map(function(raw) { return decodeRecord(raw,
 ++lastRecordIndex); });
        newRecords.forEach(function(record) { records.push(record); });

        var hotspots = null;
        if (newRecords.length > 0) {
            hotspots = newRecords[newRecords.length - 1].hotspots;
        }

        while (records.length > windowSize) {
            records.shift();
        }

        update(records, hotspots);
```

```
        getRecordsAndUpdateVisualization(data.NextShardIterator, records,
 lastRecordIndex);
    });
}

// Get a shard iterator for the output stream and begin updating the visualization.
 Note that this script will only
// read records from the first shard in the stream.
function init() {
    kinesis.describeStream({
        "StreamName": outputStream
    }, function(err, data) {
        if (err) {
            console.log(err, err.stack);
            return;
        }

        var shardId = data.StreamDescription.Shards[0].ShardId;

        kinesis.getShardIterator({
            "StreamName": outputStream,
            "ShardId": shardId,
            "ShardIteratorType": "LATEST"
        }, function(err, data) {
            if (err) {
                console.log(err, err.stack);
                return;
            }
            getRecordsAndUpdateVisualization(data.ShardIterator, [], 0);
        })
    });
}

// Start the visualization
init();
```

3. With the Python code from the first section running, open `index.html` in a web browser. The hotspot information appears on the page, as shown following.



Examples: Alerts and Errors

This section provides examples of Kinesis Data Analytics applications that use alerts and errors. Each example provides step-by-step instructions and code to help you set up and test your Kinesis Data Analytics application.

Topics

Example: Creating Simple Alerts

In this Kinesis Data Analytics application, the query runs continuously on the in-application stream that is created over the demo stream. For more information, see Continuous Queries.

If any rows show a stock price change that is greater than 1 percent, those rows are inserted into another in-application stream. In the exercise, you can configure the application output to persist the results to an external destination. You can then further investigate the 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 analytics application as described in the Kinesis Data Analytics Getting Started exercise.
2. In the SQL editor in Kinesis Data Analytics, replace the application code with the following:

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

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

   The SELECT statement in the application code filters rows in the SOURCE_SQL_STREAM_001 for stock price changes greater than 1 percent. It then inserts those rows into another in-application stream DESTINATION_SQL_STREAM using a pump. For more information about the coding pattern that explains using pumps to insert rows into in-application streams, see Application Code.
3. Choose **Save and run SQL**.
4. Add a destination. To do this, either choose the **Destination** tab in the SQL editor or choose **Add a destination** on the application details page.

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

On the **Connect to destination** page, choose **Create New**.

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

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

d. Return to the Kinesis Data Analytics console. On the **Connect to destination** page, choose the stream that you created.

If the stream does not appear, refresh the page.

e. Choose **Save and continue**.

Now you have an external destination, a Kinesis data stream, where 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 Preprocessing Data Using a Lambda Function.

Example: Creating Throttled Alerts

In this Kinesis Data Analytics application, the query runs continuously on the in-application stream created over the demo stream. For more information, see Continuous Queries. If any rows show that the stock price change is greater than 1 percent, those rows are inserted into another in-application stream. The application throttles the alerts such that an alert is sent immediately when the stock price changes. However, 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 a Kinesis Data Analytics application as described in the Kinesis Data Analytics Getting Started exercise.
2. In the SQL editor in Kinesis Data Analytics, replace the application code with the following:

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

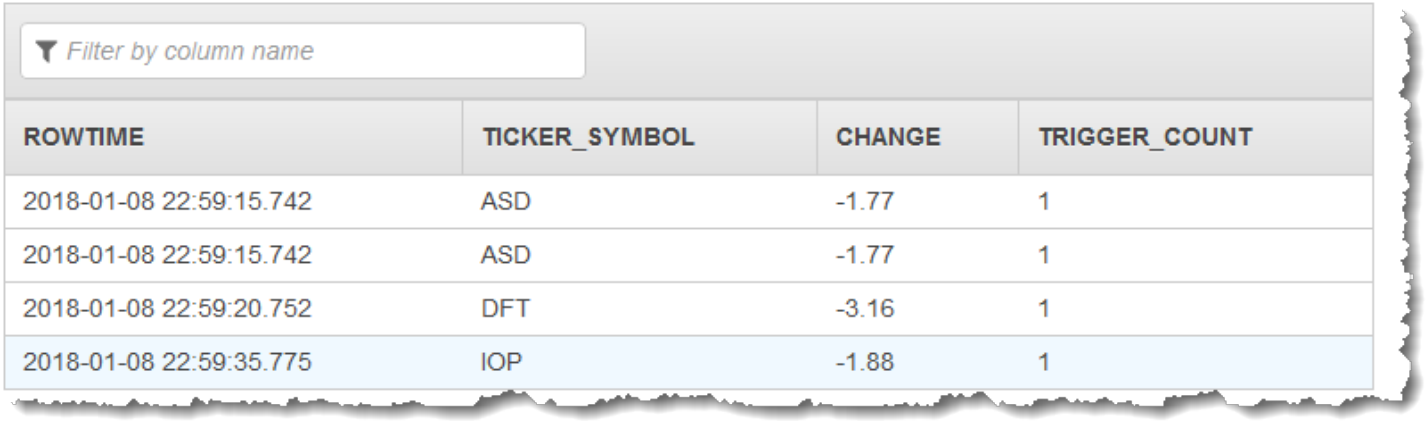
```
            price         DOUBLE);

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 to perform aggregations over last minute to keep track of triggers
    WINDOW W1 AS (PARTITION BY ticker_symbol RANGE INTERVAL '1' MINUTE PRECEDING)
)
WHERE trigger_count >= 1;
```

The SELECT statement in the application code filters rows in the SOURCE_SQL_STREAM_001 for stock price changes greater than 1 percent and inserts those rows into another in-application stream CHANGE_STREAM using a pump.

The application then creates a second stream named 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. Choose **Save and run SQL**.

The example outputs a stream to TRIGGER_COUNT_STREAM similar to the following:


Filter by column name

ROWTIME	TICKER_SYMBOL	CHANGE	TRIGGER_COUNT
2018-01-08 22:59:15.742	ASD	-1.77	1
2018-01-08 22:59:15.742	ASD	-1.77	1
2018-01-08 22:59:20.752	DFT	-3.16	1
2018-01-08 22:59:35.775	IOP	-1.88	1

Example: Exploring the In-Application Error Stream

Amazon Kinesis Data Analytics provides an in-application error stream for each application that 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.

You perform the following exercises on the console. In these examples, you introduce errors in the input configuration by editing the schema that is inferred by the discovery process, and then you verify the rows that are sent to the error stream.

Topics

- Introducing a Parse Error
- Introducing a Divide by Zero Error

Introducing a Parse Error

In this exercise, you introduce a parse error.

1. Create a Kinesis Data Analytics application as described in the Kinesis Data Analytics Getting Started exercise.
2. On the application details page, choose **Connect streaming data**.
3. If you followed the Getting Started exercise, you have a demo stream (`kinesis-analytics-demo-stream`) in your account. On the **Connect to source** page, choose this demo stream.
4. 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. Next, 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 the column type of the in-application schema that is created is invalid, Kinesis Data Analytics can't bring in data in the in-application stream. Instead, it 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, Kinesis Data Analytics could not bring the data in the in-application input stream. It sent the data to the error stream instead.

Introducing a 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 `DESTINATION_SQL_STREAM` in-application stream where the results are supposed to be written.

1. Create a Kinesis Data Analytics application as described in the Kinesis Data Analytics Getting Started exercise.

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

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

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

Because the division by zero runtime error occurs, instead of writing the results to the `DESTINATION_SQL_STREAM`, 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.

Examples: Solution Accelerators

The AWS Solutions Site has AWS CloudFormation templates available that you can use to create complete streaming data solutions quickly.

The following templates are available:

Real-time insights on AWS account activity

This solution records and visualizes resource access and usage metrics for your AWS account(s) in real-time. For more information, see Real-time insights on AWS account activity.

Real-time AWS IoT device monitoring with Kinesis Data Analytics

This solution collects, processes, analyzes and visualizes IoT device connectivity and activity data in real-time. For more information, see Real-time AWS IoT device monitoring with Kinesis Data Analytics.

Real-time web analytics with Kinesis Data Analytics

This solution collects, processes, analyzes and visualizes website clickstream data in real-time. For more information, see Real-time web analytics with Kinesis Data Analytics.

Amazon Connected Vehicle Solution

This solution collects, processes, analyzes and visualizes IoT data from vehicles in real-time. For more information, see Amazon Connected Vehicle Solution.

Security in Amazon Kinesis Data Analytics

Cloud security at AWS is the highest priority. As an AWS customer, you will benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. The effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS compliance programs. To learn about the compliance programs that apply to Kinesis Data Analytics, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Kinesis Data Analytics. The following topics show you how to configure Kinesis Data Analytics to meet your security and compliance objectives. You'll also learn how to use other Amazon services that can help you to monitor and secure your Kinesis Data Analytics resources.

Topics

Data Protection in Amazon Kinesis Data Analytics for SQL Applications

You can protect your data using tools that are provided by AWS. Kinesis Data Analytics can work with services that support encrypting data, including Kinesis Data Streams, Firehose, and Amazon S3.

Data Encryption in Kinesis Data Analytics

Encryption at Rest

Note the following about encrypting data at rest with Kinesis Data Analytics:

- You can encrypt data on the incoming Kinesis data stream using StartStreamEncryption. For more information, see What Is Server-Side Encryption for Kinesis Data Streams?.
- Output data can be encrypted at rest using Firehose to store data in an encrypted Amazon S3 bucket. You can specify the encryption key that your Amazon S3 bucket uses. For more information, see Protecting Data Using Server-Side Encryption with KMS–Managed Keys (SSE-KMS).
- Your application's code is encrypted at rest.
- Your application's reference data is encrypted at rest.

Encryption In Transit

Kinesis Data Analytics encrypts all data in transit. Encryption in transit is enabled for all Kinesis Data Analytics applications and cannot be disabled.

Kinesis Data Analytics encrypts data in transit in the following scenarios:

- Data in transit from Kinesis Data Streams to Kinesis Data Analytics.
- Data in transit between internal components within Kinesis Data Analytics.
- Data in transit between Kinesis Data Analytics and Firehose.

Key Management

Data encryption in Kinesis Data Analytics uses service-managed keys. Customer-managed keys are not supported.

Identity and Access Management in Kinesis Data Analytics

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 to access a streaming or reference source, 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 source (for example, an Kinesis stream, a Firehose delivery stream, or a reference source in an Amazon S3 bucket), 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",
                "kinesis:ListShards"
            ],
            "Resource": [
                "arn:aws:kinesis:us-east-1:123456789012:stream/inputStreamName"
            ]
        }
    ]
```

```
}
```

