Amazon Kinesis Data Analytics
Developer Guide

aws
# Table of Contents

What Is Kinesis Data Analytics for Java Applications? ......................................................... 1
Getting Started ........................................................................................................................ 1
How it Works ............................................................................................................................ 2
Creating Applications
  Building your Kinesis Data Analytics Application Code .................................................. 3
  Creating your Kinesis Data Analytics Application ............................................................ 4
  Starting your Kinesis Data Analytics Application ............................................................. 5
  Verifying your Kinesis Data Analytics Application ........................................................... 5
Connectors ............................................................................................................................... 5
  Available Connectors ........................................................................................................ 6
  Sources .............................................................................................................................. 6
  Sinks ................................................................................................................................. 6
  Asynchronous I/O ............................................................................................................ 9
Operators ................................................................................................................................. 9
  Transform Operators ...................................................................................................... 9
  Aggregation Operators ................................................................................................. 9
Runtime Properties .............................................................................................................. 10
  Working with Runtime Properties in the Console ........................................................... 10
  Working with Runtime Properties in the CLI ................................................................. 10
  Accessing Runtime Properties in a Kinesis Data Analytics for Java application .......... 12
Timestamps ............................................................................................................................ 13
Fault Tolerance ................................................................................................................... 13
  Configuring Checkpointing ............................................................................................ 14
  Checkpointing API Examples .......................................................................................... 14
  Snapshots .......................................................................................................................... 16
Scaling ................................................................................................................................ 19
  Configuring Application Parallelism and ParallelismPerKPU .......................................... 19
  Allocating Kinesis Processing Units ............................................................................... 20
  Automatic Scaling ........................................................................................................... 20
  Updating Your Application's Parallelism ......................................................................... 21
Tagging ................................................................................................................................ 22
  Adding Tags when an Application is Created .................................................................. 22
  Adding or Updating Tags for an Existing Application ..................................................... 23
  Listing Tags for an Application ....................................................................................... 23
  Removing Tags from an Application ................................................................................ 23
Getting Started ..................................................................................................................... 24
Application Components .................................................................................................... 24
Prerequisites .......................................................................................................................... 24
Step 1: Set Up an Account ................................................................................................. 25
  Sign Up for AWS ............................................................................................................. 25
  Create an IAM User ....................................................................................................... 25
  Next Step ....................................................................................................................... 27
Step 2: Set Up the AWS CLI .............................................................................................. 27
  Next Step ....................................................................................................................... 28
Step 3: Create an Application ............................................................................................. 28
  Create Two Amazon Kinesis Data Streams ..................................................................... 28
  Write Sample Records to the Input Stream ...................................................................... 29
  Download and Examine the Apache Flink Streaming Java Code .................................... 30
  Compile the Application Code ......................................................................................... 30
  Upload the Apache Flink Streaming Java Code ................................................................ 31
  Create and Run the Kinesis Data Analytics Application .................................................. 31
Step 4: Clean Up ................................................................................................................ 42
  Delete Your Kinesis Data Analytics Application ............................................................ 42
  Delete Your Kinesis Data Streams ................................................................................... 42
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>313</td>
</tr>
<tr>
<td>Create Source Kinesis Stream</td>
<td>103</td>
</tr>
<tr>
<td>Create and Update IAM Roles and Policies</td>
<td>104</td>
</tr>
<tr>
<td>Update the Python Script</td>
<td>106</td>
</tr>
<tr>
<td>Update the Java Application</td>
<td>107</td>
</tr>
<tr>
<td>Build, Upload, and Run the Application</td>
<td>108</td>
</tr>
<tr>
<td>Security</td>
<td>109</td>
</tr>
<tr>
<td>Data Protection</td>
<td>109</td>
</tr>
<tr>
<td>Data Encryption</td>
<td>109</td>
</tr>
<tr>
<td>Identity and Access Management</td>
<td>110</td>
</tr>
<tr>
<td>Trust Policy</td>
<td>111</td>
</tr>
<tr>
<td>Permissions Policy</td>
<td>111</td>
</tr>
<tr>
<td>Monitoring</td>
<td>113</td>
</tr>
<tr>
<td>Compliance Validation</td>
<td>113</td>
</tr>
<tr>
<td>Resilience</td>
<td>113</td>
</tr>
<tr>
<td>Disaster Recovery</td>
<td>114</td>
</tr>
<tr>
<td>Versioning</td>
<td>114</td>
</tr>
<tr>
<td>Infrastructure Security</td>
<td>114</td>
</tr>
<tr>
<td>Security Best Practices</td>
<td>115</td>
</tr>
<tr>
<td>Implement least privilege access</td>
<td>115</td>
</tr>
<tr>
<td>Use IAM roles to access other AWS services</td>
<td>115</td>
</tr>
<tr>
<td>Implement Server-Side Encryption in Dependent Resources</td>
<td>115</td>
</tr>
<tr>
<td>Use CloudTrail to Monitor API Calls</td>
<td>115</td>
</tr>
<tr>
<td>Logging and Monitoring</td>
<td>116</td>
</tr>
<tr>
<td>Setting Up Logging</td>
<td>116</td>
</tr>
<tr>
<td>Setting Up CloudWatch Logging Using the Console</td>
<td>117</td>
</tr>
<tr>
<td>Setting Up CloudWatch Logging Using the CLI</td>
<td>118</td>
</tr>
<tr>
<td>Application Monitoring Levels</td>
<td>121</td>
</tr>
<tr>
<td>Logging Best Practices</td>
<td>122</td>
</tr>
<tr>
<td>Logging Troubleshooting</td>
<td>122</td>
</tr>
<tr>
<td>Next Step</td>
<td>123</td>
</tr>
<tr>
<td>Analyzing Logs</td>
<td>123</td>
</tr>
<tr>
<td>Run a Sample Query</td>
<td>123</td>
</tr>
<tr>
<td>Example Queries</td>
<td>124</td>
</tr>
<tr>
<td>Metrics and Dimensions</td>
<td>125</td>
</tr>
<tr>
<td>Application Metrics</td>
<td>126</td>
</tr>
<tr>
<td>Kinesis Data Streams Connector Metrics</td>
<td>128</td>
</tr>
<tr>
<td>Amazon MSK Connector Metrics</td>
<td>128</td>
</tr>
<tr>
<td>Viewing CloudWatch Metrics</td>
<td>129</td>
</tr>
<tr>
<td>Metrics</td>
<td>130</td>
</tr>
<tr>
<td>Writing Custom Messages</td>
<td>131</td>
</tr>
<tr>
<td>Write to CloudWatch Logs Using Log4J</td>
<td>131</td>
</tr>
<tr>
<td>Write to CloudWatch Logs Using SLF4J</td>
<td>132</td>
</tr>
<tr>
<td>Using AWS CloudTrail</td>
<td>132</td>
</tr>
<tr>
<td>Kinesis Data Analytics Information in CloudTrail</td>
<td>133</td>
</tr>
<tr>
<td>Understanding Kinesis Data Analytics Log File Entries</td>
<td>133</td>
</tr>
<tr>
<td>Limits</td>
<td>136</td>
</tr>
<tr>
<td>Best Practices</td>
<td>137</td>
</tr>
<tr>
<td>Fault Tolerance: Checkpoints and Savepoints</td>
<td>137</td>
</tr>
<tr>
<td>Performance and Parallelism</td>
<td>137</td>
</tr>
<tr>
<td>Logging</td>
<td>138</td>
</tr>
<tr>
<td>Coding</td>
<td>138</td>
</tr>
<tr>
<td>Earlier Versions</td>
<td>139</td>
</tr>
<tr>
<td>Building Applications with Apache Flink 1.6.2</td>
<td>139</td>
</tr>
<tr>
<td>Upgrading Applications</td>
<td>140</td>
</tr>
<tr>
<td>Available Connectors in Apache Flink 1.6.2</td>
<td>140</td>
</tr>
<tr>
<td>Getting Started: Flink 1.6.2</td>
<td>140</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>DescribeApplicationSnapshot</td>
<td>181</td>
</tr>
<tr>
<td>DiscoverInputSchema</td>
<td>181</td>
</tr>
<tr>
<td>ListApplications</td>
<td>182</td>
</tr>
<tr>
<td>ListApplicationSnapshots</td>
<td>182</td>
</tr>
<tr>
<td>StartApplication</td>
<td>182</td>
</tr>
<tr>
<td>StopApplication</td>
<td>182</td>
</tr>
<tr>
<td>UpdateApplication</td>
<td>183</td>
</tr>
<tr>
<td>API Reference</td>
<td>184</td>
</tr>
</tbody>
</table>
What Is Amazon Kinesis Data Analytics for Java Applications?

With Amazon Kinesis Data Analytics for Java Applications, you can use Java to process and analyze streaming data. The service enables you to author and run Java code against streaming sources to perform time-series analytics, feed real-time dashboards, and create real-time metrics.

You can build Java applications in Kinesis Data Analytics using open-source libraries based on Apache Flink. Apache Flink is a popular framework and engine for processing data streams.

Kinesis Data Analytics provides the underlying infrastructure for your Apache Flink applications. It handles core capabilities like provisioning compute resources, parallel computation, automatic scaling, and application backups (implemented as checkpoints and snapshots). You can use the high-level Flink programming features (such as operators, sources, and sinks) in the same way that you use them when hosting the Flink infrastructure yourself.

Getting Started

You can start by creating a Kinesis Data Analytics application that continuously reads and processes streaming data. Then, author your Java code using your IDE of choice, and test it with live streaming data. You can also configure destinations where you want Kinesis Data Analytics to send the results.

To get started, we recommend that you read the following sections:

- Kinesis Data Analytics for Java Applications: How it Works (p. 2)
- Getting Started with Amazon Kinesis Data Analytics for Java Applications (p. 24)
Kinesis Data Analytics for Java Applications: How it Works

Kinesis Data Analytics is a fully managed AWS service that enables you to use an Apache Flink application to process streaming data and to produce output. Your application then writes output to a configured destination.

You build your Apache Flink application locally, and then make your application code available to the Kinesis Data Analytics service by uploading it to an Amazon S3 bucket. Applications are built using the Flink programming model. Most Kinesis Data Analytics for Java applications will primarily use the DataStream API. The other Flink APIs are also available for you to use, but they are less commonly used in building streaming applications.

The Apache Flink programming model is based on two building blocks:

- **Data stream**: The structured representation of a continuous flow of data records.
- **Transformation operator**: Takes one or more data streams as input, and produces one or more data streams as output.

You connect data streams with external data by adding a source or a sink, called a connector. A typical application consists of at least one data stream with a source, a data stream with one or more operators, and at least one data sink.

**This section contains the following topics:**

- Creating a Kinesis Data Analytics for Java Application (p. 2)
- Using Connectors to Move Data in Kinesis Data Analytics for Java Applications (p. 5)
- Transforming Data Using Operators in Kinesis Data Analytics for Java Applications (p. 9)
- Runtime Properties in Kinesis Data Analytics for Java Applications (p. 10)
- Tracking Events in Kinesis Data Analytics for Java Applications (p. 13)
- Implementing Fault Tolerance in Kinesis Data Analytics for Java Applications (p. 13)
- Application Scaling in Kinesis Data Analytics for Java Applications (p. 19)
- Using Tagging (p. 22)

Creating a Kinesis Data Analytics for Java Application

This topic contains information about creating a Amazon Kinesis Data Analytics for Java application.

**This topic contains the following sections:**

- Building your Kinesis Data Analytics Application Code (p. 3)
- Creating your Kinesis Data Analytics Application (p. 4)
Building your Kinesis Data Analytics Application Code

This section describes the components you use to build the application code for your Kinesis Data Analytics for Java application.

We recommend that you use the latest supported version of Apache Flink for your application code. The latest version of Apache Flink that Kinesis Data Analytics supports is 1.8.2. For information about upgrading Kinesis Data Analytics applications, see Upgrading Applications (p. 140).

You build your application code using Apache Maven. An Apache Maven project uses a `pom.xml` file to specify the versions of components that it uses.

Note
Kinesis Data Analytics supports JAR files up to 512 MB in size. If you use a JAR file larger than this, your application will fail to start.

Use the following component versions for Kinesis Data Analytics applications:

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.8 (recommended)</td>
</tr>
<tr>
<td>AWS Java SDK</td>
<td>1.11.379</td>
</tr>
<tr>
<td>Kinesis Data Analytics for Flink Runtime (aws-kinesisanalytics-runtime)</td>
<td>1.1.0</td>
</tr>
<tr>
<td>Kinesis Data Analytics Flink Connectors (aws-kinesisanalytics-flink)</td>
<td>1.1.0</td>
</tr>
<tr>
<td>Apache Maven</td>
<td>3.1</td>
</tr>
</tbody>
</table>

For an example of a `pom.xml` file for a Kinesis Data Analytics application that uses Apache Flink version 1.8.2, see the Kinesis Data Analytics Getting Started Application.

Specifying your Application's Apache Flink Version

When using Kinesis Data Analytics for Flink Runtime version 1.1.0, you specify the version of Apache Flink that your application uses when you compile your application. You provide the version of Apache Flink with the `-Dflink.version` parameter as follows:

```
mvn package -Dflink.version=1.8.2
```

For building applications with older versions of Apache Flink, see Earlier Versions (p. 139).

Using the Apache Flink Kinesis Streams Connector

In order for your application to use the Apache Flink Kinesis connector, you must download, compile, and install Apache Flink. This connector is used to consume data from a Kinesis stream used as an application source, or to write data to a Kinesis stream used for application output.
To download and install the Apache Flink source code, do the following:

1. Ensure that you have Apache Maven installed, and your `JAVA_HOME` environment variable points to a JDK rather than a JRE. You can test your Apache Maven install with the following command:
   
   ```
   mvn -version
   ```

2. Download the Apache Flink version 1.8.2 source code:
   
   ```
   wget https://archive.apache.org/dist/flink/flink-1.8.2/flink-1.8.2-src.tgz
   ```

3. Uncompress the Apache Flink source code:
   
   ```
   tar -xvf flink-1.8.2-src.tgz
   ```

4. Change to the Apache Flink source code directory:
   
   ```
   cd flink-1.8.2
   ```

5. Compile and install Apache Flink:
   
   ```
   mvn clean install -Pinclude-kinesis -DskipTests
   ```

---

### Creating your Kinesis Data Analytics Application

Once you have built your application code, you do the following to create your Kinesis Data Analytics application:

- **Upload your Application code**: Upload your application code to an Amazon S3 bucket. You specify the S3 bucket name and object name of your application code when you create your application. For a tutorial that shows how to upload your application code, see the section called “Upload the Apache Flink Streaming Java Code” (p. 31) in the Getting Started (p. 24) tutorial.

- **Create your Kinesis Data Analytics application**: Use one of the following methods to create your Kinesis Data Analytics application:
  
  - **Create your Kinesis Data Analytics application using the AWS console**: You can create and configure your application using the AWS console.

    When you create your application using the console, your application's dependent resources (such as CloudWatch Logs streams, IAM roles, and IAM policies) are created for you.

    When you create your application using the console, you specify what version of Apache Flink your application uses by selecting it from the pull-down on the **Kinesis Analytics - Create application** page.

    For a tutorial about how to use the console to create an application, see the section called “Create and Run the Application (Console)” (p. 32) in the Getting Started (p. 24) tutorial.

  - **Create your Kinesis Data Analytics application using the AWS CLI**: You can create and configure your application using the AWS CLI.

    When you create your application using the CLI, you must also create your application's dependent resources (such as CloudWatch Logs streams, IAM roles, and IAM policies) manually.

    When you create your application using the CLI, you specify what version of Apache Flink your application uses by using the `RuntimeEnvironment` parameter of the `CreateApplication` action.
Starting your Kinesis Data Analytics Application

After you have built your application code, uploaded it to S3, and created your Kinesis Data Analytics application, you then start your application. Starting a Kinesis Data Analytics application typically takes several minutes.

Use one of the following methods to start your application:

- **Start your Kinesis Data Analytics application using the AWS console**: You can run your application by choosing Run on your application's page in the AWS console.
- **Start your Kinesis Data Analytics application using the AWS API**: You can run your application using the StartApplication action.

Verifying your Kinesis Data Analytics Application

You can verify that your application is working in the following ways:

- **Using CloudWatch Logs**: You can use CloudWatch Logs and CloudWatch Logs Insights to verify that your application is running properly. For information about using CloudWatch Logs with your Kinesis Data Analytics application, see Logging and Monitoring (p. 116).
- **Using CloudWatch Metrics**: You can use CloudWatch Metrics to monitor your application's activity, or activity in the resources your application uses for input or output (such as Kinesis streams, Kinesis Data Firehose delivery streams, or Amazon S3 buckets.) For more information about CloudWatch metrics, see Working with Metrics in the Amazon CloudWatch User Guide.
- **Monitoring Output Locations**: If your application writes output to a location (such as an Amazon S3 bucket or database), you can monitor that location for written data.