Permissions Policy for Reading a Firehose Delivery Stream

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputFirehose",
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:Get*"
            ],
            "Resource": [
                "arn:aws:firehose:us-
east-1:123456789012:deliverystream/inputFirehoseName"
            ]
        }
    ]
}
```

> **Note**
>
> The `firehose:Get*` permission refers to an internal accessor that Kinesis Data Analytics uses to access the stream. There is no public accessor for a Firehose delivery stream.

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 a Kinesis Stream

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "WriteOutputKinesis",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:us-east-1:123456789012:stream/output-stream-
name"
            ]
        }
    ]
}
```

Permissions Policy for Writing to a Firehose Delivery Stream

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "WriteOutputFirehose",
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Resource": [
                "arn:aws:firehose:us-east-1:123456789012:deliverystream/output-
firehose-name"
            ]
        }
    ]
}
```

Permissions Policy for Reading a Reference Data Source from an Amazon S3 Bucket

JSON

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

Cross-service confused deputy prevention

In AWS, cross-service impersonation can occur when one service (the calling service) calls another service (the called service). The calling service can be manipulated to act on another customer's resources even though it shouldn't have the proper permissions, resulting in the confused deputy problem.

To prevent confused deputies, AWS provides tools that help you protect your data for all services using service principals that have been given access to resources in your account. This section focuses on cross-service confused deputy prevention specific to Kinesis Data Analytics however, you can learn more about this topic at The confused deputy problem section of the *IAM User Guide*.

In the context of Kinesis Data Analytics for SQL, we recommend using the aws:SourceArn and aws:SourceAccount global condition context keys in your role trust policy to limit access to the role to only those requests that are generated by expected resources.

Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must be the ARN of the resource used by Kinesis Data Analytics, which is specified with the following format: `arn:aws:kinesisanalytics:region:account:resource`.

The recommended approach to the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full resource ARN.

If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` key with wildcard characters (*) for the unknown portions of the ARN. For example: `arn:aws:kinesisanalytics::111122223333:*`.

While most actions in the Kinesis Data Analytics for SQL API such as CreateApplication, AddApplicationInput and DeleteApplication are made in context of specific applications, the DiscoverInputSchema action is not executed in the context of any application. That means the role used in this action must not fully specify a resource in the `SourceArn` condition key. Following is an example that uses a wildcard ARN:

```
{
   ...
   "ArnLike":{
      "aws:SourceArn":"arn:aws:kinesisanalytics:us-east-1:123456789012:*"
   }
   ...
}
```

The default role generated by Kinesis Data Analytics for SQL uses this wildcard. This ensures discovering input schema works seamlessly in the console experience. However, we recommend editing the Trust Policy to use a full ARN after discovering the schema to implement complete confused deputy mitigation.

Policies of roles that you provide to Kinesis Data Analytics as well as trust policies of roles generated for you can make use of aws:SourceArn and aws:SourceAccount condition keys.

In order to protect against the confused deputy problem, carry out the following steps:

To protect against the confused deputy problem

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles** and then choose the role you want to modify.

3. Choose **Edit trust policy**.
4. On the **Edit trust policy** page, replace the default JSON policy with a policy that uses one or both of the `aws:SourceArn` and `aws:SourceAccount` global condition context keys. See the following example policy:
5. Choose **Update policy**.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"kinesisanalytics.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"Account ID"
            },
            "ArnEquals":{
               "aws:SourceArn":"arn:aws:kinesisanalytics:us-
east-1:123456789012:application/my-app"
            }
         }
      }
   ]
}
```

Authentication and Access Control for

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

Access Control

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

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

- Overview of Managing Access Permissions to Your Resources
- Using Identity-Based Policies (IAM Policies) for
- API Permissions: Actions, Permissions, and Resources Reference

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Overview of Managing Access Permissions to Your Resources

> **Warning**
>
> For new projects, we recommend that you use the new Managed Service for Apache Flink Studio over for SQL Applications. Managed Service for Apache Flink Studio combines ease of use with advanced analytical capabilities, enabling you to build sophisticated stream processing applications in minutes.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

> **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*.

Topics

Resources and Operations

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

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

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

provides a set of operations to work with resources. For a list of available operations, see Actions.

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, a 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 , the resource is an application.)
- If you create a 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. We strongly recommend you grant permissions to roles and not users.
- 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 . 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 IAM JSON Policy Reference in the *IAM User Guide*.

Policies that are attached to an IAM identity are referred to as *identity-based* policies (IAM policies). Policies that are attached to a resource are referred to as *resource-based* policies. supports only identity-based policies (IAM policies).

Topics

- Identity-Based Policies (IAM Policies)
- Resource-Based 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 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 Amazon 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 Amazon service principal if you want to grant an Amazon 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 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.

JSON

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

For more information about using identity-based policies with , see Using Identity-Based Policies (IAM Policies) for. 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. doesn't support resource-based policies.

Specifying Policy Elements: Actions, Effects, and Principals

For each resource, the service defines a set of API operations. To grant permissions for these API operations, 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 Resources and Operations and Actions.

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 Resources and Operations.

- **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). doesn't support resource-based policies.

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

For a list showing all of the API operations and the resources that they apply to, see API Permissions: Actions, Permissions, and Resources Reference.

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 . 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

The following are 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 grant permissions to perform operations on resources.

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

Topics

- Permissions Required to Use the Console
- Amazon-Managed (Predefined) Policies for
- Customer Managed Policy Examples

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 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 API operations and the resources that they apply to, see API Permissions: Actions, Permissions, and Resources Reference.

Permissions Required to Use the Console

For a user to work with the console, you must grant the necessary permissions. For example, if you want a user to have permissions to create an application, grant permissions that show them the streaming sources in the account so that the user can configure input and output on the console.

We recommend the following:

- Use the Amazon-managed policies to grant user permissions. For available policies, see Amazon-Managed (Predefined) Policies for.
- 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.

Amazon-Managed (Predefined) Policies for

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These Amazon-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 Amazon-Managed Policies in the *IAM User Guide*.

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

- **`AmazonKinesisAnalyticsReadOnly`** – Grants permissions for actions that enable a user to list applications and review input/output configuration. It also grants permissions that allow a user to view a list of Kinesis streams and 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 actions and all other permissions that allows a user to create and manage 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` Amazon-managed policy.

- A user must have permission for the `iam:PassRole` action to specify an IAM role when configuring application. This Amazon-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 application with a role that does not have this prefix, you first must explicitly grant the user permission for the `iam:PassRole` action on the specific role.

You can also create your own custom IAM policies to allow permissions for actions and resources. You can attach these custom policies to the 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 a user in your account. Then 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 can't do anything on the console. As you attach policies to the user, you can verify that the user can perform various actions on the console.

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

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

Example steps

- Step 1: Create an IAM User
- Step 2: Allow the User Permissions for Actions that Are Not Specific to

- Step 3: Allow the User to View a List of Applications and View Details
- Step 4: Allow the User to Start a Specific Application
- Step 5: Allow the User to Create an Application
- Step 6: Allow the Application to Use Lambda Preprocessing

Step 1: Create an IAM User

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

First, grant a user permission for all actions that aren't specific to that the user will need when working with applications. These include permissions for working with streams (Amazon Kinesis Data Streams actions, Amazon 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:DescribeStream",
                "kinesis:ListStreams",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:ListDeliveryStreams"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:GetLogEvents",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListPolicyVersions",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/service-role/role-name"
        }
    ]
}
```

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

The following policy grants a user the following permissions:

- Permission for the `kinesisanalytics:ListApplications` action so the user can view a list of applications. 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"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:DescribeApplication"
            ],
            "Resource": "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/*"
        }
    ]
}
```

Verify these permissions by signing into the console using the 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 applications, attach the following policy to the user. The policy provides the permission for the `kinesisanalytics:StartApplication` action. You must update the policy by providing your account ID, AWS Region and application name.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:StartApplication"
            ],
            "Resource": "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/application-name"
        }
    ]
}
```

Step 5: Allow the User to Create an Application

If you want the user to create an application, you can then attach the following policy to the user. You must 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 that the user can specify any application name (and thus create multiple applications).

JSON

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

```
            "Action": [
                "kinesisanalytics:StartApplication",
                "kinesisanalytics:UpdateApplication",
                "kinesisanalytics:AddApplicationInput",
                "kinesisanalytics:AddApplicationOutput"
            ],
            "Resource": "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/application-name"
        }
    ]
}
```

Step 6: Allow the Application to Use Lambda Preprocessing

If you want the application to be able to use Lambda preprocessing, attach the following policy to the role.

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

API Permissions: Actions, Permissions, and Resources Reference

When you are setting up Access Control 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 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 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`).

API and Required Permissions for Actions

API Operation:

Required Permissions (API Action):

Resources:

API and Required Permissions for Actions

Amazon RDS API and Required Permissions for Actions

API Operation: AddApplicationInput

Action: `kinesisanalytics:AddApplicationInput`

Resources:

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

GetApplicationState

The console uses an internal method called `GetApplicationState` to sample or access application data. Your service application needs to have permissions for the internal `kinesisanalytics:GetApplicationState` API to sample or access application data through the AWS Management Console.

Monitoring Amazon Kinesis Data Analytics

Kinesis Data Analytics provides monitoring functionality for your applications. For more information, see *Monitoring*.

Compliance Validation for Amazon Kinesis Data Analytics for SQL Applications

Third-party auditors assess the security and compliance of Amazon Kinesis Data Analytics as part of multiple AWS compliance programs. These include SOC, PCI, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see Amazon Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Kinesis Data Analytics is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. If your use of Kinesis Data Analytics is subject to compliance with standards such as HIPAA or PCI, AWS provides resources to help:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub CSPM – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon Kinesis Data Analytics

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability

Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Kinesis Data Analytics offers several features to help support your data resiliency and backup needs.

Disaster Recovery

Kinesis Data Analytics runs in a serverless mode, and takes care of host degradations, Availability Zone availability, and other infrastructure related issues by performing automatic migration. When this happens, Kinesis Data Analytics ensures that the application is processed without any loss of data. For more information, see Delivery Model for Persisting Application Output to an External Destination.

Infrastructure Security in Kinesis Data Analytics for SQL Applications

As a managed service, Amazon Kinesis Data Analytics is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Kinesis Data Analytics through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Security Best Practices for Kinesis Data Analytics

Amazon Kinesis Data Analytics provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be

appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Use IAM roles to access other Amazon services

Your Kinesis Data Analytics application must have valid credentials to access resources in other services, such as Kinesis data streams, Firehose delivery streams, or Amazon S3 buckets. You should not store AWS credentials directly in the application or in an Amazon S3 bucket. These are long-term credentials that are not automatically rotated and could have a significant business impact if they are compromised.

Instead, you should use an IAM role to manage temporary credentials for your application to access other resources. When you use a role, you don't have to use long-term credentials to access other resources.

For more information, see the following topics in the *IAM User Guide*:

- IAM Roles
- Common Scenarios for Roles: Users, Applications, and Services

Implement Server-Side Encryption in Dependent Resources

Data at rest and data in transit is encrypted in Kinesis Data Analytics, and this encryption cannot be disabled. You should implement server-side encryption in your dependent resources, such as Kinesis data streams, Firehose delivery streams, and Amazon S3 buckets. For more information on implementing server-side encryption in dependent resources, see Data Protection.

Use CloudTrail to Monitor API Calls

Kinesis Data Analytics is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an Amazon service in Kinesis Data Analytics.

Using the information collected by CloudTrail, you can determine the request that was made to Kinesis Data Analytics, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information, see the section called “Using AWS CloudTrail”.

Monitoring for SQL Applications

Monitoring is an important part of maintaining the reliability, availability, and performance of and your 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 , 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 performance in your environment, by measuring performance at various times and under different load conditions. As you monitor , 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 , 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 processes data streams, see Amazon Kinesis Data Analytics for SQL Applications: How It Works.

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.

Topics

- Monitoring with Amazon CloudWatch
- Logging API Calls with AWS CloudTrail

Monitoring Tools

AWS provides various tools that you can use to monitor . 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 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.
- **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 involves manually monitoring those items that the CloudWatch alarms don't cover. The , CloudWatch, Trusted Advisor, and other AWS Management 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 metrics
  - Create and edit alarms to be notified of problems
- AWS Trusted Advisor can help you monitor your 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 applications using Amazon CloudWatch. CloudWatch collects and processes raw data from into readable, near real-time metrics. These statistics are retained for a period of two weeks. You can access the historical information and gain a better perspective on how your web application or service is performing. By default, metric data is automatically sent to CloudWatch. For more information, see What Is Amazon CloudWatch? in the *Amazon CloudWatch User Guide*.

Topics

- Metrics and Dimensions
- Viewing Metrics and Dimensions
- Creating CloudWatch Alarms to Monitor
- Working with Amazon CloudWatch Logs

Metrics and Dimensions

The AWS/KinesisAnalytics namespace includes the following metrics.

Metric	Description
Bytes	The number of bytes read (per input stream) or written (per output stream). Levels: Per input stream and per output stream
KPUs	The number of Kinesis Processing Units that are used to run your stream processing application. The average number of KPUs used each hour determines the billing for your application. Levels: Application-level
MillisBehindLatest	Indicates how far behind from the current time an application is reading from the streaming source. Levels: Application-level
Records	The number of records read (per input stream) or written (per output stream). Levels: Per input stream and per output stream
Success	1 for each successful delivery attempt to the destination configured for your application; 0 for every failed delivery attempt. The average value of this metric indicates how many successful deliveries are performed. Levels: Per destination.
InputProcessing.Duration	The time taken for each AWS Lambda function invocation performed by . Levels: Per input stream

Metric	Description
InputProcessing.OkRecords	The number of records returned by a Lambda function that were marked with Ok status. Levels: Per input stream
InputProcessing.OkBytes	The sum of bytes of the records returned by a Lambda function that were marked with Ok status. Levels: Per input stream
InputProcessing.DroppedRecords	The number of records returned by a Lambda function that were marked with Dropped status. Levels: Per input stream
InputProcessing.ProcessingFailedRecords	The number of records returned by a Lambda function that were marked with ProcessingFailed status. Levels: Per input stream
InputProcessing.Success	The number of successful Lambda invocations by . Levels: Per input stream
LambdaDelivery.OkRecords	The number of records returned by a Lambda function that were marked with Ok status. Levels: Per Lambda destination
LambdaDelivery.DeliveryFailedRecords	The number of records returned by a Lambda function that were marked with DeliveryFailed status. Levels: Per Lambda destination

Metric	Description
`LambdaDelivery.Duration`	The time taken for each Lambda function invocation performed by . Levels: Per Lambda destination

provides metrics for the following dimensions.

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

Viewing Metrics and Dimensions

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

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

To view metrics using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. In the **CloudWatch Metrics by Category** pane for , choose 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
```

metrics are collected at the following levels:

- Application
- Input stream
- Output stream

Creating CloudWatch Alarms to Monitor

You can create an Amazon 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 AWS 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** starts.
3. Choose **Kinesis Analytics Metrics**. Then scroll through the metrics to locate the metric you want to place an alarm on.

   To display just metrics, search for the file system ID of your file system. Choose the metric to create an alarm for, and then choose **Next**.
4. Enter 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 application is misconfigured, it can transition to a running state during application start. Or it can 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.

can generate configuration errors under the following conditions:

- The Kinesis data stream used for input doesn't exist.
- The Amazon 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.
- doesn't have permission to assume the IAM role that manages related permissions.

For more information about Amazon CloudWatch, see the *Amazon CloudWatch User Guide*.

Adding the PutLogEvents Policy Action

needs permissions to write misconfiguration errors to CloudWatch. You can add these permissions to the IAM role that assumes, as described following. For more information on using an IAM role for , see Identity and Access Management in Kinesis Data Analytics.

Trust Policy

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

JSON

```
{
  "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 resource, you can use the following IAM permissions policy.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt0123456789000",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:my-log-group:log-
stream:my-log-stream*"
            ]
        }
    ]
}
```

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.

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

```
      "LogStreamARN": "<Amazon Resource Name (ARN) of the CloudWatch log stream to add
 to the new application>",
      "RoleARN": "<ARN of the role to use to access the log>"
 }]
}
```

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 about `AddApplicationCloudWatchLoggingOption`, see AddApplicationCloudWatchLoggingOption.

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

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 about `UpdateApplication`, see UpdateApplication.

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

```
}
```

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 about `DeleteApplicationCloudWatchLoggingOption`, see DeleteApplicationCloudWatchLoggingOption.

```
{
   "ApplicationName": "<Name of application to delete log option from>",
   "CloudWatchLoggingOptionId": "<ID of the application log option to delete>",
   "CurrentApplicationVersionId": <Version of the application to delete the log option
 from>
}
```

Configuration Errors

The following sections contain details about errors that you might see in Amazon CloudWatch Logs from a misconfigured application.

Error Message Format

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

```
{
"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.
- `messageType`: The message type. Currently, this type can be only `ERROR`.
- `message`: The details of the error, for example:

```
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`: The 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.
- `referenceId`: The 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.
- `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 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.

```
{
"applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
"applicationVersionId": "5",
 "messageType": "ERROR",
```

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

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

```
{
 "applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
 "applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your reference data.
 Please check that the bucket and the file exist, the role has the correct permissions
 to access these resources and that Kinesis Analytics can assume the role provided.",
 "referenceId":"1.1",
 "errorCode": "ReferenceDataError",
 "messageSchemaVersion": "1"
}
```

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.

```
{
 "applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
 "applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your input. Please
 check that the resource exists, the role has the correct permissions to access the
 resource and that Kinesis Analytics can assume the role provided.",
 "inputId":null,
 "errorCode": "InputError",
 "messageSchemaVersion": "1"
}
```

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 a Kinesis source stream, an error like the following is generated.

```
{
 "applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
 "applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your input. Please
 check that the resource exists, the role has the correct permissions to access the
 resource and that Kinesis Analytics can assume the role provided.",
 "inputId":null,
 "errorCode": "InputError",
 "messageSchemaVersion": "1"
}
```

Logging API Calls with AWS CloudTrail

is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in . CloudTrail captures all API calls for as events. The calls captured include calls from the console and code calls to the API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for . If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to , the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in , that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for , create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail

in the console, the trail applies to all . The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All actions are logged by CloudTrail and are documented in the API reference. For example, calls to the `CreateApplication` and `UpdateApplication` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with AWS account root user or user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the AddApplicationCloudWatchLoggingOption and DescribeApplication actions.

```
{
    "Records": [
        {
```

```
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2019-03-14T01:03:00Z",
            "eventSource": "kinesisanalytics.amazonaws.com",
            "eventName": "AddApplicationCloudWatchLoggingOption",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "currentApplicationVersionId": 1,
                "cloudWatchLoggingOption": {
                    "roleARN": "arn:aws:iam::012345678910:role/cloudtrail_test",
                    "logStreamARN": "arn:aws:logs:us-east-1:012345678910:log-
group:cloudtrail-test:log-stream:sql-cloudwatch"
                },
                "applicationName": "cloudtrail-test"
            },
            "responseElements": null,
            "requestID": "e897cd34-45f4-11e9-8912-e52573a36cd9",
            "eventID": "57fe50e9-c764-47c3-a0aa-d0c271fa1cbb",
            "eventType": "AwsApiCall",
            "apiVersion": "2015-08-14",
            "recipientAccountId": "303967445486"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2019-03-14T05:37:20Z",
            "eventSource": "kinesisanalytics.amazonaws.com",
            "eventName": "DescribeApplication",
```

```
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "applicationName": "cloudtrail-test"
            },
            "responseElements": null,
            "requestID": "3b74eb29-461b-11e9-a645-fb677e53d147",
            "eventID": "750d0def-17b6-4c20-ba45-06d9d45e87ee",
            "eventType": "AwsApiCall",
            "apiVersion": "2015-08-14",
            "recipientAccountId": "012345678910"
        }
    ]
}
```

Limits

Discontinuation dates

After careful consideration, we have made the decision to discontinue Amazon Kinesis Data Analytics for SQL applications. To help you plan and migrate away from Amazon Kinesis Data Analytics for SQL applications, we will discontinue the offering gradually over 15 months. There are two important dates to note, **October 15, 2025**, and **January 27, 2026**.