Using Connectors to Move Data in Kinesis Data Analytics for Java Applications

In Amazon Kinesis Data Analytics for Java Applications, **connectors** are software components that move data into and out of an Amazon Kinesis Data Analytics application. Connectors are flexible integrations that enable you to read from files and directories. Connectors consist of complete modules for interacting with AWS services and third-party systems.

Types of connectors include the following:

- **Sources (p. 6)**: Provide data to your application from a Kinesis data stream, file, or other data source.
- **Sinks (p. 6)**: Send data from your application to a Kinesis data stream, Kinesis Data Firehose delivery stream, or other data destination.
- **Asynchronous I/O (p. 9)**: Provides asynchronous access to a data source (such as a database) to enrich stream events.
Available Connectors

The Apache Flink framework contains connectors for accessing data from a variety of sources. For information about connectors available in the Apache Flink framework, see Connectors in the Apache Flink documentation.

Adding Streaming Data Sources to Kinesis Data Analytics for Java Applications

Apache Flink provides connectors for reading from files, sockets, collections, and custom sources. In your application code, you use an Apache Flink source to receive data from a stream. This section describes the sources that are available for AWS services.

Kinesis Data Streams

The FlinkKinesisConsumer source provides streaming data to your application from an Amazon Kinesis data stream.

Creating a FlinkKinesisConsumer

The following code example demonstrates creating a FlinkKinesisConsumer:

```java
Properties inputProperties = new Properties();
inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "LATEST");
DataStream<String> input = env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new SimpleStringSchema(), inputProperties));
```

For more information about using a FlinkKinesisConsumer, see Download and Examine the Apache Flink Streaming Java Code (p. 30).

Writing Data Using Sinks in Kinesis Data Analytics for Java Applications

In your application code, you use an Apache Flink sink to write data from an Apache Flink stream to an AWS service, such as Kinesis Data Streams.

Apache Flink provides sinks for files, sockets, and custom sinks. The following sinks are available for AWS services:

Kinesis Data Streams

Apache Flink provides information about the Kinesis Data Streams Connector in the Apache Flink documentation.

Note
In order to use the Kinesis Data Streams connector, you need to download the source code and build it locally.

Amazon S3

You can use the Apache Flink StreamingFileSink to write objects to an Amazon S3 bucket.

For an example about how to write objects to S3, see the section called “S3 Sink” (p. 62).
Kinesis Data Firehose

The FlinkKinesisFirehoseProducer is a reliable, scalable Apache Flink sink for storing application output using the Kinesis Data Firehose service. This section describes how to set up a Maven project to create and use a FlinkKinesisFirehoseProducer.

Topics
- Creating a FlinkKinesisFirehoseProducer (p. 7)
- FlinkKinesisFirehoseProducer Code Example (p. 7)

Creating a FlinkKinesisFirehoseProducer

The following code example demonstrates creating a FlinkKinesisFirehoseProducer:

```java
Properties outputProperties = new Properties();
outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
FlinkKinesisFirehoseProducer<String> sink = new FlinkKinesisFirehoseProducer<>(outputStreamName, new SimpleStringSchema(), outputProperties);
```

FlinkKinesisFirehoseProducer Code Example

The following code example demonstrates how to create and configure a FlinkKinesisFirehoseProducer and send data from an Apache Flink data stream to the Kinesis Data Firehose service.

```java
package com.amazonaws.services.kinesisanalytics;
import com.amazonaws.services.kinesisanalytics.flink.connectors.config.ProducerConfigConstants;
import com.amazonaws.services.kinesisanalytics.flink.connectors.producer.FlinkKinesisFirehoseProducer;
import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;
import org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfig Constants;
import java.io.IOException;
import java.util.Map;
import java.util.Properties;

public class StreamingJob {
    private static final String region = "us-east-1";
    private static final String inputStreamName = "ExampleInputStream";
    private static final String outputStreamName = "ExampleOutputStream";

    public static DataStream<String> createSourceFromStaticConfig(StreamExecutionEnvironment env) {
        Properties inputProperties = new Properties();
        inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
        inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "LATEST");
```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new SimpleStringSchema(), inputProperties));}

private static DataStream<String> createSourceFromApplicationProperties(StreamExecutionEnvironment env) throws IOException {
    Map<String, Properties> applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties();
    return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new SimpleStringSchema(),
        applicationProperties.get("ConsumerConfigProperties")));}

private static FlinkKinesisFirehoseProducer<String> createFirehoseSinkFromStaticConfig() {
    /*
     * com.amazonaws.services.kinesisanalytics.flink.connectors.config.ProducerConfigConstants
     * lists of all of the properties that firehose sink can be configured with.
     */
    Properties outputProperties = new Properties();
    outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
    FlinkKinesisFirehoseProducer<String> sink = new FlinkKinesisFirehoseProducer<>(outputStreamName, new SimpleStringSchema(), outputProperties);
    ProducerConfigConstants config = new ProducerConfigConstants();
    return sink;
}

private static FlinkKinesisFirehoseProducer<String> createFirehoseSinkFromApplicationProperties() throws IOException {
    /*
     * com.amazonaws.services.kinesisanalytics.flink.connectors.config.ProducerConfigConstants
     * lists of all of the properties that firehose sink can be configured with.
     */
    Map<String, Properties> applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties();
    FlinkKinesisFirehoseProducer<String> sink = new FlinkKinesisFirehoseProducer<>(outputStreamName, new SimpleStringSchema(), applicationProperties.get("ProducerConfigProperties"));
    return sink;
}

public static void main(String[] args) throws Exception {
    // set up the streaming execution environment
    final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

    /* if you would like to use runtime configuration properties, uncomment the lines below
     * DataStream<String> input = createSourceFromApplicationProperties(env);
     */
    DataStream<String> input = createSourceFromStaticConfig(env);

    // Kinesis Firehose sink
    input.addSink(createFirehoseSinkFromStaticConfig());

    // If you would like to use runtime configuration properties, uncomment the lines below
    // input.addSink(createFirehoseSinkFromApplicationProperties());
Asynchronous I/O

env.execute("Flink Streaming Java API Skeleton");
}
}

For a complete tutorial about how to use the Kinesis Data Firehose sink, see the section called “Kinesis Data Firehose Sink” (p. 79).

Using Asynchronous I/O in Kinesis Data Analytics for Java Applications

An Asynchronous I/O operator enriches stream data using an external data source such as a database. Kinesis Data Analytics enriches the stream events asynchronously so that requests can be batched for greater efficiency.

For more information, see Asynchronous I/O in the Apache Flink Documentation.

Transforming Data Using Operators in Kinesis Data Analytics for Java Applications

To transform incoming data in a Kinesis Data Analytics for Java application, you use an Apache Flink operator. An Apache Flink operator transforms one or more data streams into a new data stream. The new data stream contains modified data from the original data stream. Apache Flink provides more than 25 pre-built stream processing operators. For more information, see Operators in the Apache Flink Documentation.

This topic contains the following sections:
- Transform Operators (p. 9)
- Aggregation Operators (p. 9)

Transform Operators

The following is an example of a simple text transformation on one of the fields of a JSON data stream.

This code creates a transformed data stream. The new data stream has the same data as the original stream, with the string “Company” appended to the contents of the TICKER field.

DataStream<ObjectNode> output = input.map(
    new MapFunction<ObjectNode, ObjectNode>() {
        @Override
        public ObjectNode map(ObjectNode value) throws Exception {
            return value.put("TICKER", value.get("TICKER").asText() + " Company");
        }
    });

Aggregation Operators

The following is an example of an aggregation operator. The code creates an aggregated data stream. The operator creates a 5-second tumbling window and returns the sum of the PRICE values for the records in the window with the same TICKER value.
DataStream<ObjectNode> output = input.keyBy(node -> node.get("TICKER").asText())
    .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
    .reduce((node1, node2) -> {
        double priceTotal = node1.get("PRICE").asDouble() + node2.get("PRICE").asDouble();
        node1.replace("PRICE", JsonNodeFactory.instance.numberNode(priceTotal));
        return node1;
    });

For a complete code example that uses operators, see Getting Started (p. 24). Source code for the
Getting Started application is available at Getting Started in the Kinesis Data Analytics Java Examples
GitHub repository.

Runtime Properties in Kinesis Data Analytics for Java Applications

You can use runtime properties to configure your application without recompiling your application code.

This topic contains the following sections:
- Working with Runtime Properties in the Console (p. 10)
- Working with Runtime Properties in the CLI (p. 10)
- Accessing Runtime Properties in a Kinesis Data Analytics for Java application (p. 12)

Working with Runtime Properties in the Console

You can add, update, or remove runtime properties from your Kinesis Data Analytics for Java application
using the console.

Note
You can't add runtime properties when you create an application in the Kinesis Data Analytics
console.

Update Runtime Properties for a Kinesis Data Analytics for Java application

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose your Kinesis Data Analytics for Java application. Choose Application details.
3. On the page for your application, choose Configure.
4. Expand the Properties section.
5. Use the controls in the Properties section to define a property group with key-value pairs. Use these
   controls to add, update, or remove property groups and runtime properties.
6. Choose Update.

Working with Runtime Properties in the CLI

You can add, update, or remove runtime properties using the AWS CLI.

This section includes example requests for API actions for configuring runtime properties for an
application. For information about how to use a JSON file for input for an API action, see Kinesis Data
Analytics API Example Code (p. 175).

Note
Replace the sample account ID (012345678901) in the examples following with your account ID.
Adding Runtime Properties when Creating an Application

The following example request for the `CreateApplication` action adds two runtime property groups (ProducerConfigProperties and ConsumerConfigProperties) when you create an application:

```
{
  "ApplicationName": "MyApplication",
  "ApplicationDescription": "my java test app",
  "RuntimeEnvironment": "FLINK-1_8",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/KA-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "java-getting-started-1.0.jar"
        }
      },
      "CodeContentType": "ZIPFILE"
    },
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap": {
            "flink.stream.initpos": "LATEST",
            "aws.region": "us-west-2",
            "AggregationEnabled": "false"
          }
        },
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-west-2"
          }
        }
      ]
    }
  }
}
```

Adding and Updating Runtime Properties in an Existing Application

The following example request for the `UpdateApplication` action adds or updates runtime properties for an existing application:

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 2,
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap": {
            "flink.stream.initpos": "LATEST",
            "aws.region": "us-west-2",
            "AggregationEnabled": "false"
          }
        },
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-west-2"
          }
        }
      ]
    }
  }
}
```
Accessing Runtime Properties in a Kinesis Data Analytics for Java application

Note
If you use a key that has no corresponding runtime property in a property group, Kinesis Data Analytics adds the key-value pair as a new property. If you use a key for an existing runtime property in a property group, Kinesis Data Analytics updates the property value.

Removing Runtime Properties

The following example request for the UpdateApplication action removes all runtime properties and property groups from an existing application:

```json
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 3,
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": []
    }
  }
}
```

Important
If you omit an existing property group or an existing property key in a property group, that property group or property is removed.

Accessing Runtime Properties in a Kinesis Data Analytics for Java application

You retrieve runtime properties in your Java application code using the static `KinesisAnalyticsRuntime.getApplicationProperties()` method, which returns a `Map<String, Properties>` object.

The following Java code example retrieves runtime properties for your application:

```java
Map<String, Properties> applicationProperties =
    KinesisAnalyticsRuntime.getApplicationProperties();
```

You retrieve a property group (as a `java.util.Properties` object) as follows:

```java
Properties consumerProperties = applicationProperties.get("ConsumerConfigProperties");
```

You typically configure an Flink source or sink by passing in the `Properties` object without needing to retrieve the individual properties. The following code example demonstrates how to create an Flink source by passing in a `Properties` object retrieved from runtime properties:

```java
private static FlinkKinesisProducer<String> createSinkFromApplicationProperties() throws IOException {
    // Code...
```
Map<String, Properties> applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties();
FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<gt;(new SimpleStringSchema(),
    applicationProperties.get("ProducerConfigProperties"));
    sink.setDefaultStream(outputStreamName);
    sink.setDefaultPartition("0");
    return sink;
}

For a complete code example that uses runtime properties, see Getting Started (p. 24). Source code for the Getting Started application is available at Getting Started in the Kinesis Data Analytics Java Examples GitHub repository.

Tracking Events in Kinesis Data Analytics for Java Applications

Kinesis Data Analytics for Java Applications tracks events using the following timestamps:

- **Processing Time**: Refers to the system time of the machine that is executing the respective operation.
- **Event Time**: Refers to the time that each individual event occurred on its producing device.
- **Ingestion Time**: Refers to the time that events enter the Kinesis Data Analytics service.

You set the time used by the streaming environment using `setStreamTimeCharacteristic`:

```
env.setStreamTimeCharacteristic(TimeCharacteristic.ProcessingTime);
env.setStreamTimeCharacteristic(TimeCharacteristic.IngestionTime);
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
```

For more information about timestamps, see Event Time in the Apache Flink Documentation.

Implementing Fault Tolerance in Kinesis Data Analytics for Java Applications

Checkpointing is the method that is used for implementing fault tolerance in Amazon Kinesis Data Analytics for Java Applications. A checkpoint is an up-to-date backup of a running application that is used to recover immediately from an unexpected application disruption or failover.

For details on checkpointing in Apache Flink applications, see Checkpointing in the Apache Flink Documentation.

A snapshot is a manually created and managed backup of application state. Snapshots let you restore your application to a previous state by calling UpdateApplication. For more information, see Managing Application Backups Using Snapshots (p. 16).

If checkpointing is enabled for your application, then the service provides fault tolerance by creating and loading backups of application data in the event of unexpected application restarts. These unexpected application restarts could be caused by unexpected job restarts, instance failures, etc. This gives the application the same semantics as failure-free execution during these restarts.
If snapshots are enabled for the application, and configured using the application's ApplicationRestoreConfiguration, then the service provides exactly-once processing semantics during application updates, or during service-related scaling or maintenance.

### Configuring Checkpointing in Kinesis Data Analytics for Java Applications

You can configure your application's checkpointing behavior. You can define whether it persists the checkpointing state, how often it saves its state to checkpoints, and the minimum interval between the end of one checkpoint operation and the beginning of another.

You configure the following settings using the CreateApplication or UpdateApplication API operations:

- **CheckpointingEnabled** — Indicates whether checkpointing is enabled in the application.
- **CheckpointInterval** — Contains the time in milliseconds between checkpoint (persistence) operations.
- **ConfigurationType** — Set this value to DEFAULT to use the default checkpointing behavior. Set this value to CUSTOM to configure other values.

**Note**

The default checkpoint behavior is as follows:

- **CheckpointingEnabled**: true
- **CheckpointInterval**: 60000
- **MinPauseBetweenCheckpoints**: 5000

If ConfigurationType is set to DEFAULT, the preceding values will be used, even if they are set to other values using either using the AWS CLI, or by setting the values in the application code.

- **MinPauseBetweenCheckpoints** — The minimum time in milliseconds between the end of one checkpoint operation and the start of another. Setting this value prevents the application from checkpointing continuously when a checkpoint operation takes longer than the CheckpointInterval.

### Checkpointing API Examples

This section includes example requests for API actions for configuring checkpointing for an application. For information about how to use a JSON file for input for an API action, see Kinesis Data Analytics API Example Code (p. 175).

#### Configure Checkpointing for a New Application

The following example request for the CreateApplication action configures checkpointing when you are creating an application:

```json
{
    "ApplicationName": "MyApplication",
    "RuntimeEnvironment": "FLINK-1_8",
    "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::mybucket",
                    "FileKey": "myflink.jar",
                    "ObjectVersion": "AbCdEfGhIjKlMnOpQrStUvWxYz12345"
                }
            }
        }
    }
}
```
Disabling Checkpointing for a New Application

The following example request for the `CreateApplication` action disables checkpointing when you are creating an application:

```json
{
  "ApplicationName": "MyApplication",
  "RuntimeEnvironment": "FLINK-1_8",
  "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::mybucket",
          "FileKey": "myflink.jar",
          "ObjectVersion": "AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "FlinkApplicationConfiguration": {
        "CheckpointConfiguration": {
          "CheckpointingEnabled": "false"
        }
      }
    }
  }
}
```

Disabling Checkpointing for an Existing Application

The following example request for the `UpdateApplication` action disables checkpointing for an existing application:

```json
{
  "ApplicationName": "MyApplication",
  "ApplicationConfigurationUpdate": {
    "FlinkApplicationConfigurationUpdate": {
      "CheckpointConfigurationUpdate": {
        "CheckpointingEnabledUpdate": "false"
      }
    }
  }
}
```

Configuring Checkpointing for an Existing Application

The following example request for the `UpdateApplication` action configures checkpointing for an existing application:

```json
{
  "ApplicationName": "MyApplication",
  "ApplicationConfigurationUpdate": {
    "FlinkApplicationConfigurationUpdate": {
      "CheckpointConfigurationUpdate": {
        "CheckpointingEnabledUpdate": "true",
        "CheckpointIntervalUpdate": 20000,
        "ConfigurationTypeUpdate": "CUSTOM",
        "MinPauseBetweenCheckpointsUpdate": 10000
      }
    }
  }
}
```

```
Managing Application Backups Using Snapshots

A **snapshot** is the Kinesis Data Analytics implementation of an Apache Flink **Savepoint**. A snapshot is a user- or service-triggered, created, and managed backup of the application state. For information about Apache Flink Savepoints, see **Savepoints** in the Apache Flink Documentation. Using snapshots, you can restart an application from a particular snapshot of application state.