1. On **October 15, 2025**, we will stop your applications and place them into a READY state. You will be able to *re-start* your applications at that time and continue to use your applications as normal, subject to service limits.
2. From **October 15, 2025**, you will not be able to create *new* Amazon Kinesis Data Analytics for SQL applications. You will be able to run any *existing* applications as normal, subject to service limits.
3. We will delete your applications starting **January 27, 2026**. You will not be able to start or operate your Amazon Kinesis Data Analytics for SQL applications. Support will no longer be available for Amazon Kinesis Data Analytics for SQL applications from that time.

We recommend that you migrate your applications to Amazon Managed Service for Apache Flink or Amazon Managed Service for Apache Flink Studio before October 15, 2025. For resources to assist with your migration, see Migrating to Managed Service for Apache Flink Studio Examples. To learn more about Amazon Managed Service for Apache Flink or Amazon Managed Service for Apache Flink Studio, see the Amazon Managed Service for Apache Flink developer guide.

Limits

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

- Kinesis Data Analytics for SQL is available in the following AWS Regions: US East (Ohio), US East (N. Virginia), US West (Oregon), Canada (Central), Europe (Paris), Europe (Ireland), Europe (Frankfurt), Europe (London), Asia Pacific (Hong Kong), Asia Pacific (Mumbai), Asia Pacific (Sydney), Asia Pacific (Singapore), Asia Pacific (Seoul), Asia Pacific (Tokyo), South America (Sao Paulo), AWS GovCloud (US-East), AWS GovCloud (US-West). We have no plans to launch Kinesis Data Analytics for SQL into additional AWS Regions.

- After June 28, 2023, you will be not be able to create new Kinesis Data Analytics for SQL applications using the AWS management console if you do not already use Kinesis Data Analytics for SQL. For information about Kinesis Data Analytics for SQL discontinuation dates, see Discontinuation dates. If you created a Kinesis Data Analytics for SQL application before June 28 2023, there are no changes to how you create and run applications today in an AWS Region where you already use Kinesis Data Analytics for SQL. However, you will no longer be able to create new applications using the AWS Console in a Region where you do not use Kinesis Data Analytics for SQL.
- After September 12, 2023, you will not be able to create new applications using Kinesis Data Firehose as a source if you do not already use Kinesis Data Analytics for SQL. For information about Kinesis Data Analytics for SQL discontinuation dates, see Discontinuation dates. Existing customers using Kinesis Data Analytics for SQL applications with `KinesisFirehoseInput` can continue to add applications with `KinesisFirehoseInput` within an existing account using Kinesis Data Analytics. If you are an existing customer and wish to create a new account with Kinesis Data Analytics for SQL applications with `KinesisFirehoseInput` you can open a support case. For more information, see the AWS Support Center.
- The size of a row in an in-application stream is limited to 512 KB. Kinesis Data Analytics uses up to 1 KB to store metadata. This metadata counts against the row limit. If record size on the streaming source is greater than 50 KB, you can split the record into multiple rows in your application stream by providing appropriate schema in the input configuration by providing row delimiter.
- The SQL code in an application is limited to 100 KB.
- The longest window we recommend for a windowed query is one hour. In-application streams are stored in volatile storage, and unexpected application interruptions will cause the application to rebuild the stream from the source data in the volatile storage.
- The most throughput we recommend for a single in-application stream is between 2 and 20 MB/second, depending on the complexity of the application's query.
- 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 streaming throughput a single Kinesis Data Analytics for SQL application can process is approximately 100 MB/sec. This assumes that you have increased the number of in-application streams to the maximum value of 64, and you have increased your KPU limit beyond 8 (see the following limit for details). If your application needs to process more than 100 MB/sec of input, do one of the following:

- Use multiple Kinesis Data Analytics for SQL applications to process input
- Use Managed Service for Apache Flink for Java Applications if you want to continue to use a single stream and application.

> **Note**
>
> We advise periodically reviewing your application's `InputProcessing.0kBytes` metric so that you can plan ahead to use multiple SQL applications or migrate to Amazon Managed Service for Apache Flink for Java Applications if your application's projected input throughput exceeds 100 MB/sec. We also advise creating a CloudWatch alarm on `InputProcessing.0kBytes` so that you are notified when your application is nearing input throughput limit. This can be useful as you can update your application query to tradeoff for higher throughput, thereby avoiding backpressure and delay in analytics. For more information, see Troubleshooting. Alarming can also be useful if you have a mechanism to reduce throughput in upstream.

- 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 Amazon 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 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).
- You can assign up to 50 tags per application.

Best Practices

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

Topics

Managing Applications

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

- Set up Amazon CloudWatch alarms – You can use the CloudWatch metrics that Kinesis Data Analytics provides to monitor the following:
  - Input bytes and input records (number of bytes and records entering the application)
  - Output bytes and output records
  - `MillisBehindLatest` (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:
  - `MillisBehindLatest` – For most cases, we recommend that you set this alarm to trigger when your application is 1 hour behind the latest data, for an average of 1 minute. For applications with lower end-to-end processing needs, you can tune this to a lower tolerance. This alarm can help ensure that your application is reading the latest data.

- To avoid getting the `ReadProvisionedThroughputException` exception, limit the number of production applications reading from the same Kinesis data stream to two applications.

  > **Note**
  >
  > In this case, *application* refers to any application that can read from the streaming source. Only a Kinesis Data Analytics application can read from a Firehose delivery stream. However, many applications can read from a Kinesis data stream, such as a 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, limit the number of applications reading the same data source.

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

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

Scaling Applications

Set up your application for your future scaling needs by proactively increasing the number of input in-application streams from the default (one). We recommend the following language choices based on the throughput of your application:

- Use multiple streams and Kinesis Data Analytics for SQL applications if your application has scaling needs beyond 100 MB/second.

- Use Managed Service for Apache Flink Applications if you want to use a single stream and application.

> **Note**
>
> We advise periodically reviewing your application's `InputProcessing.0kBytes` metric so that you can plan ahead to use multiple SQL applications or migrate to managed-flink/latest/java/ if your application's projected input throughput exceeds 100 MB/sec.

Monitoring Applications

We advise creating a CloudWatch alarm on `InputProcessing.0kBytes` so that you are notified when your application is nearing input throughput limit. This can be useful as you can update your application query to tradeoff for higher throughput, thereby avoiding backpressure and delay in analytics. For more information, see Troubleshooting. This can also be useful if you have a mechanism to reduce throughput in upstream.

- The most throughput we recommend for a single in-application stream is between 2 and 20 MB/second, depending on the complexity of the application's query.
- The maximum streaming throughput a single Kinesis Data Analytics for SQL application can process is approximately 100 MB/sec. This assumes that you have increased the number of in-application streams to the maximum value of 64, and you have increased your KPU limit beyond 8. For more information, see Limits.

> **Note**
>
> We advise periodically reviewing your application's `InputProcessing.0kBytes` metric so that you can plan ahead to use multiple SQL applications or migrate to managed-flink/latest/java/ if your application's projected input throughput exceeds 100 MB/sec.

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) 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 that 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, the discovery API might have missed sampling one or more record types. This situation 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 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, 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 is enforced.

  Any attempt to insert rows with `NULL` values when a value is required results in an error. 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 an inferred schema has a `SMALLINT` as a column type, consider changing it to an `INTEGER`.

- Use SQL functions in your application code to handle any unstructured data or columns. You might have unstructured data or columns, such as log data, in your input. For examples, see Example: Transforming DateTime Values. 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 and use the `String` and `Date Time` functions to parse and schematize the raw data.

- Make sure that you completely handle streaming source data that contains nesting more than two levels deep. When source data is JSON, you can have nesting. The discovery API infers a schema that flattens one level of nesting. For two levels of nesting, the discovery API also tries to flatten these. Beyond two levels of nesting, there is limited support for flattening. To handle nesting completely, you have to manually modify the inferred schema to suit 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 that you want to bring in your application. You can do this 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 Firehose delivery stream.

Authoring Application Code

We recommend the following:

- In your SQL statement, don't specify a time-based window that is longer than one hour for the following reasons:
  - Sometimes an application needs to be restarted, either because you updated the application or for Kinesis Data Analytics internal reasons. When it restarts, all data included in the window must be read again from the streaming data source. This takes time before Kinesis Data Analytics can emit output for that window.

  - Kinesis Data Analytics must maintain everything related to the application's state, including relevant data, for the duration. This consumes significant Kinesis Data Analytics processing units.
- During development, keep the 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, consider breaking it into multiple statements, in each step saving results in intermediate in-application streams. This might help you debug faster.
- When you're 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.

Testing Applications

When you're changing the schema or application code for your Kinesis Data Analytics application, we recommend using a test application to verify your changes before deploying them to production.

Setting up a Test Application

You can set up a test application either through the console, or by using an CloudFormation template. Using an CloudFormation template helps ensure that the code changes you make to the test application and your live application are consistent.

When setting up a test application, you can either connect the application to your live data, or you can populate a stream with mock data to test against. We recommend two methods for populating a stream with mock data:

- Use the Kinesis Data Generator (KDG). The KDG uses a data template to send random data to a Kinesis stream. The KDG is simple to use, but isn't appropriate for testing complex relationships between data items, such as for applications that detect data hotspots or anomalies.
- Use a custom Python application to send more complex data to a Kinesis data stream. A Python application can generate complex relationships between data items, such as hotspots or anomalies. For an example of a Python application that sends data clustered into a data hotspot, see Example: Detecting Hotspots on a Stream (HOTSPOTS Function).

When running your test application, view your results using a destination (such as a Firehose delivery stream to an Amazon Redshift database) instead of viewing your in-application stream on the console. The data that is displayed on the console is a sampling of the stream and doesn't contain all of the records.

Testing Schema Changes

When changing an application's input stream schema, use your test application to verify that the following are true:

- The data from your stream is being coerced into the correct data type. For example, ensure that datetime data is not being ingested into the application as a string.
- The data is being parsed and coerced into the data type that you want. If parsing or coercion errors occur, you can view them on the console, or assign a destination to the error stream and view the errors in the destination store.
- The data fields for character data are of sufficient length, and the application isn't truncating the character data. You can check the data records in your destination store to verify that your application data isn't being truncated.

Testing Code Changes

Testing changes to your SQL code requires some domain knowledge of your application. You must be able to determine what output needs to be tested and what the correct output should be. For potential problem areas to verify when modifying your application's SQL code, see Troubleshooting Amazon Kinesis Data Analytics for SQL Applications.

Troubleshooting Amazon Kinesis Data Analytics for SQL Applications

The following can help you troubleshoot problems that you might encounter with Amazon Kinesis Data Analytics for SQL Applications.

Topics

Stopped applications

- **What is a stopped Kinesis Data Analytics for SQL application?**

  A stopped application is an application that we have observed not processing any records for a minimum of three months. This means customers are paying for Kinesis Data Analytics for SQL resources they are not using.

- **When will AWS begin stopping idle applications?**

  AWS will begin stopping idle applications on November 14, 2023 and complete by November 21, 2023. We will stop idle applications in the office hours timezone of that Region.

- **Can stopped Kinesis Data Analytics for SQL applications be re-started?**

  Yes. If you require to re-start your application you can do so as normal. There is no need to cut a support ticket.

- **When AWS stops an idle application will any of my query results also be deleted?**

No. First, because your application is idle it is not processing queries. Second, your query results are not stored in Kinesis Data Analytics for SQL. You configure your Kinesis Data Analytics for SQL application with a sink destination where the results of its calculations are sent (for example, in Amazon S3 or another data stream). As such, you retain full ownership of your data and it will remain retrievable under the terms of that storage service.

- **What do I do if I don't want my application stopped?**

  You can email the service team (kda-sql-questions@amazon.com) requesting applications not be stopped any time before November 10, 2023. The email should include your account ID and application ARN.

Unable to Run SQL Code

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

- For more information about SQL statements, see Kinesis Data Analytics for SQL examples. 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're 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 can't 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, or data that uses a format other than comma-separated values (CSV), or the discovery API did not discover your schema. In these cases, you can define a schema manually 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.

Reference Data is Out of Date

Reference data is loaded from the Amazon Simple Storage Service (Amazon S3) object into the application when the application is started or updated, or during application interruptions that are caused by service issues.

Reference data is not loaded into the application when updates are made to the underlying Amazon S3 object.

If the reference data in the application is not up to date, you can reload the data by following these steps:

1. On the Kinesis Data Analytics console, choose the application name in the list, and then choose **Application details**.
2. Choose **Go to SQL editor** to open the **Real-time analytics** page for the application.
3. In the **Source Data** view, choose your reference data table name.
4. Choose **Actions**, **Synchronize reference data table**.

Application Not Writing to Destination

If data is not being written to the destination, check the following:

- Verify that the application's role has sufficient permission to access the destination. For more information, see Permissions Policy for Writing to a Kinesis Stream or Permissions Policy for Writing to a Firehose Delivery Stream.
- Verify that the application destination is correctly configured and that the application is using the correct name for the output stream.
- Check the Amazon CloudWatch metrics for your output stream to see if data is being written. For information about using CloudWatch metrics, see Monitoring with Amazon CloudWatch.
- Add a CloudWatch log stream using the section called "AddApplicationCloudWatchLoggingOption". Your application will write configuration errors to the log stream.

If the role and destination configuration look correct, try restarting the application, specifying `LAST_STOPPED_POINT` for the InputStartingPositionConfiguration.

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 general rule, 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.

For more information, see Best Practices.

Invalid Code Errors When Running an Application

When you can't 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 can't redefine the same stream in your code. For more information about creating a stream, see CREATE STREAM in the *Amazon Kinesis Data Analytics SQL Reference*. 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 in the *Amazon Kinesis Data Analytics SQL Reference*.
- **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 can't have the same name as the service-reserved stream `error_stream`.

Application is Writing Errors to the Error Stream

If your application is writing errors to the in-application error stream, you can decode the value in the `DATA_ROW` field using standard libraries. For more information about the error stream, see Error Handling.

Insufficient Throughput or High MillisBehindLatest

If your application's MillisBehindLatest metric is steadily increasing or consistently is above 1000 (one second), it can be due to the following reasons:

- Check your application's InputBytes CloudWatch metric. If you are ingesting more than 4 MB/sec, this can cause an increase in `MillisBehindLatest`. To improve your application's throughput, increase the value of the `InputParallelism` parameter. For more information, see Parallelizing Input Streams for Increased Throughput.
- Check your application's output delivery Success metric for failures in delivering to your destination. Verify that you have correctly configured the output, and that your output stream has sufficient capacity.
- If your application uses an AWS Lambda function for pre-processing or as an output, check the application's InputProcessing.Duration or LambdaDelivery.Duration CloudWatch metric. If the Lambda function invocation duration is longer than 5 seconds, consider doing the following:
  - Increase the Lambda function's **Memory** allocation. You can do this on the AWS Lambda console, on the **Configuration** page, under **Basic settings**. For more information, see Configuring Lambda Functions in the *AWS Lambda Developer Guide*.
  - Increase the number of shards in your input stream of the application. This increases the number of parallel functions that the application will invoke, which might increase throughput.
  - Verify that the function is not making blocking calls that are affecting performance, such as synchronous requests for external resources.
  - Examine your AWS Lambda function to see whether there are other areas where you can improve performance. Check the CloudWatch Logs of the application Lambda function. For more information, see Accessing Amazon CloudWatch Metrics for in the *AWS Lambda Developer Guide*.

- Verify that your application is not reaching the default limit for Kinesis Processing Units (KPU). If your application is reaching this limit, you can request a limit increase. For more information, see Automatically Scaling Applications to Increase Throughput.
- If your application is still having issues after having your KPU limit increase, check that your application's input throughput does not exceed 100MB/sec. If it exceeds 100MB/sec, we recommend implementing changes to reduce overall throughput to stabilize the application, for example by reducing the amount of data being sent to the data source that the Kinesis Data Analytics Sql application reads from. We also recommended other approaches, including increasing the paralellism of the application, reducing the time period of computations, changing columnar data types from VARCHAR to data types with smaller sizes (e.g., INTEGER, LONG, etc), and reducing data processed by sampling or filtering.

> **Note**
>
> We advise periodically reviewing your application's `InputProcessing.OkBytes` metric so that you can plan ahead to use multiple SQL applications or migrate to managed-flink/latest/java/ if your application's projected input throughput will exceed 100 MB/sec.

Kinesis Data Analytics SQL Reference

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

API Reference

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Managed Service for Apache Flink API V2 Documentation.

You can use the AWS CLI to explore the Amazon Kinesis Data Analytics API. This guide provides Getting Started with Amazon Kinesis Data Analytics for SQL Applications exercises that use the AWS CLI.

Topics

Actions

The following actions are supported:

- DescribeApplication
- DiscoverInputSchema
- ListApplications
- ListTagsForResource
- StartApplication
- StopApplication
- TagResource
- UntagResource
- UpdateApplication

AddApplicationCloudWatchLoggingOption

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

Adds 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

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

Request Parameters

The request accepts the following data in JSON format.

ApplicationName

The Kinesis Analytics application name.

Type: String

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

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

Required: Yes

CloudWatchLoggingOption

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 object

Required: Yes

CurrentApplicationVersionId

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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AddApplicationInput

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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.

Any configuration update, including adding a streaming source using this operation, results in a new version of the application. You can use the DescribeApplication 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

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

Type: String

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

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

Required: Yes

CurrentApplicationVersionId

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

Type: Long

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

Required: Yes

Input

The Input to add.

Type: Input 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.

message

Test

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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++

- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AddApplicationInputProcessingConfiguration

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

Adds an InputProcessingConfiguration 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

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

Type: String

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

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

Required: Yes

CurrentApplicationVersionId

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

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 operation.

Type: String

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

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

Required: Yes

InputProcessingConfiguration

The InputProcessingConfiguration to add to the application.

Type: InputProcessingConfiguration 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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++

- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AddApplicationOutput

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

Adds an external destination to your Amazon Kinesis Analytics application.

If you want Amazon Kinesis Analytics to deliver data from an in-application stream within your application to an external destination (such as an Amazon Kinesis stream, 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 operation to find the current application version.

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

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

Request Syntax

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "Output": {
      "DestinationSchema": {
         "RecordFormatType": "string"
      },
```

```
      "KinesisFirehoseOutput": {
         "ResourceARN": "string",
         "RoleARN": "string"
      },
      "KinesisStreamsOutput": {
         "ResourceARN": "string",
         "RoleARN": "string"
      },
      "LambdaOutput": {
         "ResourceARN": "string",
         "RoleARN": "string"
      },
      "Name": "string"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName

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

Type: String

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

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

Required: Yes

CurrentApplicationVersionId

Version of the application to which you want to add the output configuration. You can use the DescribeApplication 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

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 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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AddApplicationReferenceDataSource

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

Adds a reference data source to an existing application.