**Note**
For your application to restart properly with correct state data, we recommend that your application create a snapshot every 5-10 minutes.

In Kinesis Data Analytics for Java Applications, you manage snapshots using the following API actions:

- CreateApplicationSnapshot
- DeleteApplicationSnapshot
- DescribeApplicationSnapshot
- ListApplicationSnapshots

For the per-application limit on the number of snapshots, see **Limits** (p. 136). If your application reaches the limit on snapshots, then creating a snapshot fails with a LimitExceededException.

Kinesis Data Analytics never deletes user-created snapshots. You must manually delete your snapshots using the DeleteApplicationSnapshot action.

To load a saved snapshot of application state when starting an application, use the ApplicationRestoreConfiguration parameter of the StartApplication or UpdateApplication action.

**This topic contains the following sections:**
- Automatic Snapshot Creation (p. 16)
- Restoring From a Snapshot That Contains Incompatible State Data (p. 17)
- Snapshot API Examples (p. 17)

### Automatic Snapshot Creation

If SnapshotsEnabled is set to `true` in the ApplicationSnapshotConfiguration for the application, Kinesis Data Analytics automatically creates and uses a snapshot when the application is updated, scaled, or stopped to provide exactly-once processing semantics.

**Note**
Setting ApplicationSnapshotConfiguration::SnapshotsEnabled to `false` will lead to data loss during application updates.
The automatically created snapshot has the following qualities:

- The snapshot is managed by the service, is not exposed to the user, and does not count against your snapshot limit.
- Only one snapshot is automatically maintained.

**Restoring From a Snapshot That Contains Incompatible State Data**

Because snapshots contain information about operators, restoring state data from a snapshot for an operator that has changed since the previous application version may have unexpected results. An application will fault if it attempts to restore state data from a snapshot that does not correspond to the current operator. The faulted application will be stuck in either the STOPPING or UPDATING state.

To allow an application to restore from a snapshot that contains incompatible state data, set the `AllowNonRestoredState` parameter of the `FlinkRunConfiguration` to `true` using the `UpdateApplication` action.

You will see the following behavior when an application is restored from an obsolete snapshot:

- **Operator added**: If a new operator is added, the savepoint has no state data for the new operator. No fault will occur, and it is not necessary to set `AllowNonRestoredState`.
- **Operator deleted**: If an existing operator is deleted, the savepoint has state data for the missing operator. A fault will occur unless `AllowNonRestoredState` is set to `true`.
- **Operator modified**: If compatible changes are made, such as changing a parameter's type to a compatible type, the application can restore from the obsolete snapshot. For more information about restoring from snapshots, see Savepoints in the Apache Flink Documentation. An application that uses Apache Flink version 1.8 can possibly be restored from a snapshot with a different schema. An application that uses Apache Flink version 1.6 cannot be restored.

If you need to resume an application that is incompatible with existing savepoint data, we recommend that you skip restoring from the snapshot by setting the `ApplicationRestoreType` parameter of the `StartApplication` action to `SKIP_RESTORE_FROM_SNAPSHOT`.

For more information about how Apache Flink deals with incompatible state data, see State Schema Evolution in the Apache Flink Documentation.

**Snapshot API Examples**

This section includes example requests for API actions for using snapshots with an application. For information about how to use a JSON file for input for an API action, see Kinesis Data Analytics API Example Code (p. 175).

**Enable Snapshots for an Application**

The following example request for the `UpdateApplication` action enables snapshots for an application:

```json
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "ApplicationSnapshotConfigurationUpdate": {
         "SnapshotsEnabledUpdate": "true"
      }
   }
}```
Create a Snapshot

The following example request for the CreateApplicationSnapshot action creates a snapshot of the current application state:

```json
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MyCustomSnapshot"
}
```

List Snapshots for an Application

The following example request for the ListApplicationSnapshots action lists the first 50 snapshots for the current application state:

```json
{
   "ApplicationName": "MyApplication",
   "Limit": 50
}
```

List Details for an Application Snapshot

The following example request for the DescribeApplicationSnapshot action lists details for a specific application snapshot:

```json
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MyCustomSnapshot"
}
```

Delete a Snapshot

The following example request for the DeleteApplicationSnapshot action deletes a previously saved snapshot. You can get the SnapshotCreationTimestamp value using either ListApplicationSnapshots or DeleteApplicationSnapshot:

```json
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MyCustomSnapshot",
   "SnapshotCreationTimestamp": 12345678901.0,
}
```

Restart an Application Using a Named Snapshot

The following example request for the StartApplication action starts the application using the saved state from a specific snapshot:

```json
{
   "ApplicationName": "MyApplication",
   "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_CUSTOM_SNAPSHOT",
         "SnapshotName": "MyCustomSnapshot"
      }
   }
}
```
Scaling

Restart an Application Using the Most Recent Snapshot

The following example request for the `startApplication` action starts the application using the most recent snapshot:

```
{
  "ApplicationName": "MyApplication",
  "RunConfiguration": {
    "ApplicationRestoreConfiguration": {
      "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
    }
  }
}
```

Restart an Application Using No Snapshot

The following example request for the `startApplication` action starts the application without loading application state, even if a snapshot is present:

```
{
  "ApplicationName": "MyApplication",
  "RunConfiguration": {
    "ApplicationRestoreConfiguration": {
      "ApplicationRestoreType": "SKIP_RESTORE_FROM_SNAPSHOT"
    }
  }
}
```

Application Scaling in Kinesis Data Analytics for Java Applications

You can configure the parallel execution of tasks and the allocation of resources for Amazon Kinesis Data Analytics for Java Applications to implement scaling. For information about how Apache Flink schedules parallel instances of tasks, see Parallel Execution in the Apache Flink documentation.

Topics

- Configuring Application Parallelism and ParallelismPerKPU (p. 19)
- Allocating Kinesis Processing Units (p. 20)
- Automatic Scaling (p. 20)
- Updating Your Application's Parallelism (p. 21)

Configuring Application Parallelism and ParallelismPerKPU

You configure the parallel execution for your Kinesis Data Analytics for Java application tasks (such as reading from a source or executing an operator) using the following `ParallelismConfiguration` properties:
• **Parallelism** — Use this property to set the default Apache Flink application parallelism. All operators, sources, and sinks execute with this parallelism unless they are overridden in the application code. The default is 1, and the default maximum is 256.

• **ParallelismPerKPU** — Use this property to set the number of parallel tasks that can be scheduled per Kinesis Processing Unit (KPU) of your application. The default is 1, and the maximum is 8. For applications that have blocking operations (for example, I/O), a higher value of ParallelismPerKPU leads to full utilization of KPU resources.

**Note**
The limit for Parallelism is equal to ParallelismPerKPU times the limit for KPUs (which has a default of 32). The KPUs limit can be increased by requesting a limit increase. For instructions on how to request a limit increase, see "To request a limit increase" in [AWS Service Limits](https://aws.amazon.com/service-limits/).

For information about setting task parallelism for a specific operator, see Setting the Parallelism: Operator in the Apache Flink documentation.

### Allocating Kinesis Processing Units

Kinesis Data Analytics provisions capacity as KPUs. A single KPU provides you with 1 vCPU and 4 GB of memory. For every KPU allocated, 50 GB of running application storage is also provided.

Kinesis Data Analytics calculates the KPUs that are needed to run your application using the Parallelism and ParallelismPerKPU properties, as follows:

| Allocated KPUs for the application | = Parallelism/ParallelismPerKPU |

Kinesis Data Analytics quickly gives your applications resources in response to spikes in throughput or processing activity. It removes resources from your application gradually after the activity spike has passed. To disable the automatic allocation of resources, set the AutoScalingEnabled value to false, as described later in [Updating Your Application's Parallelism](#) (p. 21).

The default limit for KPUs for your application is 32. For instructions on how to request an increase to this limit, see "To request a limit increase" in [AWS Service Limits](https://aws.amazon.com/service-limits/).

### Automatic Scaling

Kinesis Data Analytics elastically scales your application's parallelism to accommodate the data throughput of your source and your operator complexity for most scenarios. Kinesis Data Analytics monitors the resource (CPU) usage of your application, and elastically scales your application's parallelism. Your application scales up (increases parallelism) is typically triggered in minutes after a constant spike in CPU usage. Your application scales down (decreases parallelism) over a few hours after a drop in CPU usage. Kinesis Data Analytics will not reduce your application's CurrentParallelism value to less than your application's Parallelism setting.

You can use the AutoScalingEnabled property (part of FlinkApplicationConfiguration) to enable or disable auto scaling behavior. Your AWS account is charged for KPUs that Kinesis Data Analytics provisions which is a function of your application's parallelism and parallelismPerKPU settings. An activity spike increases your Kinesis Data Analytics costs.

For information about pricing, see [Amazon Kinesis Data Analytics pricing](https://aws.amazon.com/kinesis/data-analytics/pricing/).

Note the following about application scaling:

• Automatic scaling is enabled by default.

• Your application has a default limit of 32 KPUs. For more information, see [Limits](#) (p. 136).
Updating Your Application's Parallelism

When autoscaling updates application parallelism, the application experiences downtime. To avoid this downtime, do the following:

• Disable automatic scaling

• Configure your application's parallelism and parallelismPerKPU with the UpdateApplication action. For more information about setting your application's parallelism settings, see the section called "Updating Your Application's Parallelism" (p. 21) following.

• Periodically monitor your application's resource usage to verify that your application has the correct parallelism settings for its workload. For information about monitoring allocation resource usage, see the section called "Metrics and Dimensions" (p. 125).

Updating Your Application's Parallelism

This section contains sample requests for API actions that set an application's parallelism. For more examples and instructions for how to use request blocks with API actions, see Kinesis Data Analytics API Example Code (p. 175).

The following example request for the CreateApplication action sets parallelism when you are creating an application:

```json
{
    "ApplicationName": "string",
    "RuntimeEnvironment": "FLINK-1_8",
    "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::mybucket",
                    "FileKey": "myflink.jar",
                    "ObjectVersion": "AbCdEfGhIjKlMnOpQrStUvWxYz12345"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "FlinkApplicationConfiguration": {
            "ParallelismConfiguration": {
                "AutoScalingEnabled": "true",
                "ConfigurationType": "CUSTOM",
                "Parallelism": 4,
                "ParallelismPerKPU": 4
            }
        }
    }
}
```

The following example request for the UpdateApplication action sets parallelism for an existing application:

```json
{
    "ApplicationName": "MyApplication",
    "ApplicationConfigurationUpdate": {
        "FlinkApplicationConfigurationUpdate": {
            "ParallelismConfigurationUpdate": {
                "AutoScalingEnabledUpdate": "true",
                "ConfigurationTypeUpdate": "CUSTOM",
                "ParallelismUpdate": 4,
                "ParallelismPerKPUUpdate": 4
            }
        }
    }
}
```
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 AWS Identity and Access Management 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 (p. 22)
- Adding or Updating Tags for an Existing Application (p. 23)
- Listing Tags for an Application (p. 23)
- Removing Tags from an Application (p. 23)

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:

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

```
{
}
```

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:

```
{
    "TagKeys": [ "KeyOfFirstTagToRemove", "KeyOfSecondTagToRemove" ]
}
```
Getting Started with Amazon Kinesis Data Analytics for Java Applications

This section introduces you to the fundamental concepts of Amazon Kinesis Data Analytics for Java Applications. It describes the available options for creating and testing your applications. It also provides instructions for installing the necessary tools to complete the tutorials in this guide and to create your first application.

Topics
- Components of a Kinesis Data Analytics for Java Application (p. 24)
- Prerequisites for Completing the Exercises (p. 24)
- Step 1: Set Up an AWS Account and Create an Administrator User (p. 25)
- Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 27)
- Step 3: Create and Run a Kinesis Data Analytics for Java Application (p. 28)
- Step 4: Clean Up AWS Resources (p. 42)

Components of a Kinesis Data Analytics for Java Application

To process data, your Kinesis Data Analytics application uses a Java/Apache Maven application that processes input and produces output using the Apache Flink runtime.

A Kinesis Data Analytics for Java application has the following components:

- **Runtime properties**: You can use runtime properties to configure your application without recompiling your application code.
- **Source**: The application consumes data by using a source. A source connector reads data from a Kinesis data stream, an Amazon S3 bucket, etc. For more information, see Sources (p. 6).
- **Operators**: The application processes data by using one or more operators. An operator can transform, enrich, or aggregate data. For more information, see Operators (p. 9).
- **Sink**: The application produces data to external sources by using sinks. A sink connector writes data to a Kinesis data stream, a Kinesis Data Firehose delivery stream, an Amazon S3 bucket, etc. For more information, see Sinks (p. 6).

After you create, compile, and package your Java application, you upload the code package to an Amazon Simple Storage Service (Amazon S3) bucket. You then create a Kinesis Data Analytics application. You pass in the code package location, a Kinesis data stream as the streaming data source, and typically a streaming or file location that receives the application's processed data.

Prerequisites for Completing the Exercises

To complete the steps in this guide, you must have the following:

- **Java Development Kit (JDK)** version 8. Set the JAVA_HOME environment variable to point to your JDK install location.
• We recommend that you use a development environment (such as Eclipse Java Neon or IntelliJ Idea) to develop and compile your application.
• Git Client. Install the Git client if you haven't already.
• Apache Maven Compiler Plugin. Maven must be in your working path. To test your Apache Maven installation, enter the following:

```bash
$ mvn -version
```

Note
Kinesis Data Analytics for Java Applications only supports Java applications that are built with Apache Maven.

To get started, go to Step 1: Set Up an AWS Account and Create an Administrator User (p. 25).

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

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

1. Sign Up for AWS (p. 25)
2. Create an IAM User (p. 25)

**Sign Up for AWS**

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

With Kinesis Data Analytics, you pay only for the resources that 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 Tier.

If you already have an AWS account, skip to the next task. If you don't have an AWS account, follow these steps to create one.

**To create an AWS account**

2. Follow the online instructions.
   
   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

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

**Create an IAM User**

Services in AWS, such as Amazon Kinesis Data Analytics, require that you provide credentials when you access them. This is so that the service can determine whether you have permissions to access the resources that are owned by that service. The AWS Management Console requires that you enter your password.
You can create access keys for your AWS account to access the AWS Command Line Interface (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 permissions. Follow the procedure to create adminuser in your account.

**To create a group for administrators**

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Groups, and then choose Create New Group.
3. For Group Name, enter a name for your group, such as Administrators, and then choose Next Step.
4. In the list of policies, select the check box next to the AdministratorAccess policy. You can use the Filter menu and the Search box to filter the list of policies.
5. Choose Next Step, and then choose Create Group.

Your new group is listed under Group Name.

**To create an IAM user for yourself, add it to the Administrators group, and create a password**

1. In the navigation pane, choose Users, and then choose Add user.
2. In the User name box, enter a user name.
3. Choose both Programmatic access and AWS Management Console access.
4. Choose Next: Permissions.
5. Select the check box next to the Administrators group. Then choose Next: Review.
6. Choose Create user.

**To sign in as the new IAM user**

1. Sign out of the AWS Management Console.
2. Use the following URL format to sign in to the console:

   https://aws_account_number.signin.aws.amazon.com/console/

   The aws_account_number is your AWS account ID without any hyphens. For example, if your AWS account ID is 1234-5678-9012, replace aws_account_number with 123456789012. For information about how to find your account number, see Your AWS Account ID and Its Alias in the IAM User Guide.
3. Enter the IAM user name and password that you just created. When you’re signed in, the navigation bar displays your_user_name @ your_aws_account_id.

   **Note**
   If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias.
To create or remove an account alias

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. On the navigation pane, choose Dashboard.
3. Find the IAM users sign-in link.
4. To create the alias, choose Customize. Enter the name you want to use for your alias, and then choose Yes, Create.
5. To remove the alias, choose Customize, and then choose Yes, Delete. The sign-in URL reverts to using your AWS account ID.

To sign in after you create an account alias, use the following URL:

https://your_account_alias.signin.aws.amazon.com/console/

To verify the sign-in link for IAM users for your account, open the IAM console and check under IAM users sign-in link on the dashboard.

For more information about IAM, see the following:

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

Next Step

Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 27)

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

In this step, you download and configure the AWS CLI to use with Amazon Kinesis Data Analytics for Java Applications.

Note

The getting started exercises in this guide assume that you are using administrator credentials (adminuser) in your account to perform the operations.

Note

If you already have the AWS CLI installed, you might need to upgrade to get the latest functionality. For more information, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide. To check the version of the AWS CLI, run the following command:

aws --version

The exercises in this tutorial require the following AWS CLI version or later:

aws-cli/1.16.63

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:
Installing the AWS Command Line Interface
Configuring the AWS CLI

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 AWS Regions and Endpoints in the Amazon Web Services General Reference.

**Note**
The example code and commands in this tutorial use the US West (Oregon) Region. To use a different AWS Region, change the Region in the code and commands for this tutorial to the Region you want to use.

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

```
aws help
```

After you set up an AWS account and the AWS CLI, you can try the next exercise, in which you configure a sample application and test the end-to-end setup.

**Next Step**

Step 3: Create and Run a Kinesis Data Analytics for Java Application (p. 28)

**Step 3: Create and Run a Kinesis Data Analytics for Java Application**

In this exercise, you create a Kinesis Data Analytics for Java application with data streams as a source and a sink.

This section contains the following steps:

- Create Two Amazon Kinesis Data Streams (p. 28)
- Write Sample Records to the Input Stream (p. 29)
- Download and Examine the Apache Flink Streaming Java Code (p. 30)
- Compile the Application Code (p. 30)
- Upload the Apache Flink Streaming Java Code (p. 31)
- Create and Run the Kinesis Data Analytics Application (p. 31)

**Create Two Amazon Kinesis Data Streams**

Before you create a Kinesis Data Analytics for Java application for this exercise, create two Kinesis data streams (ExampleInputStream and ExampleOutputStream). Your application uses these streams for the application source and destination streams.
You can create these streams using either the Amazon Kinesis console or the following AWS CLI command. For console instructions, see Creating and Updating Data Streams in the Amazon Kinesis Data Streams Developer Guide.

To create the data streams (AWS CLI)

1. To create the first stream (ExampleInputStream), use the following Amazon Kinesis create-stream AWS CLI command.

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. To create the second stream that the application uses to write output, run the same command, changing the stream name to ExampleOutputStream.