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

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

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

Request Syntax

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "ReferenceDataSource": {
      "ReferenceSchema": {
         "RecordColumns": [
            {
               "Mapping": "string",
               "Name": "string",
               "SqlType": "string"
            }
         ],
         "RecordEncoding": "string",
         "RecordFormat": {
```

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

Request Parameters

The request accepts the following data in JSON format.

ApplicationName

Name of an existing application.

Type: String

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

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

Required: Yes

CurrentApplicationVersionId

Version of the application for which you are adding the reference data source. You can use the DescribeApplication 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

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 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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateApplication

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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

```
{
   "ApplicationCode": "string",
   "ApplicationDescription": "string",
   "ApplicationName": "string",
   "CloudWatchLoggingOptions": [
```

```
      {
         "LogStreamARN": "string",
         "RoleARN": "string"
      }
   ],
   "Inputs": [
      {
         "InputParallelism": {
            "Count": number
         },
         "InputProcessingConfiguration": {
            "InputLambdaProcessor": {
               "ResourceARN": "string",
               "RoleARN": "string"
            }
         },
         "InputSchema": {
            "RecordColumns": [
               {
                  "Mapping": "string",
                  "Name": "string",
                  "SqlType": "string"
               }
            ],
            "RecordEncoding": "string",
            "RecordFormat": {
               "MappingParameters": {
                  "CSVMappingParameters": {
                     "RecordColumnDelimiter": "string",
                     "RecordRowDelimiter": "string"
                  },
                  "JSONMappingParameters": {
                     "RecordRowPath": "string"
                  }
               },
               "RecordFormatType": "string"
            }
         },
         "KinesisFirehoseInput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "KinesisStreamsInput": {
            "ResourceARN": "string",
```

```
            "RoleARN": "string"
         },
         "NamePrefix": "string"
      }
   ],
   "Outputs": [
      {
         "DestinationSchema": {
            "RecordFormatType": "string"
         },
         "KinesisFirehoseOutput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "KinesisStreamsOutput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "LambdaOutput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "Name": "string"
      }
   ],
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationCode

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 102400.

Required: No

ApplicationDescription

Summary description of the application.

Type: String

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

Required: No

ApplicationName

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

Type: String

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

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

Required: Yes

CloudWatchLoggingOptions

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 objects

Required: No

Inputs

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 objects

Required: No

Outputs

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 objects

Required: No

Tags

A list of one or more tags to assign to the application. A tag is a key-value pair that identifies an application. Note that the maximum number of application tags includes system tags. The maximum number of user-defined application tags is 50. For more information, see Using Tagging.

Type: Array of Tag objects

Array Members: Minimum number of 1 item. Maximum number of 200 items.

Required: No

Response Syntax

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

Response Elements

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

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

ApplicationSummary

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 object

Errors

CodeValidationException

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

message

Test

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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

LimitExceededException

Exceeded the number of applications allowed.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

TooManyTagsException

Application created with too many tags, or too many tags added to an application. Note that the maximum number of application tags includes system tags. The maximum number of user-defined application tags is 50.

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplication

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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

Name of the Amazon Kinesis Analytics application to delete.

Type: String

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

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

Required: Yes

CreateTimestamp

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.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplicationCloudWatchLoggingOption

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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

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

Request Parameters

The request accepts the following data in JSON format.

ApplicationName

The Kinesis Analytics application name.

Type: String

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

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

Required: Yes

CloudWatchLoggingOptionId

The `CloudWatchLoggingOptionId` of the CloudWatch logging option to delete. You can get the `CloudWatchLoggingOptionId` by using the DescribeApplication operation.

Type: String

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

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

Required: Yes

CurrentApplicationVersionId

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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplicationInputProcessingConfiguration

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

Deletes an InputProcessingConfiguration from an input.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

ApplicationName

The Kinesis Analytics application name.

Type: String

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

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

Required: Yes

CurrentApplicationVersionId

The version ID of the Kinesis Analytics application.

Type: Long

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

Required: Yes

InputId

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 operation.

Type: String

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

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

Required: Yes

Response Elements

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

Errors

ConcurrentModificationException

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

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplicationOutput

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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

Amazon Kinesis Analytics application name.

Type: String

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

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

Required: Yes

CurrentApplicationVersionId

Amazon Kinesis Analytics application version. You can use the DescribeApplication 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

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 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 operation to get the specific `OutputId`.

Type: String

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

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

Required: Yes

Response Elements

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

Errors

ConcurrentModificationException

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

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3

- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplicationReferenceDataSource

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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 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

Name of an existing application.

Type: String

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

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

Required: Yes

CurrentApplicationVersionId

Version of the application. You can use the DescribeApplication 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

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

Type: String

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

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

Required: Yes

Response Elements

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

Errors

ConcurrentModificationException

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

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3

- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeApplication

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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 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

Name of the application.

Type: String

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

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

Required: Yes

Response Syntax

```
{
   "ApplicationDetail": {
      "ApplicationARN": "string",
      "ApplicationCode": "string",
      "ApplicationDescription": "string",
      "ApplicationName": "string",
      "ApplicationStatus": "string",
      "ApplicationVersionId": number,
      "CloudWatchLoggingOptionDescriptions": [
         {
            "CloudWatchLoggingOptionId": "string",
            "LogStreamARN": "string",
            "RoleARN": "string"
         }
      ],
      "CreateTimestamp": number,
      "InputDescriptions": [
         {
            "InAppStreamNames": [ "string" ],
            "InputId": "string",
            "InputParallelism": {
               "Count": number
            },
            "InputProcessingConfigurationDescription": {
               "InputLambdaProcessorDescription": {
                  "ResourceARN": "string",
                  "RoleARN": "string"
               }
            },
            "InputSchema": {
               "RecordColumns": [
                  {
                     "Mapping": "string",
                     "Name": "string",
                     "SqlType": "string"
                  }
               ],
               "RecordEncoding": "string",
               "RecordFormat": {
                  "MappingParameters": {
                     "CSVMappingParameters": {
```

```
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                     },
                     "JSONMappingParameters": {
                        "RecordRowPath": "string"
                     }
                  },
                  "RecordFormatType": "string"
               }
            },
            "InputStartingPositionConfiguration": {
               "InputStartingPosition": "string"
            },
            "KinesisFirehoseInputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "KinesisStreamsInputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "NamePrefix": "string"
         }
      ],
      "LastUpdateTimestamp": number,
      "OutputDescriptions": [
         {
            "DestinationSchema": {
               "RecordFormatType": "string"
            },
            "KinesisFirehoseOutputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "KinesisStreamsOutputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "LambdaOutputDescription": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "Name": "string",
            "OutputId": "string"
```

```
         }
      ],
      "ReferenceDataSourceDescriptions": [
         {
            "ReferenceId": "string",
            "ReferenceSchema": {
               "RecordColumns": [
                  {
                     "Mapping": "string",
                     "Name": "string",
                     "SqlType": "string"
                  }
               ],
               "RecordEncoding": "string",
               "RecordFormat": {
                  "MappingParameters": {
                     "CSVMappingParameters": {
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                     },
                     "JSONMappingParameters": {
                        "RecordRowPath": "string"
                     }
                  },
                  "RecordFormatType": "string"
               }
            },
            "S3ReferenceDataSourceDescription": {
               "BucketARN": "string",
               "FileKey": "string",
               "ReferenceRoleARN": "string"
            },
            "TableName": "string"
         }
      ]
   }
}
```

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

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 object

Errors

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python

- AWS SDK for Ruby V3

DiscoverInputSchema

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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

The InputProcessingConfiguration to use to preprocess the records before discovering the schema of the records.

Type: InputProcessingConfiguration object

Required: No

InputStartingPositionConfiguration

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

Type: InputStartingPositionConfiguration object

Required: No

ResourceARN

Amazon Resource Name (ARN) of the streaming source.

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 on your behalf.

Type: String

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

Pattern: `arn:.*`

Required: No

S3Configuration

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

Type: S3Configuration object

Required: No

Response Syntax

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

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 object

ParsedInputRecords

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

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

Type: Array of strings

RawInputRecords

Raw stream data that was sampled to infer the schema.

Type: Array of strings

Errors

InvalidArgumentException

Specified input parameter value is invalid.

message

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

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListApplications

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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.

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

Name of the application to start the list with. When using pagination to retrieve the list, you don't need to specify this parameter in the first request. However, in subsequent requests, you add the last application name from the previous response to get the next page of applications.

Type: String

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

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

Required: No

Limit

Maximum number of applications to list.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 50.

Required: No

Response Syntax

```
{
   "ApplicationSummaries": [
      {
         "ApplicationARN": "string",
         "ApplicationName": "string",
         "ApplicationStatus": "string"
      }
   ],
   "HasMoreApplications": boolean
}
```

Response Elements

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

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

ApplicationSummaries

List of `ApplicationSummary` objects.

Type: Array of ApplicationSummary objects

HasMoreApplications

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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListTagsForResource

Retrieves the list of key-value tags assigned to the application. For more information, see Using Tagging.

Request Syntax

```
{
   "ResourceARN": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ResourceARN

The ARN of the application for which to retrieve tags.

Type: String

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

Pattern: `arn:.*`

Required: Yes

Response Syntax

```
{
   "Tags": [
      {
         "Key": "string",
         "Value": "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.

Tags

The key-value tags assigned to the application.

Type: Array of Tag objects

Array Members: Minimum number of 1 item. Maximum number of 200 items.

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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartApplication

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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 operation.

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

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

ApplicationName

Name of the application.

Type: String

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

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

Required: Yes

InputConfigurations

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 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.

message

test

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3

- AWS SDK for Python
- AWS SDK for Ruby V3

StopApplication

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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 operation to find the application state. After the application is stopped, Amazon Kinesis Analytics stops reading data from the input, the application stops processing data, and there is no output written to the destination.

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

ApplicationName

Name of the running application to stop.

Type: String

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

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

Required: Yes

Response Elements

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

Errors

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3

- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

TagResource

Adds one or more key-value tags to a Kinesis Analytics application. Note that the maximum number of application tags includes system tags. The maximum number of user-defined application tags is 50. For more information, see Using Tagging.

Request Syntax

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ResourceARN

The ARN of the application to assign the tags.

Type: String

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

Pattern: `arn:.*`

Required: Yes

Tags

The key-value tags to assign to the application.

Type: Array of Tag objects

Array Members: Minimum number of 1 item. Maximum number of 200 items.

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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

TooManyTagsException

Application created with too many tags, or too many tags added to an application. Note that the maximum number of application tags includes system tags. The maximum number of user-defined application tags is 50.