```
$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

Write Sample Records to the Input Stream

In this section, you use a Python script to write sample records to the stream for the application to process.

Note
- This section requires the AWS SDK for Python (Boto).

1. Create a file named `stock.py` with the following contents:

```python
import json
import boto3
import random
import datetime
kinesis = boto3.client('kinesis')

def getReferrer():
    data = {}
    now = datetime.datetime.now()
    str_now = now.isoformat()
    data['EVENT_TIME'] = str_now
    data['TICKER'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
    price = random.random() * 100
    data['PRICE'] = round(price, 2)
    return data

while True:
    data = json.dumps(getReferrer())
    print(data)
    kinesis.put_record(
        StreamName="ExampleInputStream",
        Data=data,
        PartitionKey="partitionkey")
```
Download and Examine the Apache Flink Streaming Java Code

The Java application code for this example is available from GitHub. To download the application code, do the following:

1. Clone the remote repository using the following command:

   ```
   git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-examples
   ```


Note the following about the application code:

- A Project Object Model (pom.xml) file contains information about the application's configuration and dependencies, including the Kinesis Data Analytics for Java Applications libraries.
- The `BasicStreamingJob.java` file contains the main method that defines the application's functionality.
- The application uses a Kinesis source to read from the source stream. The following snippet creates the Kinesis source:

  ```java
  return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
       new SimpleStringSchema(), inputProperties));
  ```

- Your application creates source and sink connectors to access external resources using a `StreamExecutionEnvironment` object.
- The application creates source and sink connectors using static properties. To use dynamic application properties, use the `createSourceFromApplicationProperties` and `createSinkFromApplicationProperties` methods to create the connectors. These methods read the application's properties to configure the connectors.

For more information about runtime properties, see Runtime Properties (p. 10).

Compile the Application Code

In this section, you use the Apache Maven compiler to create the Java code for the application. For information about installing Apache Maven and the Java Development Kit (JDK), see Prerequisites for Completing the Exercises (p. 24).

**Note**

In order to use the Kinesis connector for the following application, you need to download, build, and install Apache Maven. For more information, see the section called “Using the Apache Flink Kinesis Streams Connector” (p. 3).

To compile the application code

1. To use your application code, you compile and package it into a JAR file. You can compile and package your code in one of two ways:
Upload the Apache Flink Streaming Java Code

In this section, you create an Amazon Simple Storage Service (Amazon S3) bucket and upload your application code.

To upload the application code

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose Create bucket.
3. Enter ka-app-code-<username> in the Bucket name field. Add a suffix to the bucket name, such as your user name, to make it globally unique. Choose Next.
4. In the Configure options step, keep the settings as they are, and choose Next.
5. In the Set permissions step, keep the settings as they are, and choose Next.
6. Choose Create bucket.
7. In the Amazon S3 console, choose the ka-app-code-<username> bucket, and choose Upload.
8. In the Select files step, choose Add files. Navigate to the aws-kinesis-analytics-java-apps-1.0.jar file that you created in the previous step. Choose Next.
9. You don't need to change any of the settings for the object, so choose Upload.

Your application code is now stored in an Amazon S3 bucket where your application can access it.

Create and Run the Kinesis Data Analytics Application

You can create and run a Kinesis Data Analytics for Java application using either the console or the AWS CLI.

Note
When you create the application using the console, your AWS Identity and Access Management (IAM) and Amazon CloudWatch Logs resources are created for you. When you create the application using the AWS CLI, you create these resources separately.
Topics

- Create and Run the Application (Console) (p. 32)
- Create and Run the Application (AWS CLI) (p. 37)

Create and Run the Application (Console)

Follow these steps to create, configure, update, and run the application using the console.

Create the Application

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. On the Amazon Kinesis Data Analytics dashboard, choose Create analytics application.
3. On the Kinesis Analytics - Create application page, provide the application details as follows:
   - For Application name, enter MyApplication.
   - For Description, enter My java test app.
   - For Runtime, choose Apache Flink.
   - Leave the version pulldown as Apache Flink 1.8 (Recommended Version).
5. Choose Create application.

**Note**
When you create a Kinesis Data Analytics for Java application using the console, you have the option of having an IAM role and policy created for your application. Your application uses this role and policy to access its dependent resources. These IAM resources are named using your application name and Region as follows:
• Policy: kinesis-analytics-service-MyApplication-us-west-2
• Role: kinesis-analytics-MyApplication-us-west-2

Edit the IAM Policy

Edit the IAM policy to add permissions to access the Kinesis data streams.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Policies. Choose the kinesis-analytics-service-MyApplication-us-west-2 policy that the console created for you in the previous section.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (012345678901) with your account ID.

```json
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Sid": "ReadCode",
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:GetObjectVersion"
        ],
        "Resource": ["arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-apps-1.0.jar"
        ]
    },
    {
        "Sid": "DescribeLogGroups",
        "Effect": "Allow",
        "Action": ["logs:DescribeLogGroups"],
        ]
    },
    {
        "Sid": "DescribeLogStreams",
        "Effect": "Allow",
        "Action": ["logs:DescribeLogStreams"],
        ]
    },
    {
        "Sid": "PutLogEvents",
        "Effect": "Allow",
        "Action": ["logs:PutLogEvents"],
        ]
    }
},
```
Configure the Application

1. On the MyApplication page, choose Configure.

2. On the Configure application page, provide the Code location:
   - For Amazon S3 bucket, enter ka-app-code-<username>.
   - For Path to Amazon S3 object, enter aws-kinesis-analytics-java-apps-1.0.jar.


4. Under Properties, for Group ID, enter ProducerConfigProperties.

5. Enter the following application properties and values:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>flink.inputstream.initpos</td>
<td>LATEST</td>
</tr>
<tr>
<td>aws.region</td>
<td>us-west-2</td>
</tr>
<tr>
<td>AggregationEnabled</td>
<td>false</td>
</tr>
</tbody>
</table>

6. Under Monitoring, ensure that the Monitoring metrics level is set to Application.

7. For CloudWatch logging, select the Enable check box.

8. Choose Update.
Create and Run the Kinesis Data Analytics Application

Note
When you choose to enable Amazon CloudWatch logging, Kinesis Data Analytics creates a log group and log stream for you. The names of these resources are as follows:

- Log group: /aws/kinesis-analytics/MyApplication
- Log stream: kinesis-analytics-log-stream

Run the Application


2. When the application is running, refresh the page. The console shows the Application graph.
Stop the Application

On the **MyApplication** page, choose **Stop**. Confirm the action.

Update the Application

Using the console, you can update application settings such as application properties, monitoring settings, and the location or file name of the application JAR. You can also reload the application JAR from the Amazon S3 bucket if you need to update the application code.

On the **MyApplication** page, choose **Configure**. Update the application settings and choose **Update**.

Create and Run the Application (AWS CLI)

In this section, you use the AWS CLI to create and run the Kinesis Data Analytics application. Kinesis Data Analytics for Java Applications uses the `kinesisanalyticsv2` AWS CLI command to create and interact with Kinesis Data Analytics applications.

Create a Permissions Policy

**Note**

You must create a permissions policy and role for your application. If you do not create these IAM resources, your application cannot access its data and log streams.

First, you create a permissions policy with two statements: one that grants permissions for the *read* action on the source stream, and another that grants permissions for *write* actions on the sink stream. You then attach the policy to an IAM role (which you create in the next section). Thus, when Kinesis Data Analytics assumes the role, the service has the necessary permissions to read from the source stream and write to the sink stream.

Use the following code to create the **KAReadSourceStreamWriteSinkStream** permissions policy. Replace `username` with the user name that you used to create the Amazon S3 bucket to store the application code. Replace the account ID in the Amazon Resource Names (ARNs) (**012345678901**) with your account ID.

```
For step-by-step instructions to create a permissions policy, see Tutorial: Create and Attach Your First Customer Managed Policy in the IAM User Guide.

Note
To access other AWS services, you can use the AWS SDK for Java. Kinesis Data Analytics automatically sets the credentials required by the SDK to those of the service execution IAM role that is associated with your application. No additional steps are needed.

Create an IAM Role

In this section, you create an IAM role that the Kinesis Data Analytics for Java application can assume to read a source stream and write to the sink stream.

Kinesis Data Analytics cannot access your stream without permissions. You grant these permissions via an IAM role. Each IAM role has two policies attached. The trust policy grants Kinesis Data Analytics permission to assume the role, and the permissions policy determines what Kinesis Data Analytics can do after assuming the role.

You attach the permissions policy that you created in the preceding section to this role.

To create an IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles, Create Role.
3. Under Select type of trusted identity, choose AWS Service. Under Choose the service that will use this role, choose Kinesis. Under Select your use case, choose Kinesis Analytics.
4. Choose Next: Permissions.
5. On the Attach permissions policies page, choose Next: Review. You attach permissions policies after you create the role.
6. On the Create role page, enter KA-stream-rw-role for the Role name. Choose Create role.

Now you have created a new IAM role called KA-stream-rw-role. Next, you update the trust and permissions policies for the role.
6. Attach the permissions policy to the role.

**Note**
For this exercise, Kinesis Data Analytics assumes this role for both reading data from a Kinesis data stream (source) and writing output to another Kinesis data stream. So you attach the policy that you created in the previous step, the section called “Create a Permissions Policy” (p. 37).

a. On the **Summary** page, choose the **Permissions** tab.
b. Choose **Attach Policies**.
c. In the search box, enter **KAReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
d. Choose the **KAReadInputStreamWriteOutputStream** policy, and choose **Attach policy**.

You now have created the service execution role that your application uses to access resources. Make a note of the ARN of the new role.

For step-by-step instructions for creating a role, see **Creating an IAM Role (Console)** in the **IAM User Guide**.

**Create the Kinesis Data Analytics Application**

1. Save the following JSON code to a file named `create_request.json`. Replace the sample role ARN with the ARN for the role that you created previously. Replace the bucket ARN suffix (**username**) with the suffix that you chose in the previous section. Replace the sample account ID (**012345678901**) in the service execution role with your account ID.

```json
{
  "ApplicationName": "test",
  "ApplicationDescription": "my java test app",
  "RuntimeEnvironment": "FLINK-1_8",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/KA-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "aws-kinesis-analytics-java-apps-1.0.jar"
        }
      },
      "CodeContentType": "ZIPFILE"
    },
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap": {
            "flink.stream.initpos": "LATEST",
            "aws.region": "us-west-2",
            "AggregationEnabled": "false"
          }
        },
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-west-2"
          }
        }
      ]
    }
  }
}
```
2. Execute the `CreateApplication` action with the preceding request to create the application:

```bash
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

The application is now created. You start the application in the next step.

**Start the Application**

In this section, you use the `StartApplication` action to start the application.

**To start the application**

1. Save the following JSON code to a file named `start_request.json`.

```json
{
   "ApplicationName": "test",
   "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
      }
   }
}
```

2. Execute the `StartApplication` action with the preceding request to start the application:

```bash
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

The application is now running. You can check the Kinesis Data Analytics metrics on the Amazon CloudWatch console to verify that the application is working.

**Stop the Application**

In this section, you use the `StopApplication` action to stop the application.

**To stop the application**

1. Save the following JSON code to a file named `stop_request.json`.

```json
{"ApplicationName": "test"}
```

2. Execute the `StopApplication` action with the following request to stop the application:

```bash
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

The application is now stopped.

**Add a CloudWatch Logging Option**

You can use the AWS CLI to add an Amazon CloudWatch log stream to your application. For information about using CloudWatch Logs with your application, see the section called “Setting Up Logging” (p. 116).
Update Environment Properties

In this section, you use the `UpdateApplication` action to change the environment properties for the application without recompiling the application code. In this example, you change the AWS Region of the source and destination streams.

To update environment properties for the application

1. Save the following JSON code to a file named `update_properties_request.json`.

```json
{"ApplicationName": "test",
 "CurrentApplicationVersionId": 1,
 "ApplicationConfigurationUpdate": {
  "EnvironmentPropertyUpdates": [
   "PropertyGroups": [
    {"PropertyGroupId": "ProducerConfigProperties",
     "PropertyMap": {
      "flink.stream.initpos": "LATEST",
      "aws.region": "us-west-2",
      "AggregationEnabled": "false"
     }
    },
    {"PropertyGroupId": "ConsumerConfigProperties",
     "PropertyMap": {
      "aws.region": "us-west-2"
     }
    }
   ]
  }
 }
}
```

2. Execute the `UpdateApplication` action with the preceding request to update environment properties:

```
aws kinesisanalyticsv2 update-application --cli-input-json file://update_properties_request.json
```

Update the Application Code

When you need to update your application code with a new version of your code package, you use the `UpdateApplication` AWS CLI action.

**Note**

To load a new version of the application code with the same filename, you must specify the new object version. For more information about using Amazon S3 object versions, see [Enabling or Disabling Versioning](#).

To use the AWS CLI, delete your previous code package from your Amazon S3 bucket, upload the new version, and call `UpdateApplication`, specifying the same Amazon S3 bucket and object name, and the new object version. The application will restart with the new code package.

The following sample request for the `UpdateApplication` action reloads the application code and restarts the application. Update the `CurrentApplicationVersionId` to the current application version. You can check the current application version using the `ListApplications` or `DescribeApplication` actions. Update the bucket name suffix (`<username>`) with the suffix that you chose in the section called “Create Dependent Resources” (p. 80) section.
Step 4: Clean Up AWS Resources

This section includes procedures for cleaning up AWS resources created in the Getting Started tutorial.

This topic contains the following sections:
- Delete Your Kinesis Data Analytics Application (p. 42)
- Delete Your Kinesis Data Streams (p. 42)
- Delete Your Amazon S3 Object and Bucket (p. 42)
- Delete Your IAM Resources (p. 43)
- Delete Your CloudWatch Resources (p. 43)

Delete Your Kinesis Data Analytics Application

2. In the Kinesis Data Analytics panel, choose MyApplication.
3. Choose Configure.
4. In the Snapshots section, choose Disable and then choose Update.
5. In the application’s page, choose Delete and then confirm the deletion.

Delete Your Kinesis Data Streams

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. In the Kinesis Data Streams panel, choose ExampleInputStream.
3. In the ExampleInputStream page, choose Delete Kinesis Stream and then confirm the deletion.
4. In the Kinesis streams page, choose the ExampleOutputStream, choose Actions, choose Delete, and then confirm the deletion.

Delete Your Amazon S3 Object and Bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the ka-app-code-<username> bucket.
3. Choose Delete and then enter the bucket name to confirm deletion.
Delete Your IAM Resources

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation bar, choose Policies.
3. In the filter control, enter kinesis.
4. Choose the kinesis-analytics-service-MyApplication-<your-region> policy.
5. Choose Policy Actions and then choose Delete.
6. In the navigation bar, choose Roles.
7. Choose the kinesis-analytics-MyApplication-<your-region> role.
8. Choose Delete role and then confirm the deletion.

Delete Your CloudWatch Resources

2. In the navigation bar, choose Logs.
3. Choose the /aws/kinesis-analytics/MyApplication log group.
4. Choose Delete Log Group and then confirm the deletion.
Kinesis Data Analytics for Java Applications: 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 the following:

- How it Works (p. 2)
- Getting Started (p. 24)

Note
These examples assume that you are using the US West (Oregon) Region (us-west-2). If you are using a different Region, update your application code, commands, and IAM roles appropriately.

Topics
- Example: Tumbling Window (p. 44)
- Example: Sliding Window (p. 53)
- Example: Writing to S3 (p. 62)
- Tutorial: Using a Kinesis Data Analytics for Java application to Replicate Data from One MSK Cluster to Another in a VPC (p. 71)
- Example: Writing to Kinesis Data Firehose (p. 79)
- Example: Using a Custom Sink to Write to CloudWatch Logs (p. 92)
- Example: Read From a Kinesis Stream in a Different Account (p. 103)

Example: Tumbling Window

In this exercise, you create a Kinesis Data Analytics for Java application that aggregates data using a tumbling window.

Note
To set up required prerequisites for this exercise, first complete the Getting Started (p. 24) exercise.

This topic contains the following sections:
- Create Dependent Resources (p. 45)
- Write Sample Records to the Input Stream (p. 45)
- Download and Examine the Application Code (p. 46)
- Compile the Application Code (p. 46)
- Upload the Apache Flink Streaming Java Code (p. 47)
Create Dependent Resources

Before you create a Kinesis Data Analytics for Java application for this exercise, you create the following dependent resources:

- Two Kinesis data streams (ExampleInputStream and ExampleOutputStream)
- An Amazon S3 bucket to store the application's code (ka-app-code-<username>)

You can create the Kinesis streams and Amazon S3 bucket using the console. For instructions for creating these resources, see the following topics:

- Creating and Updating Data Streams in the Amazon Kinesis Data Streams Developer Guide. Name your data stream ExampleInputStream and ExampleOutputStream.
- How Do I Create an S3 Bucket? in the Amazon Simple Storage Service Developer Guide. Give the Amazon S3 bucket a globally unique name by appending your login name, such as ka-app-code-<username>.

Write Sample Records to the Input Stream

In this section, you use a Python script to write sample records to the stream for the application to process.

**Note**

This section requires the AWS SDK for Python (Boto).

1. Create a file named `stock.py` with the following contents:

   ```python
   import json
   import boto3
   import random
   import datetime

   kinesis = boto3.client('kinesis')
   def getReferrer():
       data = {}
       now = datetime.datetime.now()
       str_now = now.isoformat()
       data['EVENT_TIME'] = str_now
       data['TICKER'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
       price = random.random() * 100
       data['PRICE'] = round(price, 2)
       return data

   while True:
       data = json.dumps(getReferrer())
       print(data)
       kinesis.put_record(
           StreamName="ExampleInputStream",
           Data=data,
           PartitionKey="partitionkey")
   ```

2. Run the `stock.py` script:
Keep the script running while completing the rest of the tutorial.

**Download and Examine the Application Code**

The Java application code for this example is available from GitHub. To download the application code, do the following:

1. Install the Git client if you haven't already. For more information, see [Installing Git](#).
2. Clone the remote repository with the following command:
   ```
   git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-examples
   ```

   The application code is located in the `TumblingWindowStreamingJob.java` file. Note the following about the application code:

   - The application uses a Kinesis source to read from the source stream. The following snippet creates the Kinesis source:
     ```
     return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
       new SimpleStringSchema(), inputProperties));
     ```
   - The application uses the `timeWindow` operator to find the count of values for each stock symbol over a 5-second tumbling window. The following code creates the operator and sends the aggregated data to a new Kinesis Data Streams sink:
     ```
     input.flatMap(new Tokenizer()) // Tokenizer for generating words
       .keyBy(0) // Logically partition the stream for each word
       .timeWindow(Time.seconds(5)) // Tumbling window definition
       .sum(1) // Sum the number of words per partition
       .map(value -> value.f0 + "," + value.f1.toString() + "\n")
       .addSink(createSinkFromStaticConfig());
     ```

**Compile the Application Code**

To compile the application, do the following:

1. Install Java and Maven if you haven't already. For more information, see [Prerequisites](#) in the Getting Started tutorial.
2. In order to use the Kinesis connector for the following application, you need to download, build, and install Apache Maven. For more information, see the section called "Using the Apache Flink Kinesis Streams Connector" (p. 3).
3. Compile the application with the following command:
   ```
   mvn package -Dflink.version=1.8.2
   ```
Note
The provided source code relies on libraries from Java 1.8. If you are using a development environment, ensure that your project’s Java version is 1.8.

Compiling the application creates the application JAR file (target/aws-kinesis-analytics-java-apps-1.0.jar).

Upload the Apache Flink Streaming Java Code

In this section, you upload your application code to the Amazon S3 bucket you created in the Create Dependent Resources (p. 45) section.

1. In the Amazon S3 console, choose the ka-app-code-<username> bucket, and choose Upload.
2. In the Select files step, choose Add files. Navigate to the aws-kinesis-analytics-java-apps-1.0.jar file that you created in the previous step.
3. You don’t need to change any of the settings for the object, so choose Upload.

Your application code is now stored in an Amazon S3 bucket where your application can access it.

Create and Run the Kinesis Data Analytics Application

Follow these steps to create, configure, update, and run the application using the console.

Create the Application

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. On the Amazon Kinesis Data Analytics dashboard, choose Create analytics application.
3. On the Kinesis Analytics - Create application page, provide the application details as follows:
   - For Application name, enter MyApplication.
   - For Runtime, choose Apache Flink.

   Note
   Kinesis Data Analytics for Java Applications uses Apache Flink version 1.8.2.
   - Leave the version pulldown as Apache Flink 1.8 (Recommended Version).
Choose **Create application**.

**Note**
When you create a Kinesis Data Analytics for Java application using the console, you have the option of having an IAM role and policy created for your application. Your application uses this role and policy to access its dependent resources. These IAM resources are named using your application name and Region as follows:
• Policy: kinesis-analytics-service-MyApplication-us-west-2
• Role: kinesis-analytics-MyApplication-us-west-2

Edit the IAM Policy

Edit the IAM policy to add permissions to access the Kinesis data streams.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Policies. Choose the kinesis-analytics-service-MyApplication-us-west-2 policy that the console created for you in the previous section.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (012345678901) with your account ID.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": ["logs:DescribeLogGroups"
            ],
            "Resource": [
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": ["kinesis:*"],
        }
    ],
    "Condition": {}
}
```
Configure the Application

1. On the **MyApplication** page, choose **Configure**.

2. On the **Configure application** page, provide the **Code location**:
   - For **Amazon S3 bucket**, enter `ka-app-code-<username>`.
   - For **Path to Amazon S3 object**, enter `aws-kinesis-analytics-java-apps-1.0.jar`.
4. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
5. For **CloudWatch logging**, select the **Enable** check box.
6. Choose **Update**.
Note
When you choose to enable CloudWatch logging, Kinesis Data Analytics creates a log group and log stream for you. The names of these resources are as follows:

- Log group: /aws/kinesis-analytics/MyApplication
- Log stream: kinesis-analytics-log-stream

This log stream is used to monitor the application. This is not the same log stream that the application uses to send results.

Run the Application

2. When the application is running, refresh the page. The console shows the **Application graph**.

![Application graph](image)

You can check the Kinesis Data Analytics metrics on the CloudWatch console to verify that the application is working.

### Clean Up AWS Resources

This section includes procedures for cleaning up AWS resources created in the Tumbling Window tutorial.

**This topic contains the following sections:**
- Delete Your Kinesis Data Analytics Application (p. 52)
- Delete Your Kinesis Data Streams (p. 52)
- Delete Your Amazon S3 Object and Bucket (p. 52)
- Delete Your IAM Resources (p. 53)
- Delete Your CloudWatch Resources (p. 53)

#### Delete Your Kinesis Data Analytics Application

2. In the Kinesis Data Analytics panel, choose **MyApplication**.
3. Choose **Configure**.
4. In the **Snapshots** section, choose **Disable** and then choose **Update**.
5. In the application's page, choose **Delete** and then confirm the deletion.

#### Delete Your Kinesis Data Streams

2. In the Kinesis Data Streams panel, choose **ExampleInputStream**.
3. In the **ExampleInputStream** page, choose **Delete Kinesis Stream** and then confirm the deletion.
4. In the **Kinesis streams** page, choose the **ExampleOutputStream**, choose **Actions**, choose **Delete**, and then confirm the deletion.

#### Delete Your Amazon S3 Object and Bucket

1. Open the Amazon S3 console at [https://console.aws.amazon.com/s3/](https://console.aws.amazon.com/s3/).
2. Choose the **ka-app-code-<username>** bucket.
3. Choose **Delete** and then enter the bucket name to confirm deletion.

### Delete Your IAM Resources

2. In the navigation bar, choose **Policies**.
3. In the filter control, enter `kinesis`.
4. Choose the `kinesis-analytics-service-MyApplication-<your-region>` policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the `kinesis-analytics-MyApplication-<your-region>` role.
8. Choose **Delete role** and then confirm the deletion.

### Delete Your CloudWatch Resources

2. In the navigation bar, choose **Logs**.
3. Choose the `/aws/kinesis-analytics/MyApplication` log group.
4. Choose **Delete Log Group** and then confirm the deletion.

### Example: Sliding Window

In this exercise, you create a Kinesis Data Analytics for Java application that aggregates data using a sliding window.

**Note**

To set up required prerequisites for this exercise, first complete the Getting Started (p. 24) exercise.

This topic contains the following sections:

- Create Dependent Resources (p. 53)
- Write Sample Records to the Input Stream (p. 54)
- Download and Examine the Application Code (p. 54)
- Compile the Application Code (p. 55)
- Upload the Apache Flink Streaming Java Code (p. 55)
- Create and Run the Kinesis Data Analytics Application (p. 56)
- Clean Up AWS Resources (p. 61)

### Create Dependent Resources

Before you create a Kinesis Data Analytics for Java application for this exercise, you create the following dependent resources:

- Two Kinesis data streams (`ExampleInputStream` and `ExampleOutputStream`).
- An Amazon S3 bucket to store the application's code (`ka-app-code-<username>`)

You can create the Kinesis streams and Amazon S3 bucket using the console. For instructions for creating these resources, see the following topics:
Write Sample Records to the Input Stream

In this section, you use a Python script to write sample records to the stream for the application to process.

Note
This section requires the AWS SDK for Python (Boto).

1. Create a file named stock.py with the following contents:

```python
import json
import boto3
import random
import datetime

kinesis = boto3.client('kinesis')
def getReferrer():
    data = {}
    now = datetime.datetime.now()
    str_now = now.isoformat()
    data['EVENT_TIME'] = str_now
    data['TICKER'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
    price = random.random() * 100
    data['PRICE'] = round(price, 2)
    return data

while True:
    data = json.dumps(getReferrer())
    print(data)
    kinesis.put_record(
        StreamName="ExampleInputStream",
        Data=data,
        PartitionKey="partitionkey")
```

2. Run the stock.py script:

```bash
$ python stock.py
```

Keep the script running while completing the rest of the tutorial.

Download and Examine the Application Code

The Java application code for this example is available from GitHub. To download the application code, do the following:

1. Install the Git client if you haven't already. For more information, see Installing Git.
2. Clone the remote repository with the following command:

```bash
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-examples
```
Compile the Application Code

To compile the application, do the following:

1. Install Java and Maven if you haven't already. For more information, see Prerequisites (p. 24) in the Getting Started (p. 24) tutorial.

2. In order to use the Kinesis connector for the following application, you need to download, build, and install Apache Maven. For more information, see the section called “Using the Apache Flink Kinesis Streams Connector” (p. 3).

3. Compile the application with the following command:

   ```sh
   mvn package -Dflink.version=1.8.2
   ```

   **Note**

   The provided source code relies on libraries from Java 1.8. If you are using a development environment, ensure that your project’s Java version is 1.8.

   Compiling the application creates the application JAR file (`target/aws-kinesis-analytics-java-apps-1.0.jar`).

Upload the Apache Flink Streaming Java Code

In this section, you upload your application code to the Amazon S3 bucket that you created in the Create Dependent Resources (p. 53) section.
1. In the Amazon S3 console, choose the ka-app-code-<username> bucket, and then choose **Upload**.

2. In the Select files step, choose **Add files**. Navigate to the aws-kinesis-analytics-java-apps-1.0.jar file that you created in the previous step.

3. You don't need to change any of the settings for the object, so choose **Upload**.

Your application code is now stored in an Amazon S3 bucket where your application can access it.

**Create and Run the Kinesis Data Analytics Application**

Follow these steps to create, configure, update, and run the application using the console.

**Create the Application**


2. On the Amazon Kinesis Data Analytics dashboard, choose **Create analytics application**.

3. On the Kinesis Analytics - Create application page, provide the application details as follows:
   - For **Application name**, enter *MyApplication*.
   - For **Runtime**, choose *Apache Flink*.
   - Leave the version pulldown as *Apache Flink 1.8 (Recommended Version)*.

4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
5. Choose Create application.

**Note**
When you create a Kinesis Data Analytics for Java application using the console, you have the option of having an IAM role and policy created for your application. Your application uses this role and policy to access its dependent resources. These IAM resources are named using your application name and Region as follows:
Edit the IAM Policy

Edit the IAM policy to add permissions to access the Kinesis data streams.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Policies. Choose the kinesis-analytics-service-MyApplication-us-west-2 policy that the console created for you in the previous section.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (012345678901) with your account ID.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "ReadCode",
         "Effect": "Allow",
         "Action": [
            "s3:GetObject",
            "logs:DescribeLogGroups",
            "s3:GetObjectVersion"
         ],
         "Resource": [
            "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-apps-1.0.jar"
         ],
      },
      {
         "Sid": "DescribeLogStreams",
         "Effect": "Allow",
         "Action": "logs:DescribeLogStreams",
      },
      {
         "Sid": "PutLogEvents",
         "Effect": "Allow",
         "Action": "logs:PutLogEvents",
      },
      {
         "Sid": "ListCloudwatchLogGroups",
         "Effect": "Allow",
         "Action": ["logs:DescribeLogGroups"
         ],
         ]
      },
      {
         "Sid": "ReadInputStream",
         "Effect": "Allow",
         "Action": "kinesis:*",
      }
   ]
}
```
Configure the Application

1. On the **MyApplication** page, choose **Configure**.

2. On the **Configure application** page, provide the **Code location**:
   - For **Amazon S3 bucket**, enter `ka-app-code-<username>`.
   - For **Path to Amazon S3 object**, enter `aws-kinesis-analytics-java-apps-1.0.jar`.

3. Under **Access to application resources**, for **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.

4. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.

5. For **CloudWatch logging**, select the **Enable** check box.

6. Choose **Update**.
Note
When you choose to enable Amazon CloudWatch logging, Kinesis Data Analytics creates a log group and log stream for you. The names of these resources are as follows:

- Log group: /aws/kinesis-analytics/MyApplication
- Log stream: kinesis-analytics-log-stream

This log stream is used to monitor the application. This is not the same log stream that the application uses to send results.

Configure the Application Parallelism

This application example uses parallel execution of tasks. The following application code sets the parallelism of the `min` operator:

```
.setParallelism(3) // Set parallelism for the min operator
```

The application parallelism can't be greater than the provisioned parallelism, which has a default of 1. To increase your application's parallelism, use the following AWS CLI action:

```
aws kinesisanalyticsv2 update-application
  --application-name MyApplication
```
Run the Application


2. When the application is running, refresh the page. The console shows the Application graph.

You can check the Kinesis Data Analytics metrics on the CloudWatch console to verify that the application is working.

Clean Up AWS Resources

This section includes procedures for cleaning up AWS resources created in the Sliding Window tutorial.

This topic contains the following sections:
- Delete Your Kinesis Data Analytics Application (p. 61)
- Delete Your Kinesis Data Streams (p. 62)
- Delete Your Amazon S3 Object and Bucket (p. 62)
- Delete Your IAM Resources (p. 62)
- Delete Your CloudWatch Resources (p. 62)

Delete Your Kinesis Data Analytics Application

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. In the Kinesis Data Analytics panel, choose MyApplication.
3. Choose Configure.
4. In the Snapshots section, choose Disable and then choose Update.
5. In the application's page, choose Delete and then confirm the deletion.

Delete Your Kinesis Data Streams
2. In the Kinesis Data Streams panel, choose ExampleInputStream.
3. In the ExampleInputStream page, choose Delete Kinesis Stream and then confirm the deletion.
4. In the Kinesis streams page, choose the ExampleOutputStream, choose Actions, choose Delete, and then confirm the deletion.