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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UntagResource

Removes one or more tags from a Kinesis Analytics application. For more information, see Using Tagging.

Request Syntax

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ResourceARN

The ARN of the Kinesis Analytics application from which to remove the tags.

Type: String

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

Pattern: `arn:.*`

Required: Yes

TagKeys

A list of keys of tags to remove from the specified application.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 200 items.

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

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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

TooManyTagsException

Application created with too many tags, or too many tags added to an application. Note that the maximum number of application tags includes system tags. The maximum number of user-defined application tags is 50.

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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateApplication

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

Updates an existing Amazon Kinesis Analytics application. Using this API, you can update application code, input configuration, and output configuration.

Note that Amazon Kinesis Analytics updates the `CurrentApplicationVersionId` each time you update your application.

This operation requires permission for the `kinesisanalytics:UpdateApplication` action.

Request Syntax

```
{
   "ApplicationName": "string",
   "ApplicationUpdate": {
      "ApplicationCodeUpdate": "string",
      "CloudWatchLoggingOptionUpdates": [
         {
            "CloudWatchLoggingOptionId": "string",
            "LogStreamARNUpdate": "string",
            "RoleARNUpdate": "string"
         }
      ],
      "InputUpdates": [
         {
            "InputId": "string",
            "InputParallelismUpdate": {
               "CountUpdate": number
            },
            "InputProcessingConfigurationUpdate": {
               "InputLambdaProcessorUpdate": {
                  "ResourceARNUpdate": "string",
                  "RoleARNUpdate": "string"
               }
```

```
            },
            "InputSchemaUpdate": {
               "RecordColumnUpdates": [
                  {
                     "Mapping": "string",
                     "Name": "string",
                     "SqlType": "string"
                  }
               ],
               "RecordEncodingUpdate": "string",
               "RecordFormatUpdate": {
                  "MappingParameters": {
                     "CSVMappingParameters": {
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                     },
                     "JSONMappingParameters": {
                        "RecordRowPath": "string"
                     }
                  },
                  "RecordFormatType": "string"
               }
            },
            "KinesisFirehoseInputUpdate": {
               "ResourceARNUpdate": "string",
               "RoleARNUpdate": "string"
            },
            "KinesisStreamsInputUpdate": {
               "ResourceARNUpdate": "string",
               "RoleARNUpdate": "string"
            },
            "NamePrefixUpdate": "string"
         }
      ],
      "OutputUpdates": [
         {
            "DestinationSchemaUpdate": {
               "RecordFormatType": "string"
            },
            "KinesisFirehoseOutputUpdate": {
               "ResourceARNUpdate": "string",
               "RoleARNUpdate": "string"
            },
            "KinesisStreamsOutputUpdate": {
```

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

```
    },
    "CurrentApplicationVersionId": number
}
```

Request Parameters

The request accepts the following data in JSON format.

ApplicationName

Name of the Amazon Kinesis Analytics application to update.

Type: String

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

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

Required: Yes

ApplicationUpdate

Describes application updates.

Type: ApplicationUpdate object

Required: Yes

CurrentApplicationVersionId

The current application version ID. You can use the DescribeApplication 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.

message

Test

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.

message

HTTP Status Code: 400

InvalidArgumentException

Specified input parameter value is invalid.

message

HTTP Status Code: 400

ResourceInUseException

Application is not available for this operation.

message

HTTP Status Code: 400

ResourceNotFoundException

Specified application can't be found.

message

HTTP Status Code: 400

UnsupportedOperationException

The request was rejected because a specified parameter is not supported or a specified resource is not valid 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 V2
- AWS SDK for .NET V4
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Data Types

The following data types are supported:

- ApplicationDetail
- ApplicationSummary
- ApplicationUpdate
- CloudWatchLoggingOption
- CloudWatchLoggingOptionDescription
- CloudWatchLoggingOptionUpdate
- CSVMappingParameters

- DestinationSchema
- Input
- InputConfiguration
- InputDescription
- InputLambdaProcessor
- InputLambdaProcessorDescription
- InputLambdaProcessorUpdate
- InputParallelism
- InputParallelismUpdate
- InputProcessingConfiguration
- InputProcessingConfigurationDescription
- InputProcessingConfigurationUpdate
- InputSchemaUpdate
- InputStartingPositionConfiguration
- InputUpdate
- JSONMappingParameters
- KinesisFirehoseInput
- KinesisFirehoseInputDescription
- KinesisFirehoseInputUpdate
- KinesisFirehoseOutput
- KinesisFirehoseOutputDescription
- KinesisFirehoseOutputUpdate
- KinesisStreamsInput
- KinesisStreamsInputDescription
- KinesisStreamsInputUpdate
- KinesisStreamsOutput
- KinesisStreamsOutputDescription
- KinesisStreamsOutputUpdate
- LambdaOutput
- LambdaOutputDescription

- LambdaOutputUpdate
- MappingParameters
- Output
- OutputDescription
- OutputUpdate
- RecordColumn
- RecordFormat
- ReferenceDataSource
- ReferenceDataSourceDescription
- ReferenceDataSourceUpdate
- S3Configuration
- S3ReferenceDataSource
- S3ReferenceDataSourceDescription
- S3ReferenceDataSourceUpdate
- SourceSchema
- Tag

ApplicationDetail

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

Provides a description of the application, including the application Amazon Resource Name (ARN), status, latest version, and input and output configuration.

Contents

ApplicationARN

ARN of the application.

Type: String

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

Pattern: `arn:.*`

Required: Yes

ApplicationName

Name of the application.

Type: String

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

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

Required: Yes

ApplicationStatus

Status of the application.

Type: String

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

Required: Yes

ApplicationVersionId

Provides the current application version.

Type: Long

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

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 102400.

Required: No

ApplicationDescription

Description of the application.

Type: String

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

Required: No

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 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 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 objects

Required: No

ReferenceDataSourceDescriptions

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

Type: Array of ReferenceDataSourceDescription 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 Java V2
- AWS SDK for Ruby V3

ApplicationSummary

> **Note**
>
> This documentation is for version 1 of the Amazon Kinesis Data Analytics API, which only supports SQL applications. Version 2 of the API supports SQL and Java applications. For more information about version 2, see Amazon Kinesis Data Analytics API V2 Documentation.

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

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

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

Required: Yes

ApplicationStatus

Status of the application.

Type: String

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

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 Java V2
- AWS SDK for Ruby V3

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 102400.

Required: No

CloudWatchLoggingOptionUpdates

Describes application CloudWatch logging option updates.

Type: Array of CloudWatchLoggingOptionUpdate objects

Required: No

InputUpdates

Describes application input configuration updates.

Type: Array of InputUpdate objects

Required: No

OutputUpdates

Describes application output configuration updates.

Type: Array of OutputUpdate objects

Required: No

ReferenceDataSourceUpdates

Describes application reference data source updates.

Type: Array of ReferenceDataSourceUpdate 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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

CloudWatchLoggingOptionDescription

Description of the CloudWatch logging option.

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 used must have the `PutLogEvents` policy action enabled.

Type: String

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

Pattern: `arn:.*`

Required: Yes

CloudWatchLoggingOptionId

ID of the CloudWatch logging option description.

Type: String

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

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 Java V2
- AWS SDK for Ruby V3

CloudWatchLoggingOptionUpdate

Describes CloudWatch logging option updates.

Contents

CloudWatchLoggingOptionId

ID of the CloudWatch logging option to update

Type: String

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

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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 Java V2
- AWS SDK for Ruby V3

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: 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 Java V2
- AWS SDK for Ruby V3

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

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 object

Required: Yes

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

Length Constraints: Minimum length of 1. Maximum length of 32.

Required: Yes

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 object

Required: No

InputProcessingConfiguration

The InputProcessingConfiguration 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.

Type: InputProcessingConfiguration object

Required: No

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 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 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 Java V2
- AWS SDK for Ruby V3

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 operation.

Type: String

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

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

Required: Yes

InputStartingPositionConfiguration

Point at which you want the application to start processing records from the streaming source.

Type: InputStartingPositionConfiguration 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 Java V2
- AWS SDK for Ruby V3

InputDescription

Describes the application input configuration. For more information, see Configuring Application Input.

Contents

InAppStreamNames

Returns the in-application stream names that are mapped to the stream source.

Type: Array of strings

Length Constraints: Minimum length of 1. Maximum length of 32.

Required: No

InputId

Input ID associated with the application input. This is the ID that Amazon Kinesis Analytics assigns to each input configuration you add to your application.

Type: String

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

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

Required: No

InputParallelism

Describes the configured parallelism (number of in-application streams mapped to the streaming source).

Type: InputParallelism 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 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 object

Required: No

InputStartingPositionConfiguration

Point at which the application is configured to read from the input stream.

Type: InputStartingPositionConfiguration 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 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 object

Required: No

NamePrefix

In-application name prefix.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

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 Java V2
- AWS SDK for Ruby V3

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.

> **Note**
>
> To specify an earlier version of the Lambda function than the latest, include the Lambda function version in the Lambda function ARN. For more information about Lambda ARNs, see Example ARNs: AWS Lambda

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

InputLambdaProcessorUpdate

Represents an update to the InputLambdaProcessor 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.

> **Note**
>
> To specify an earlier version of the Lambda function than the latest, include the Lambda function version in the Lambda function ARN. For more information about Lambda ARNs, see Example ARNs: AWS Lambda

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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 Java V2
- AWS SDK for Ruby V3

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 Java V2
- AWS SDK for Ruby V3

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 that is used to preprocess the records in the stream before being processed by your application code.

Type: InputLambdaProcessor 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 Java V2
- AWS SDK for Ruby V3

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.

Type: InputLambdaProcessorDescription 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 Java V2
- AWS SDK for Ruby V3

InputProcessingConfigurationUpdate

Describes updates to an InputProcessingConfiguration.

Contents

InputLambdaProcessorUpdate

Provides update information for an InputLambdaProcessor.

Type: InputLambdaProcessorUpdate 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 Java V2
- AWS SDK for Ruby V3

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 Java V2
- AWS SDK for Ruby V3

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 Java V2
- AWS SDK for Ruby V3

InputUpdate

Describes updates to a specific input configuration (identified by the `InputId` of an application).