Delete Your Amazon S3 Object and Bucket
1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the ka-app-code-<username> bucket.
3. Choose Delete and then enter the bucket name to confirm deletion.

Delete Your IAM Resources
1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation bar, choose Policies.
3. In the filter control, enter kinesis.
4. Choose the kinesis-analytics-service-MyApplication-<your-region> policy.
5. Choose Policy Actions and then choose Delete.
6. In the navigation bar, choose Roles.
7. Choose the kinesis-analytics-MyApplication-<your-region> role.
8. Choose Delete role and then confirm the deletion.

Delete Your CloudWatch Resources
2. In the navigation bar, choose Logs.
3. Choose the /aws/kinesis-analytics/MyApplication log group.
4. Choose Delete Log Group and then confirm the deletion.

Example: Writing to S3

In this exercise, you create a Kinesis Data Analytics for Java application that has a Kinesis data stream as a source and an Amazon S3 bucket as a sink. Using the sink, you can verify the output of the application in the Amazon S3 console.

Note
To set up required prerequisites for this exercise, first complete the Getting Started (p. 24) exercise.
Create Dependent Resources

Before you create a Kinesis Data Analytics for Java application for this exercise, you create the following dependent resources:

• A Kinesis data stream (ExampleInputStream).
• An Amazon S3 bucket to store the application's code and output (ka-app-<username>)

You can create the Kinesis stream and Amazon S3 bucket using the console. For instructions for creating these resources, see the following topics:

• Creating and Updating Data Streams in the Amazon Kinesis Data Streams Developer Guide. Name your data stream ExampleInputStream.
• How Do I Create an S3 Bucket? in the Amazon Simple Storage Service Developer Guide. Give the Amazon S3 bucket a globally unique name by appending your login name, such as ka-app-<username>.
Create two folders (code and data) in the Amazon S3 bucket.

The application creates the following CloudWatch resources if they don't already exist:

• A log group called /aws/kinesis-analytics-java/MyApplication.
• A log stream called kinesis-analytics-log-stream.

Write Sample Records to the Input Stream

In this section, you use a Python script to write sample records to the stream for the application to process.

Note
This section requires the AWS SDK for Python (Boto).

1. Create a file named stock.py with the following contents:

```python
import json
import boto3
import random
import datetime

kinesis = boto3.client('kinesis')
```
def getReferrer():
    data = {}
    now = datetime.datetime.now()
    str_now = now.isoformat()
    data['EVENT_TIME'] = str_now
    data['TICKER'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
    price = random.random() * 100
    data['PRICE'] = round(price, 2)
    return data

while True:
    data = json.dumps(getReferrer())
    print(data)
    kinesis.put_record(
        StreamName="ExampleInputStream",
        Data=data,
        PartitionKey="partitionkey")

2. Run the stock.py script:

```python
# python stock.py
```

Keep the script running while completing the rest of the tutorial.

### Download and Examine the Application Code

The Java application code for this example is available from GitHub. To download the application code, do the following:

1. Install the Git client if you haven't already. For more information, see [Installing Git](#).
2. Clone the remote repository with the following command:

   ```
   git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-examples
   ```


The application code is located in the `S3StreamingSinkJob.java` file. Note the following about the application code:

- The application uses a Kinesis source to read from the source stream. The following snippet creates the Kinesis source:

```java
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                                                     new SimpleStringSchema(),
                                                     inputProperties));
```

- The application uses an Apache Flink S3 sink to write to Amazon S3.

```
input.flatMap(new Tokenizer()) // Tokenizer for generating words
.keyBy(0) // Logically partition the stream for each word
.timeWindow(Time.minutes(1)) // Tumbling window definition
.sum(1) // Sum the number of words per partition
.map(value -> value.f0 + " count: " + value.f1.toString() + "\n")
.addSink(createS3SinkFromStaticConfig());
```
Modify the Application Code

In this section, you modify the application code to write output to your Amazon S3 bucket.

Update the following line with your username to specify the application’s output location:

```java
private static final String s3SinkPath = "s3a://ka-app-<username>/data";
```

Compile the Application Code

To compile the application, do the following:

1. Install Java and Maven if you haven't already. For more information, see Prerequisites (p. 24) in the Getting Started (p. 24) tutorial.
2. In order to use the Kinesis connector for the following application, you need to download, build, and install Apache Maven. For more information, see the section called “Using the Apache Flink Kinesis Streams Connector” (p. 3).
3. Compile the application with the following command:

   ```bash
   mvn package -Dflink.version=1.8.2
   ```

Compiling the application creates the application JAR file `target/aws-kinesis-analytics-java-apps-1.0.jar`.

Note

The provided source code relies on libraries from Java 1.8. If you are using a development environment, ensure that your project's Java version is 1.8.

Upload the Apache Flink Streaming Java Code

In this section, you upload your application code to the Amazon S3 bucket you created in the Create Dependent Resources (p. 63) section.

1. In the Amazon S3 console, choose the `ka-app-<username>` bucket, navigate to the code folder, and choose **Upload**.
2. In the **Select files** step, choose **Add files**. Navigate to the `aws-kinesis-analytics-java-apps-1.0.jar` file that you created in the previous step.
3. You don't need to change any of the settings for the object, so choose **Upload**.

Your application code is now stored in an Amazon S3 bucket where your application can access it.

Create and Run the Kinesis Data Analytics Application

Follow these steps to create, configure, update, and run the application using the console.

Create the Application

2. On the Amazon Kinesis Data Analytics dashboard, choose **Create analytics application**.
3. On the **Kinesis Analytics - Create application** page, provide the application details as follows:
   - For **Application name**, enter **MyApplication**.
   - For **Runtime**, choose **Apache Flink**.
   - Leave the version pulldown as **Apache Flink 1.8 (Recommended Version)**.
4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
5. Choose Create application.

**Note**
When you create a Kinesis Data Analytics for Java application using the console, you have the option of having an IAM role and policy created for your application. Your application uses this role and policy to access its dependent resources. These IAM resources are named using your application name and Region as follows:

- Policy: kinesis-analytics-service-MyApplication-us-west-2
- Role: kinesis-analytics-MyApplication-us-west-2

**Edit the IAM Policy**

Edit the IAM policy to add permissions to access the Kinesis data stream.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Policies. Choose the kinesis-analytics-service-MyApplication-us-west-2 policy that the console created for you in the previous section.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (012345678901) with your account ID. Replace <username> with your username.

```json
{
    "Sid": "ReadCode",
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
    ],
    "Resource": [
        "arn:aws:s3:::kinesis-analytics-placeholder-s3-bucket/kinesis-
        analytics-placeholder-s3-object"
    ],
    "Sid": "ListCloudwatchLogGroups",
    "Effect": "Allow",
    "Action": ["logs:DescribeLogGroups"
    ],
    "Resource": [
        "arn:aws:logs:region:account-id:log-group:*"
    ],
    "Sid": "ListCloudwatchLogStreams",
    "Effect": "Allow",
    "Action": ["logs:DescribeLogStreams"
    ],
    "Resource": [
    ],
    "Sid": "PutCloudwatchLogs",
    "Effect": "Allow",
    "Action": ["logs:PutLogEvents"
    ]
}
```
Configure the Application

1. On the MyApplication page, choose Configure.

2. On the Configure application page, provide the Code location:
   - For Amazon S3 bucket, enter ka-app-<username>.
   - For Path to Amazon S3 object, enter code/aws-kinesis-analytics-java-apps-1.0.jar.


4. Under Monitoring, ensure that the Monitoring metrics level is set to Application.

5. For CloudWatch logging, select the Enable check box.

6. Choose Update.
Note
When you choose to enable CloudWatch logging, Kinesis Data Analytics creates a log group and log stream for you. The names of these resources are as follows:

- Log group: /aws/kinesis-analytics/MyApplication
- Log stream: kinesis-analytics-log-stream

This log stream is used to monitor the application. This is not the same log stream that the application uses to send results.

Run the Application

2. When the application is running, refresh the page. The console shows the Application graph.

Verify the Application Output

In the Amazon S3 console, open the data folder in your S3 bucket.

After a few minutes, objects containing aggregated data from the application will appear.

Clean Up AWS Resources

This section includes procedures for cleaning up AWS resources created in the Amazon S3 tutorial.

This topic contains the following sections:
- Delete Your Kinesis Data Analytics Application (p. 70)
- Delete Your Kinesis Data Stream (p. 70)
- Delete Your Amazon S3 Objects and Bucket (p. 70)
- Delete Your IAM Resources (p. 71)
- Delete Your CloudWatch Resources (p. 71)

Delete Your Kinesis Data Analytics Application

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. In the Kinesis Data Analytics panel, choose MyApplication.
3. Choose Configure.
4. In the Snapshots section, choose Disable and then choose Update.
5. In the application's page, choose Delete and then confirm the deletion.

Delete Your Kinesis Data Stream

2. In the Kinesis Data Streams panel, choose ExampleInputStream.
3. In the ExampleInputStream page, choose Delete Kinesis Stream and then confirm the deletion.

Delete Your Amazon S3 Objects and Bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the ka-app-<username> bucket.
3. Choose Delete and then enter the bucket name to confirm deletion.

Delete Your IAM Resources
1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation bar, choose Policies.
3. In the filter control, enter kinesis.
4. Choose the kinesis-analytics-service-MyApplication-<your-region> policy.
5. Choose Policy Actions and then choose Delete.
6. In the navigation bar, choose Roles.
7. Choose the kinesis-analytics-MyApplication-<your-region> role.
8. Choose Delete role and then confirm the deletion.

Delete Your CloudWatch Resources
2. In the navigation bar, choose Logs.
3. Choose the /aws/kinesis-analytics/MyApplication log group.
4. Choose Delete Log Group and then confirm the deletion.

Tutorial: Using a Kinesis Data Analytics for Java application to Replicate Data from One MSK Cluster to Another in a VPC

The following tutorial demonstrates how to create an Amazon VPC with an Amazon MSK cluster and two topics, and how to create a Kinesis Data Analytics for Java application that reads from one Amazon MSK topic and writes to another.

Note
To set up required prerequisites for this exercise, first complete the Getting Started (p. 24) exercise.

This tutorial contains the following sections:
- Create an Amazon VPC with an Amazon MSK cluster (p. 71)
- Create the Application Code (p. 72)
- Upload the Apache Flink Streaming Java Code (p. 72)
- Create the Application (p. 73)
- Configure the Application (p. 74)
- Run the Application (p. 78)
- Test the Application (p. 79)

Create an Amazon VPC with an Amazon MSK cluster

To create a sample VPC and Amazon MSK cluster to access from a Kinesis Data Analytics application, follow the Getting Started Using Amazon MSK tutorial.
When completing the tutorial, note the following:

- In Step 5: Create a Topic, repeat the `kafka-topics.sh --create` command to create a destination topic named `AWSKafkaTutorialTopicDestination`:

  ```bash
  bin/kafka-topics.sh --create --zookeeper ZooKeeperConnectionString --replication-factor 3 --partitions 1 --topic AWSKafkaTutorialTopicDestination
  ```

- Record the bootstrap server list for your cluster. You can get the list of bootstrap servers with the following command (replace `ClusterArn` with the ARN of your MSK cluster):

  ```bash
  aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
  ```

- When following the steps in the tutorials, be sure to use your selected AWS region in your code, commands, and console entries.

Create the Application Code

In this section, you'll download and compile the application JAR file.

The Java application code for this example is available from GitHub. To download the application code, do the following:

1. Install the Git client if you haven't already. For more information, see Installing Git.
2. Clone the remote repository with the following command:

   ```bash
   git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-examples
   ```

3. The application code is located in the `amazon-kinesis-data-analytics-java-examples/KafkaConnectors/KafkaGettingStartedJob.java` file. You can examine the code to familiarize yourself with the structure of Kinesis Data Analytics for Java application code.

4. Use either the command-line Maven tool or your preferred development environment to create the JAR file. To compile the JAR file using the command-line Maven tool, enter the following:

   ```bash
   mvn package -Dflink.version=1.8.2
   ```

   If the build is successful, the following file is created:

   ```bash
   target/KafkaGettingStartedJob-1.0.jar
   ```

   **Note**

   The provided source code relies on libraries from Java 1.8. If you are using a development environment, ensure that your project's Java version is 1.8.

Upload the Apache Flink Streaming Java Code

In this section, you upload your application code to the Amazon S3 bucket you created in the Getting Started (p. 24) tutorial.
Create the Application

1. Open the Kinesis Data Analytics console at  https://console.aws.amazon.com/kinesisanalytics.
2. On the Amazon Kinesis Data Analytics dashboard, choose Create analytics application.
3. On the Kinesis Analytics - Create application page, provide the application details as follows:
   - For Application name, enter MyApplication.
   - For Runtime, choose Apache Flink 1.8.

Your application code is now stored in an Amazon S3 bucket where your application can access it.
5. Choose **Create application**.

**Note**
When you create a Kinesis Data Analytics for Java application using the console, you have the option of having an IAM role and policy created for your application. Your application uses this role and policy to access its dependent resources. These IAM resources are named using your application name and Region as follows:

- **Policy**: kinesis-analytics-service-MyApplication-us-west-2
- **Role**: kinesis-analytics-MyApplication-us-west-2

### Configure the Application

1. On the **MyApplication** page, choose **Configure**.

![MyApplication page](image.png)

2. On the **Configure application** page, provide the **Code location**:

   - For **Amazon S3 bucket**, enter `ka-app-code-<username>`.
   - For **Path to Amazon S3 object**, enter `KafkaGettingStartedJob-1.0.jar`.


**Note**
When you specify application resources using the console (such as CloudWatch Logs or an Amazon VPC), the console modifies your application execution role to grant permission to access those resources.
4. Under Properties, choose Add Group. Create a property group named **KafkaSource** with the following properties:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>AWSKafkaTutorialTopic</td>
</tr>
<tr>
<td>bootstrap.servers</td>
<td><em>The bootstrap server list you saved previously</em></td>
</tr>
<tr>
<td>security.protocol</td>
<td>SSL</td>
</tr>
<tr>
<td>ssl.truststore.location</td>
<td>/usr/local/openjdk-8/jre/lib/security/cacerts</td>
</tr>
<tr>
<td>ssl.truststore.password</td>
<td>changeit</td>
</tr>
</tbody>
</table>
**Note**
The `ssl.truststore.password` for the default certificate is "changeit"; you do not need to change this value if you are using the default certificate.

Choose **Add Group** again. Create a property group named **KafkaSink** with the following properties:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>AWSKafkaTutorialTopicDestination</td>
</tr>
<tr>
<td>bootstrap.servers</td>
<td>The bootstrap server list you saved previously</td>
</tr>
<tr>
<td>security.protocol</td>
<td>SSL</td>
</tr>
<tr>
<td>ssl.truststore.location</td>
<td>/usr/local/openjdk-8/jre/lib/security/cacerts</td>
</tr>
<tr>
<td>ssl.truststore.password</td>
<td>changeit</td>
</tr>
</tbody>
</table>
The application code reads the above application properties to configure the source and sink used to interact with your VPC and Amazon MSK cluster. For more information about using properties, see Runtime Properties (p. 10).

5. Under Snapshots, choose Disable. This will make it easier to update the application without loading invalid application state data.

6. Under Monitoring, ensure that the Monitoring metrics level is set to Application.

7. For CloudWatch logging, choose the Enable check box.

8. In the Virtual Private Cloud (VPC) section, choose the VPC to associate with your application. Choose the subnets and security group associated with your VPC that you want the application to use to access VPC resources.
9. Choose **Update**.

**Note**
When you choose to enable CloudWatch logging, Kinesis Data Analytics creates a log group and log stream for you. The names of these resources are as follows:

- Log group: /aws/kinesis-analytics/MyApplication
- Log stream: kinesis-analytics-log-stream

This log stream is used to monitor the application.

**Run the Application**

1. On the **MyApplication** page, choose **Run**. Confirm the action.
2. When the application is running, refresh the page. The console shows the Application graph.

Test the Application

In this section, you write records to the source topic. The application reads records from the source topic and writes them to the destination topic. You verify the application is working by writing records to the source topic and reading records from the destination topic.

To write and read records from the topics, follow the steps in Step 6: Produce and Consume Data in the Getting Started Using Amazon MSK tutorial.

To read from the destination topic, use the destination topic name instead of the source topic in your second connection to the cluster:

```
bin/kafka-console-consumer.sh --bootstrap-server BootstrapBrokerString --consumer.config client.properties --topic AWSKafkaTutorialTopicDestination --from-beginning
```

If no records appear in the destination topic, see the Cannot access resources in a VPC (p. 167) section in the Troubleshooting (p. 167) topic.

Example: Writing to Kinesis Data Firehose

In this exercise, you create a Kinesis Data Analytics for Java application that has a Kinesis data stream as a source and a Kinesis Data Firehose delivery stream as a sink. Using the sink, you can verify the output of the application in an Amazon S3 bucket.

**Note**
To set up required prerequisites for this exercise, first complete the Getting Started (p. 24) exercise.

This section contains the following steps:
- Create Dependent Resources (p. 80)
- Write Sample Records to the Input Stream (p. 80)
Create Dependent Resources

Before you create a Kinesis Data Analytics for Java application for this exercise, you create the following dependent resources:

- A Kinesis data stream (ExampleInputStream)
- A Kinesis Data Firehose delivery stream that the application writes output to (ExampleDeliveryStream).
- An Amazon S3 bucket to store the application's code (ka-app-code-<username>)

You can create the Kinesis stream, Amazon S3 buckets, and Kinesis Data Firehose delivery stream using the console. For instructions for creating these resources, see the following topics:

- Creating and Updating Data Streams in the Amazon Kinesis Data Streams Developer Guide. Name your data stream ExampleInputStream.
- Creating an Amazon Kinesis Data Firehose Delivery Stream in the Amazon Kinesis Data Firehose Developer Guide. Name your delivery stream ExampleDeliveryStream. When you create the Kinesis Data Firehose delivery stream, also create the delivery stream's S3 destination and IAM role.
- How Do I Create an S3 Bucket? in the Amazon Simple Storage Service Developer Guide. Give the Amazon S3 bucket a globally unique name by appending your login name, such as ka-app-code-<username>.

Write Sample Records to the Input Stream

In this section, you use a Python script to write sample records to the stream for the application to process.

**Note**
This section requires the AWS SDK for Python (Boto).

1. Create a file named stock.py with the following contents:

```python
import json
import boto3
import random
import datetime

kinesis = boto3.client('kinesis')
def getReferrer():
    data = {}
    now = datetime.datetime.now()
    str_now = now.isoformat()
    data['EVENT_TIME'] = str_now
    data['TICKER'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
    price = random.random() * 100
    data['PRICE'] = round(price, 2)
    return data
```
while True:
    data = json.dumps(getReferrer())
    print(data)
    kinesis.put_record(
        StreamName="ExampleInputStream",
        Data=data,
        PartitionKey="partitionkey")

2. Run the stock.py script:

    $ python stock.py

Keep the script running while completing the rest of the tutorial.

Download and Examine the Apache Flink Streaming Java Code

The Java application code for this example is available from GitHub. To download the application code, do the following:

1. Clone the remote repository with the following command:

    git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-examples


The application code is located in the FirehoseSinkStreamingJob.java file. Note the following about the application code:

- The application uses a Kinesis source to read from the source stream. The following snippet creates the Kinesis source:

    return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                                                new SimpleStringSchema(), inputProperties));

- The application uses a Kinesis Data Firehose sink to write data to a delivery stream. The following snippet creates the Kinesis Data Firehose sink:

    FlinkKinesisFirehoseProducer<String> sink = new
    FlinkKinesisFirehoseProducer<>((outputDeliveryStreamName, new SimpleStringSchema(),
                                 outputProperties));

Compile the Application Code

To compile the application, do the following:

1. Install Java and Maven if you haven't already. For more information, see Prerequisites (p. 24) in the Getting Started (p. 24) tutorial.

2. In order to use the Kinesis connector for the following application, you need to download, build, and install Apache Maven. For more information, see the section called “Using the Apache Flink Kinesis Streams Connector” (p. 3).
3. Compile the application with the following command:

```
mvn package -Dflink.version=1.8.2
```

**Note**

The provided source code relies on libraries from Java 1.8. If you are using a development environment, ensure that your project's Java version is 1.8.

Compiling the application creates the application JAR file (`target/aws-kinesis-analytics-java-apps-1.0.jar`).

**Upload the Apache Flink Streaming Java Code**

In this section, you upload your application code to the Amazon S3 bucket that you created in the Create Dependent Resources (p. 80) section.

**To upload the application code**

1. Open the Amazon S3 console at [https://console.aws.amazon.com/s3/](https://console.aws.amazon.com/s3/).
2. In the console, choose the `ka-app-code-<username>` bucket, and then choose **Upload**.
3. In the **Select files** step, choose **Add files**. Navigate to the `java-getting-started-1.0.jar` file that you created in the previous step.
4. You don't need to change any of the settings for the object, so choose **Upload**.

Your application code is now stored in an Amazon S3 bucket where your application can access it.

**Create and Run the Kinesis Data Analytics Application**

You can create and run a Kinesis Data Analytics for Java application using either the console or the AWS CLI.

**Note**

When you create the application using the console, your AWS Identity and Access Management (IAM) and Amazon CloudWatch Logs resources are created for you. When you create the application using the AWS CLI, you create these resources separately.

**Topics**

- Create and Run the Application (Console) (p. 82)
- Create and Run the Application (AWS CLI) (p. 87)

**Create and Run the Application (Console)**

Follow these steps to create, configure, update, and run the application using the console.

**Create the Application**

2. On the Amazon Kinesis Data Analytics dashboard, choose **Create analytics application**.
3. On the **Kinesis Analytics - Create application** page, provide the application details as follows:
   - For **Application name**, enter `MyApplication`.
   - For **Description**, enter `My java test app`.
• For **Runtime**, choose **Apache Flink**.

  **Note**
  Kinesis Data Analytics for Java Applications uses Apache Flink version 1.8.2.

• Leave the version pulldown as **Apache Flink 1.8 (Recommended Version)**.

4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.

5. Choose **Create application**.
Create and Run the Kinesis Data Analytics Application

Note
When you create the application using the console, you have the option of having an IAM role and policy created for your application. The application uses this role and policy to access its dependent resources. These IAM resources are named using your application name and Region as follows:

- Policy: kinesis-analytics-service-MyApplication-us-west-2
- Role: kinesis-analytics-MyApplication-us-west-2

Edit the IAM Policy

Edit the IAM policy to add permissions to access the Kinesis data stream and Kinesis Data Firehose delivery stream.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Policies. Choose the kinesis-analytics-service-MyApplication-us-west-2 policy that the console created for you in the previous section.
4. Add the highlighted section of the following policy example to the policy. Replace all the instances of the sample account IDs (012345678901) with your account ID.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-started-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ]
        }
    ]
}
```
Create and Run the Kinesis Data Analytics Application

Configure the Application

1. On the **MyApplication** page, choose **Configure**.

2. On the **Configure application** page, provide the **Code location**:
   - For **Amazon S3 bucket**, enter `ka-app-code-<username>`.  
   - For **Path to Amazon S3 object**, enter `java-getting-started-1.0.jar`.

3. Under **Access to application resources**, for **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.

4. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.

5. For **CloudWatch logging**, select the **Enable** check box.

6. Choose **Update**.
Note
When you choose to enable CloudWatch logging, Kinesis Data Analytics creates a log group and log stream for you. The names of these resources are as follows:

- Log group: /aws/kinesis-analytics/MyApplication
- Log stream: kinesis-analytics-log-stream
Run the Application


![MyApplication page](image)

2. When the application is running, refresh the page. The console shows the Application graph.

![Application graph](image)

Stop the Application

On the MyApplication page, choose Stop. Confirm the action.

![MyApplication page](image)

Update the Application

Using the console, you can update application settings such as application properties, monitoring settings, and the location or file name of the application JAR.

On the MyApplication page, choose Configure. Update the application settings and choose Update.

**Note**

To update the application's code on the console, you must either change the object name of the JAR, use a different S3 bucket, or use the AWS CLI as described in the section called "Update the Application Code" (p. 91) section. If the file name or the bucket does not change, the application code is not reloaded when you choose Update on the Configure page.

Create and Run the Application (AWS CLI)

In this section, you use the AWS CLI to create and run the Kinesis Data Analytics application.
Create a Permissions Policy

First, you create a permissions policy with two statements: one that grants permissions for the read action on the source stream, and another that grants permissions for write actions on the sink stream. You then attach the policy to an IAM role (which you create in the next section). Thus, when Kinesis Data Analytics assumes the role, the service has the necessary permissions to read from the source stream and write to the sink stream.

Use the following code to create the `KAReadSourceStreamWriteSinkStream` permissions policy. Replace `username` with the user name that you will use to create the Amazon S3 bucket to store the application code. Replace the account ID in the Amazon Resource Names (ARNs) (012345678901) with your account ID.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": ["s3:GetObject", "s3:GetObjectVersion"],
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
        },
        {
            "Sid": "WriteDeliveryStream",
            "Effect": "Allow",
            "Action": "firehose:*",
        }
    ]
}
```

For step-by-step instructions to create a permissions policy, see Tutorial: Create and Attach Your First Customer Managed Policy in the IAM User Guide.

**Note**

To access other AWS services, you can use the AWS SDK for Java. Kinesis Data Analytics automatically sets the credentials required by the SDK to those of the service execution IAM role that is associated with your application. No additional steps are needed.

Create an IAM Role

In this section, you create an IAM role that the Kinesis Data Analytics for Java application can assume to read a source stream and write to the sink stream.

Kinesis Data Analytics cannot access your stream if it doesn't have permissions. You grant these permissions via an IAM role. Each IAM role has two policies attached. The trust policy grants Kinesis Data Analytics permission to assume the role. The permissions policy determines what Kinesis Data Analytics can do after assuming the role.

You attach the permissions policy that you created in the preceding section to this role.
To create an IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles, Create Role.
3. Under Select type of trusted identity, choose AWS Service. Under Choose the service that will use this role, choose Kinesis. Under Select your use case, choose Kinesis Analytics.

Choose Next: Permissions.
4. On the Attach permissions policies page, choose Next: Review. You attach permissions policies after you create the role.
5. On the Create role page, enter KA-stream-rw-role for the Role name. Choose Create role.

Now you have created a new IAM role called KA-stream-rw-role. Next, you update the trust and permissions policies for the role.
6. Attach the permissions policy to the role.

   Note
   For this exercise, Kinesis Data Analytics assumes this role for both reading data from a Kinesis data stream (source) and writing output to another Kinesis data stream. So you attach the policy that you created in the previous step, the section called “Create a Permissions Policy” (p. 88).

   a. On the Summary page, choose the Permissions tab.
   b. Choose Attach Policies.
   c. In the search box, enter KAReadSourceStreamWriteSinkStream (the policy that you created in the previous section).
   d. Choose the KAReadStreamWriteOutputStream policy, and choose Attach policy.

You now have created the service execution role that your application will use to access resources. Make a note of the ARN of the new role.

For step-by-step instructions for creating a role, see Creating an IAM Role (Console) in the IAM User Guide.

Create the Kinesis Data Analytics Application

1. Save the following JSON code to a file named create_request.json. Replace the sample role ARN with the ARN for the role that you created previously. Replace the bucket ARN suffix with the suffix that you chose in the the section called “Create Dependent Resources” (p. 80) section (ka-app-code-<username>). Replace the sample account ID (012345678901) in the service execution role with your account ID.

   ```json
   {
   "ApplicationName": "test",
   "ApplicationDescription": "my java test app",
   "RuntimeEnvironment": "FLINK-1_8",
   "ServiceExecutionRole": "arn:aws:iam::012345678901:role/KA-stream-rw-role",
   "ApplicationConfiguration": {
      "ApplicationCodeConfiguration": {
         "CodeContent": {
            "S3ContentLocation": {
               "BucketARN": "arn:aws:s3:::ka-app-code-username",
               "FileKey": "java-getting-started-1.0.jar"
            }
         },
         "CodeContentType": "ZIPFILE"
      }
   }
   }
   ```
Create and Run the Kinesis Data Analytics Application

2. Execute the `CreateApplication` action with the preceding request to create the application:

   ```bash
   aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
   ```

   The application is now created. You start the application in the next step.

Start the Application

In this section, you use the `StartApplication` action to start the application.

To start the application

1. Save the following JSON code to a file named `start_request.json`.

   ```json
   {
     "ApplicationName": "test",
     "RunConfiguration": {
       "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
       }
     }
   }
   ```

2. Execute the `StartApplication` action with the preceding request to start the application:

   ```bash
   aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
   ```

   The application is now running. You can check the Kinesis Data Analytics metrics on the Amazon CloudWatch console to verify that the application is working.

Stop the Application

In this section, you use the `StopApplication` action to stop the application.

To stop the application

1. Save the following JSON code to a file named `stop_request.json`.

   ```json
   {"ApplicationName": "test"}
   ```

2. Execute the `StopApplication` action with the following request to stop the application:

   ```bash
   aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
   ```

   The application is now stopped.

Add a CloudWatch Logging Option

You can use the AWS CLI to add an Amazon CloudWatch log stream to your application. For information about using CloudWatch Logs with your application, see the section called “Setting Up Logging” (p. 116).
Update the Application Code

When you need to update your application code with a new version of your code package, you use the UpdateApplication AWS CLI action.

To use the AWS CLI, delete your previous code package from your Amazon S3 bucket, upload the new version, and call UpdateApplication, specifying the same Amazon S3 bucket and object name.

The following sample request for the UpdateApplication action reloads the application code and restarts the application. Update the CurrentApplicationVersionId to the current application version. You can check the current application version using the ListApplications or DescribeApplication actions. Update the bucket name suffix (<username>) with the suffix you chose in the the section called “Create Dependent Resources” (p. 80) section.

```json
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-<username>",
                    "FileKeyUpdate": "java-getting-started-1.0.jar"
                }
            }
        }
    }
}
```

Clean Up AWS Resources

This section includes procedures for cleaning up AWS resources created in the Getting Started tutorial.

This topic contains the following sections:
- Delete Your Kinesis Data Analytics Application (p. 91)
- Delete Your Kinesis Data Stream (p. 91)
- Delete Your Kinesis Data Firehose Delivery Stream (p. 92)
- Delete Your Amazon S3 Object and Bucket (p. 92)
- Delete Your IAM Resources (p. 92)
- Delete Your CloudWatch Resources (p. 92)

Delete Your Kinesis Data Analytics Application

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. In the Kinesis Data Analytics panel, choose MyApplication.
3. Choose Configure.
4. In the Snapshots section, choose Disable and then choose Update.
5. In the application's page, choose Delete and then confirm the deletion.

Delete Your Kinesis Data Stream

2. In the Kinesis Data Streams panel, choose ExampleInputStream.
3. In the ExampleInputStream page, choose Delete Kinesis Stream and then confirm the deletion.

Delete Your Kinesis Data Firehose Delivery Stream

2. In the Kinesis Data Firehose panel, choose ExampleDeliveryStream.
3. In the ExampleDeliveryStream page, choose Delete delivery stream and then confirm the deletion.

Delete Your Amazon S3 Object and Bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the ka-app-code-<username> bucket.
3. Choose Delete and then enter the bucket name to confirm deletion.
4. If you created an Amazon S3 bucket for your Kinesis Data Firehose delivery stream's destination, delete that bucket too.

Delete Your IAM Resources

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation bar, choose Policies.
3. In the filter control, enter kinesis.
4. Choose the kinesis-analytics-service-MyApplication-<your-region> policy.
5. Choose Policy Actions and then choose Delete.
6. If you created a new policy for your Kinesis Data Firehose delivery stream, delete that policy too.
7. In the navigation bar, choose Roles.
8. Choose the kinesis-analytics-MyApplication-<your-region> role.
9. Choose Delete role and then confirm the deletion.
10. If you created a new role for your Kinesis Data Firehose delivery stream, delete that role too.

Delete Your CloudWatch Resources

2. In the navigation bar, choose Logs.
3. Choose the /aws/kinesis-analytics/MyApplication log group.
4. Choose Delete Log Group and then confirm the deletion.

Example: Using a Custom Sink to Write to CloudWatch Logs

In this exercise, you create a Kinesis Data Analytics for Java application that has a Kinesis data stream as a source and an Amazon CloudWatch log stream as a sink. Using the sink, you can verify the output of the application in the CloudWatch console.

Note
To set up required prerequisites for this exercise, first complete the Getting Started (p. 24) exercise.
Create Dependent Resources

Before you create a Kinesis Data Analytics for Java application for this exercise, you create the following dependent resources:

- A Kinesis data stream (ExampleInputStream).
- An Amazon S3 bucket to store the application's code (ka-app-code-<username>)

You can create the Kinesis stream and Amazon S3 bucket using the console. For instructions for creating these resources, see the following topics:

- Creating and Updating Data Streams in the Amazon Kinesis Data Streams Developer Guide. Name your data stream ExampleInputStream.
- How Do I Create an S3 Bucket? in the Amazon Simple Storage Service Developer Guide. Give the Amazon S3 bucket a globally unique name by appending your login name, such as ka-app-code-<username>.

The application creates the following CloudWatch resources if they don't already exist:

- A log group called /aws/kinesis-analytics-java/test.
- A log stream called StockPriceStatistics. You can view this log stream to verify that the application is working.

Write Sample Records to the Input Stream

In this section, you use a Python script to write sample records to the stream for the application to process.

Note
This section requires the AWS SDK for Python (Boto).