Contents

InputId

Input ID of the application input to be updated.

Type: String

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

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

Required: Yes

InputParallelismUpdate

Describes the parallelism updates (the number in-application streams Amazon Kinesis Analytics creates for the specific streaming source).

Type: InputParallelismUpdate object

Required: No

InputProcessingConfigurationUpdate

Describes updates for an input processing configuration.

Type: InputProcessingConfigurationUpdate 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 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 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 object

Required: No

NamePrefixUpdate

Name prefix for in-application streams that Amazon Kinesis Analytics creates for the specific streaming source.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

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 Java V2
- AWS SDK for Ruby V3

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

KinesisFirehoseOutputUpdate

When updating an output configuration using the UpdateApplication 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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

KinesisStreamsOutputDescription

For an application output, describes the Amazon Kinesis stream configured as its destination.

Contents

ResourceARN

Amazon Resource Name (ARN) of the Amazon Kinesis stream.

Type: String

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

Pattern: `arn:.*`

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:.*`

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 Java V2
- AWS SDK for Ruby V3

KinesisStreamsOutputUpdate

When updating an output configuration using the UpdateApplication 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:.*`

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 Java V2
- AWS SDK for Ruby V3

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.

> **Note**
>
> To specify an earlier version of the Lambda function than the latest, include the Lambda function version in the Lambda function ARN. For more information about Lambda ARNs, see Example ARNs: AWS Lambda

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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:.*`

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 Java V2
- AWS SDK for Ruby V3

LambdaOutputUpdate

When updating an output configuration using the UpdateApplication operation, provides information about an AWS Lambda function configured as the destination.

Contents

ResourceARNUpdate

Amazon Resource Name (ARN) of the destination Lambda function.

> **Note**
>
> To specify an earlier version of the Lambda function than the latest, include the Lambda function version in the Lambda function ARN. For more information about Lambda ARNs, see Example ARNs: AWS Lambda

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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 object

Required: No

JSONMappingParameters

Provides additional mapping information when JSON is the record format on the streaming source.

Type: JSONMappingParameters 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 Java V2
- AWS SDK for Ruby V3

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 object

Required: Yes

Name

Name of the in-application stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Required: Yes

KinesisFirehoseOutput

Identifies an Amazon Kinesis Firehose delivery stream as the destination.

Type: KinesisFirehoseOutput object

Required: No

KinesisStreamsOutput

Identifies an Amazon Kinesis stream as the destination.

Type: KinesisStreamsOutput object

Required: No

LambdaOutput

Identifies an AWS Lambda function as the destination.

Type: LambdaOutput 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 Java V2
- AWS SDK for Ruby V3

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 object

Required: No

KinesisFirehoseOutputDescription

Describes the Amazon Kinesis Firehose delivery stream configured as the destination where output is written.

Type: KinesisFirehoseOutputDescription object

Required: No

KinesisStreamsOutputDescription

Describes Amazon Kinesis stream configured as the destination where output is written.

Type: KinesisStreamsOutputDescription object

Required: No

LambdaOutputDescription

Describes the AWS Lambda function configured as the destination where output is written.

Type: LambdaOutputDescription object

Required: No

Name

Name of the in-application stream configured as output.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Required: No

OutputId

A unique identifier for the output configuration.

Type: String

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

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

Required: No

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 Java V2
- AWS SDK for Ruby V3

OutputUpdate

Describes updates to the output configuration identified by the `OutputId`.

Contents

OutputId

Identifies the specific output configuration that you want to update.

Type: String

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

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

Required: Yes

DestinationSchemaUpdate

Describes the data format when records are written to the destination. For more information, see Configuring Application Output.

Type: DestinationSchema object

Required: No

KinesisFirehoseOutputUpdate

Describes an Amazon Kinesis Firehose delivery stream as the destination for the output.

Type: KinesisFirehoseOutputUpdate object

Required: No

KinesisStreamsOutputUpdate

Describes an Amazon Kinesis stream as the destination for the output.

Type: KinesisStreamsOutputUpdate object

Required: No

LambdaOutputUpdate

Describes an AWS Lambda function as the destination for the output.

Type: LambdaOutputUpdate object

Required: No

NameUpdate

If you want to specify a different in-application stream for this output configuration, use this field to specify the new in-application stream name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

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 Java V2
- AWS SDK for Ruby V3

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

Name

Name of the column created in the in-application input stream or reference table.

Type: String

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

Mapping

Reference to the data element in the streaming input or the reference data source. This element is required if the RecordFormatType is `JSON`.

Type: String

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 Java V2

- AWS SDK for Ruby V3

RecordFormat

Describes the record format and relevant mapping information that should be applied to schematize the records on the stream.

Contents

RecordFormatType

The type of record format.

Type: String

Valid Values: `JSON | CSV`

Required: Yes

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 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 Java V2
- AWS SDK for Ruby V3

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 object

Required: Yes

TableName

Name of the in-application table to create.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

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` operation to trigger reloading of data into your application.

Type: S3ReferenceDataSource 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 Java V2
- AWS SDK for Ruby V3

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 operation.

Type: String

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

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

Required: Yes

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 object

Required: Yes

TableName

The in-application table name created by the specific reference data source configuration.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

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 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 Java V2
- AWS SDK for Ruby V3

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 operation to get this value.

Type: String

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

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

Required: Yes

ReferenceSchemaUpdate

Describes the format of the data in the streaming source, and how each data element maps to corresponding columns created in the in-application stream.

Type: SourceSchema 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 object

Required: No

TableNameUpdate

In-application table name that is created by this update.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

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 Java V2
- AWS SDK for Ruby V3

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

Length Constraints: Minimum length of 1. Maximum length of 1024.

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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 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

Length Constraints: Minimum length of 1. Maximum length of 1024.

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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

Length Constraints: Minimum length of 1. Maximum length of 1024.

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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

Length Constraints: Minimum length of 1. Maximum length of 1024.

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:.*`

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 Java V2
- AWS SDK for Ruby V3

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

RecordFormat

Specifies the format of the records on the streaming source.

Type: RecordFormat object

Required: Yes

RecordEncoding

Specifies the encoding of the records in the streaming source. For example, UTF-8.

Type: String

Pattern: `UTF-8`

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 Java V2
- AWS SDK for Ruby V3

Tag

A key-value pair (the value is optional) that you can define and assign to AWS resources. If you specify a tag that already exists, the tag value is replaced with the value that you specify in the request. Note that the maximum number of application tags includes system tags. The maximum number of user-defined application tags is 50. For more information, see Using Tagging.

Contents

Key

The key of the key-value tag.

Type: String

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

Required: Yes

Value

The value of the key-value tag. The value is optional.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

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 Java V2
- AWS SDK for Ruby V3

Document History for Amazon Kinesis Data Analytics

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:** May 8, 2019

Change	Description	Date
Tagging Kinesis Data Analytics Applications	Use application tagging to determine per-application costs, control access, or for user-defined purposes. For more information, see Using Tagging.	May 8, 2019
Logging Kinesis Data Analytics API Calls with AWS CloudTrail	Amazon Kinesis Data Analytics is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Kinesis Data Analytics. For more informati on, see Using AWS CloudTrail.	March 22, 2019
Kinesis Data Analytics available in Frankfurt region	Kinesis Analytics is now available in the Europe (Frankfurt) Region region. For more information, see and Endpoints: Kinesis Data Analytics.	July 18, 2018
Use reference data in the console	You can now work with application reference data in the console. For more	July 13, 2018

Change	Description	Date
	information, see Example: Adding Reference Data to a Kinesis Data Analytics Application .	
Windowed query examples	Example applications for windows and aggregation. For more information, see Examples: Windows and Aggregation .	July 9, 2018
Testing applications	Guidance on testing changes to application schema and code. For more information, see Testing Applications .	July 3, 2018
Example applications for preprocessing data	Additional code samples for REGEX_LOG_PARSE, REGEX_REPLACE, and DateTime operators. For more information, see Examples: Transforming Data .	May 18, 2018
Increase in size of returned rows and SQL code	The limit for the size for a returned row is increased to 512 KB, and the limit for the size of the SQL code in an application is increased to 100 KB. For more informati on, see Limits.	May 2, 2018

Change	Description	Date
AWS Lambda function examples in Java and .NET	Code samples for creating Lambda functions for preprocessing records and for application destinations. For more information, see Creating Lambda Functions for Preprocessing and Creating Lambda Functions for Application Destinations.	March 22, 2018
New `HOTSPOTS` function	Locate and return information about relatively dense regions in your data. For more information, see Example: Detecting Hotspots on a Stream (HOTSPOTS Function).	March 19, 2018
Lambda function as a destination	Send analytics results to a Lambda function as a destination. For more information, see Using a Lambda Function as Output.	December 20, 2017
New `RANDOM_CUT_FOREST_WITH_EXPLANATION` function	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).	November 2, 2017

Change	Description	Date
Schema discovery on static data	Run schema discovery on static data stored in an Amazon S3 bucket. For more information, see Using the Schema Discovery Feature on Static Data.	October 6, 2017
Lambda preprocessing feature	Preprocess records in an input stream with AWS Lambda before analysis. For more information, see Preprocessing Data Using a Lambda Function.	October 6, 2017
Auto scaling applications	Automatically increase the data throughput of your application with auto scaling. For more information, see Automatically Scaling Applications to Increase Throughput.	September 13, 2017
Multiple in-application input streams	Increase application throughput with multiple in-application streams. For more information, see Parallelizing Input Streams for Increased Throughput.	June 29, 2017

Change	Description	Date
Guide to using the AWS Management Console for Kinesis Data Analytics	Edit an inferred schema and SQL code using the schema editor and SQL editor in the Kinesis Data Analytics console. For more informati on, see Step 4 (Optional) Edit the Schema and SQL Code Using the Console.	April 7, 2017
Public release	Public release of the *Amazon Kinesis Data Analytics Developer Guide*.	August 11, 2016
Preview release	Preview release of the *Amazon Kinesis Data Analytics Developer Guide*.	January 29, 2016

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.