1. Create a file named stock.py with the following contents:

```python
import json
import boto3
import random
import datetime

kinesis = boto3.client('kinesis')
def getReferrer():
    data = {}
```
Download and Examine the Application Code

The Java application code for this example is available from GitHub. To download the application code, do the following:

1. Install the Git client if you haven’t already. For more information, see Installing Git.
2. Clone the remote repository with the following command:

   ```
   git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-examples
   ```


   The application code is located in the `CustomSinkStreamingJob.java` and `CloudWatchLogSink.java` files. Note the following about the application code:

   • The application uses a Kinesis source to read from the source stream. The following snippet creates the Kinesis source:

   ```
   return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
               new SimpleStringSchema(), inputProperties));
   ```

   • The application uses a custom Flink sink to write to CloudWatch Logs. The sink is defined in the `CloudWatchLogSink.java` file.

   The sink sends messages to the log stream in batches. The following code writes messages to the batch, and sends the batch to CloudWatch Logs. It sends it either in intervals of `MAX_BUFFER_TIME_MILLIS` (60 seconds), or if the batch reaches `MAX_BATCH_SIZE` in length (10,000 messages):

   ```
   logEvents.add(new
               InputLogEvent().withMessage(message).withTimestamp(System.currentTimeMillis()));
   if (logEvents.size() >= MAX_BATCH_SIZE || lastFlushTimeMillis +
               MAX_BUFFER_TIME_MILLIS <= System.currentTimeMillis()) { 
   ```

2. Run the `stock.py` script:

   ```
   # python stock.py
   ```

   Keep the script running while completing the rest of the tutorial.
// flush the messages
PutLogEventsRequest putLogEventsRequest = new PutLogEventsRequest()
   .withLogEvents(logEvents)
   .withLogGroupName(logGroupName)
   .withLogStreamName(logStreamName)
   .withSequenceToken(getUploadSequenceToken());
awsLogsClient.putLogEvents(putLogEventsRequest);
lastFlushTimeMillis = System.currentTimeMillis();
logEvents.clear();
}

- The application creates an output stream that contains the maximum price for each stock within a
  10-minute window that slides by 5 minutes. For more information about Flink time windows, see
  Windows in the Apache Flink documentation. The output stream that the sink creates then sends the
  aggregate data to the CloudWatch Logs sink.

The following code from the CustomSinkStreamingJob.java file sends aggregate data to the
CloudWatch Logs sink:

input.map(value -> {
   JsonNode jsonNode = jsonParser.readValue(value, JsonNode.class);
   return new Tuple2<>(jsonNode.get("TICKER").asText(),
   jsonNode.get("PRICE").asDouble());
}).returns(Types.TUPLE(Types.STRING, Types.DOUBLE))
   .keyBy(0)
   .timeWindow(Time.seconds(10), Time.seconds(5))
   .max(1)
   .map(value -> value.f0 + ":  max - " + value.f1.toString() + "\n")
   .addSink(new CloudWatchLogSink(region, CLOUD_WATCH_LOG_GROUP,
   CLOUD_WATCH_LOG_STREAM));

## Compile the Application Code

To compile the application, do the following:

1. Install Java and Maven if you haven't already. For more information, see Prerequisites (p. 24) in the
   Getting Started (p. 24) tutorial.

2. In order to use the Kinesis connector for the following application, you need to download, build,
   and install Apache Maven. For more information, see the section called “Using the Apache Flink
   Kinesis Streams Connector” (p. 3).

3. Compile the application with the following command:

   mvn package -Dflink.version=1.8.2

   **Note**
   
   The provided source code relies on libraries from Java 1.8. If you are using a development
   environment, ensure that your project's Java version is 1.8.

   Compiling the application creates the application JAR file target/aws-kinesis-analytics-java-
   apps-1.0.jar.

## Upload the Apache Flink Streaming Java Code

In this section, you upload your application code to the Amazon S3 bucket you created in the Create
Dependent Resources (p. 93) section.
1. In the Amazon S3 console, choose the ka-app-code-<username> bucket, and choose **Upload**.

2. In the **Select files** step, choose **Add files**. Navigate to the aws-kinesis-analytics-java-apps-1.0.jar file that you created in the previous step.

3. You don't need to change any of the settings for the object, so choose **Upload**.

Your application code is now stored in an Amazon S3 bucket where your application can access it.

## Create and Run the Kinesis Data Analytics Application

Follow these steps to create, configure, update, and run the application using the console.

### Create the Application

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.

2. On the Amazon Kinesis Data Analytics dashboard, choose **Create analytics application**.

3. On the **Kinesis Analytics - Create application** page, provide the application details as follows:

   - For **Application name**, enter **MyApplication**.
   - For **Runtime**, choose **Apache Flink**.
   - Leave the version pulldown as **Apache Flink 1.8 (Recommended Version)**.

4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.

   **TODO: Update or remove screenshot**
5. Choose **Create application**.

**Note**
When you create a Kinesis Data Analytics for Java application using the console, you have the option of having an IAM role and policy created for your application. Your application uses this role and policy to access its dependent resources. These IAM resources are named using your application name and Region as follows:
Edit the IAM Policy

Edit the IAM policy to add permissions to access the Kinesis data streams.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Policies. Choose the kinesis-analytics-service-MyApplication-us-west-2 policy that the console created for you in the previous section.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (012345678901) with your account ID.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "logs:DescribeLogGroups",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-apps-1.0.jar"
      ]
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": "logs:DescribeLogStreams",
    },
    {
      "Sid": "PutLogEvents",
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
    },
    {
      "Sid": "DescribeLogGroups",
      "Effect": "Allow",
      "Action": ["logs:DescribeLogGroups"
      ],
      ]
    },
    {
      "Sid": "ReadInputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
    }
  ]
}
```
Configure the Application

1. On the MyApplication page, choose Configure.

![Configure Application Image]

2. On the Configure application page, provide the Code location:
   - For Amazon S3 bucket, enter ka-app-code-<username>.
   - For Path to Amazon S3 object, enter aws-kinesis-analytics-java-apps-1.0.jar.
4. Under Monitoring, ensure that the Monitoring metrics level is set to Application.
5. For CloudWatch logging, select the Enable check box.
6. Choose Update.
Note
When you choose to enable CloudWatch logging, Kinesis Data Analytics creates a log group and log stream for you. The names of these resources are as follows:

- Log group: /aws/kinesis-analytics/MyApplication
- Log stream: kinesis-analytics-log-stream

This log stream is used to monitor the application. This is not the same log stream that the application uses to send results.

Run the Application

2. When the application is running, refresh the page. The console shows the **Application graph**.

Verify the Application Output

In the CloudWatch console, open the `/aws/kinesis-analytics-java/test/StockPriceStatistics` log stream.

After a few minutes, the log stream will contain aggregated data from the application.

<table>
<thead>
<tr>
<th>Time (UTC +00:00)</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-03-27</td>
<td></td>
</tr>
<tr>
<td>19:56:55</td>
<td>TBV: max - 83.04</td>
</tr>
<tr>
<td>19:56:55</td>
<td>AAPL: max - 99.21</td>
</tr>
<tr>
<td>19:56:55</td>
<td>MSFT: max - 95.39</td>
</tr>
<tr>
<td>19:56:55</td>
<td>INTC: max - 98.24</td>
</tr>
<tr>
<td>19:57:00</td>
<td>AAPL: max - 99.21</td>
</tr>
<tr>
<td>19:57:00</td>
<td>TBV: max - 97.28</td>
</tr>
</tbody>
</table>

Clean Up AWS Resources

This section includes procedures for cleaning up AWS resources created in the CloudWatch tutorial.

**This topic contains the following sections:**
- Delete Your Kinesis Data Analytics Application (p. 102)
- Delete Your Kinesis Data Streams (p. 102)
- Delete Your Amazon S3 Object and Bucket (p. 102)
- Delete Your IAM Resources (p. 102)
- Delete Your CloudWatch Resources (p. 102)
Delete Your Kinesis Data Analytics Application

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. In the Kinesis Data Analytics panel, choose MyApplication.
3. Choose Configure.
4. In the Snapshots section, choose Disable and then choose Update.
5. In the application's page, choose Delete and then confirm the deletion.

Delete Your Kinesis Data Streams

2. In the Kinesis Data Streams panel, choose ExampleInputStream.
3. In the ExampleInputStream page, choose Delete Kinesis Stream and then confirm the deletion.
4. In the Kinesis streams page, choose the ExampleOutputStream, choose Actions, choose Delete, and then confirm the deletion.

Delete Your Amazon S3 Object and Bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the ka-app-code-<username> bucket.
3. Choose Delete and then enter the bucket name to confirm deletion.

Delete Your IAM Resources

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation bar, choose Policies.
3. In the filter control, enter kinesis.
4. Choose the kinesis-analytics-service-MyApplication-<your-region> policy.
5. Choose Policy Actions and then choose Delete.
6. In the navigation bar, choose Roles.
7. Choose the kinesis-analytics-MyApplication-<your-region> role.
8. Choose Delete role and then confirm the deletion.

Delete Your CloudWatch Resources

2. In the navigation bar, choose Logs.
3. Choose the /aws/kinesis-analytics/MyApplication log group.
4. Choose Delete Log Group and then confirm the deletion.
5. Choose the /aws/kinesis-analytics-java/test log group.
6. Choose Delete Log Group and then confirm the deletion.
Example: Read From a Kinesis Stream in a Different Account

This example demonstrates how to create an Amazon Kinesis Data Analytics application that reads data from a Kinesis stream in a different account. In this example, you will use one account for the source Kinesis stream, and a second account for the Kinesis Data Analytics application and sink Kinesis stream.

This topic contains the following sections:
- Prerequisites (p. 103)
- Setup (p. 103)
- Create Source Kinesis Stream (p. 104)
- Create and Update IAM Roles and Policies (p. 104)
- Update the Python Script (p. 106)
- Update the Java Application (p. 107)
- Build, Upload, and Run the Application (p. 108)

Prerequisites

- In this tutorial, you modify the Getting Started example to read data from a Kinesis stream in a different account. Complete the Getting Started (p. 24) tutorial before proceeding.
- You need two AWS accounts to complete this tutorial: one for the source stream, and one for the application and the sink stream. Use the AWS account you used for the Getting Started tutorial for the application and sink stream. Use a different AWS account for the source stream.

Setup

You will access your two AWS accounts by using named profiles. Modify your AWS credentials and configuration files to include two profiles that contain the region and connection information for your two accounts.

The following example credential file contains two named profiles, ka-source-stream-account-profile and ka-sink-stream-account-profile. Use the account you used for the Getting Started tutorial for the sink stream account.

```
[ka-source-stream-account-profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

[ka-sink-stream-account-profile]
aws_access_key_id=AKIAI44QH8DHBEINSTANCE
aws_secret_access_key=je7MtGbClwBF/2Zp9Ut/k/h3yCo8nvbEXAMPLEKEY
```

The following example configuration file contains the same named profiles with region and output format information.

```
[profile ka-source-stream-account-profile]
region=us-west-2
output=json

[profile ka-sink-stream-account-profile]
```
region=us-west-2
output=json

Note
This tutorial does not use the ka-sink-stream-account-profile. It is included as an example of how to access two different AWS accounts using profiles.

For more information on using named profiles with the AWS CLI, see Named Profiles in the AWS Command Line Interface documentation.

Create Source Kinesis Stream

In this section, you will create the Kinesis stream in the source account.

Enter the following command to create the Kinesis stream that the application will use for input. Note that the --profile parameter specifies which account profile to use.

$ aws kinesis create-stream
   --stream-name SourceAccountExampleInputStream
   --shard-count 1
   --profile ka-source-stream-account-profile

Create and Update IAM Roles and Policies

To allow object access across AWS accounts, you create an IAM role and policy in the source account. Then, you modify the IAM policy in the sink account. For information about creating IAM roles and policies, see the following topics in the AWS Identity and Access Management User Guide:

- Creating IAM Roles
- Creating IAM Policies

Sink Account Roles and Policies

1. Edit the kinesis-analytics-service-MyApplication-us-west-2 policy from the Getting Started tutorial. This policy allows the role in the source account to be assumed in order to read the source stream.

Add the highlighted section below to the policy. Replace the sample account ID (SOURCE01234567) with the ID of the account you will use for the source stream.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AssumeRoleInSourceAccount",
         "Effect": "Allow",
         "Action": "sts:AssumeRole",
         "Resource": "arn:aws:iam::SOURCE01234567:role/KA-Source-Stream-Role"
      },
      {
         "Sid": "ReadCode",
         "Effect": "Allow",
         "Action": [
            "s3:GetObject",
            "s3:GetObjectVersion"
         ],
         "Resource": ["arn:aws:s3:::SOURCE01234567:role/KA-Source-Stream-Role"
```
Create and Update IAM Roles and Policies

2. Open the kinesis-analytics-MyApplication-us-west-2 role, and make a note of its Amazon Resource Name (ARN). You will need it in the next section. The role ARN looks like the following.

```
arn:aws:iam::SINK012345678:role/service-role/kinesis-analytics-MyApplication-us-west-2
```

Source Account Roles and Policies

1. Create a policy in the source account called KA-Source-Stream-Policy. Use the following JSON for the policy. Replace the sample account number with the account number of the source account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:ListShards"
            ],
            "Resource": [
            ]
        }
    ]
}
```
2. Create a role in the source account called KA-Source-Stream-Role. Do the following to create the role using the Kinesis Analytics use case:

1. In the IAM Management Console, choose Create Role.
2. On the Create Role page, choose AWS Service. In the service list, choose Kinesis.
3. In the Select your use case section, choose Kinesis Analytics.
4. Choose Next: Permissions.
5. Add the KA-Source-Stream-Policy permissions policy you created in the previous step. Choose Next:Tags.
6. Choose Next: Review.
7. Name the role KA-Source-Stream-Role. Your application will use this role to access the source stream.

3. Add the kinesis-analytics-MyApplication-us-west-2 ARN from the sink account to the trust relationship of the KA-Source-Stream-Role role in the source account:

1. Open the KA-Source-Stream-Role in the IAM console.
2. Choose the Trust Relationships tab.
3. Choose Edit trust relationship.
4. Use the following code for the trust relationship. Replace the sample account ID (SINK012345678) with your sink account ID.

```json
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Principal": {
"AWS": "arn:aws:iam::SINK012345678:role/service-role/kinesis-analytics-MyApplication-us-west-2"
},
"Action": "sts:AssumeRole"
}
]
}
```

## Update the Python Script

In this section, you update the Python script that generates sample data to use the source account profile.

Update the `stock.py` script with the following highlighted changes.

```python
import json
import boto3
import random
import datetime
import os
```
Update the Java Application

In this section, you update the Java application code to assume the source account role when reading from the source stream.

Make the following changes to the `BasicStreamingJob.java` file. Replace the example source account number (SOURCE01234567) with your source account number.

```java
package com.amazonaws.services.kinesisanalytics;

import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;
import java.io.IOException;
import java.util.Map;
import java.util.Properties;

/**
 * A basic Kinesis Data Analytics for Java application with Kinesis data streams
 * as source and sink.
 */
public class BasicStreamingJob {
    private static final String region = "us-west-2";
    private static final String inputStreamName = "SourceAccountExampleInputStream";
    private static final String outputStreamName = ExampleOutputStream;
    private static final String roleArn = "arn:aws:iam::SOURCE01234567:role/KA-Source-Stream-Role";
    private static final String roleSessionName = "ksassumedrolesession";

    private static DataStream<String>
    createSourceFromStaticConfig(StreamExecutionEnvironment env) {
        Properties inputProperties = new Properties();
        inputProperties.setProperty(AWSConfigConstants.AWS_CREDENTIALS_PROVIDER, "ASSUME_ROLE");
```
Build, Upload, and Run the Application

Do the following to update and run the application:

1. Build the application again by running the following command in the directory with the `pom.xml` file.
   ```bash
   mvn package -Dflink.version=1.8.2
   ```

2. Delete the previous JAR file from your Amazon Simple Storage Service (Amazon S3) bucket, and then upload the new `aws-kinesis-analytics-java-apps-1.0.jar` file to the S3 bucket.

3. In the application's page in the Kinesis Data Analytics console, choose **Configure, Update** to reload the application JAR file.

4. Run the `stock.py` script to send data to the source stream.
   ```bash
   python stock.py
   ```

The application now reads data from the Kinesis stream in the other account.

You can verify that the application is working by checking the `PutRecords.Bytes` metric of the `ExampleOutputStream` stream. If there is activity in the output stream, the application is functioning properly.
Security in Amazon Kinesis Data Analytics for Java Applications

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 for Java Applications, 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 for Java Applications. The following topics show you how to configure Kinesis Data Analytics for Java Applications to meet your security and compliance objectives. You'll also learn how to use other AWS services that can help you to monitor and secure your Kinesis Data Analytics for Java Applications resources.

**Topics**

- Data Protection in Amazon Kinesis Data Analytics for Java Applications (p. 109)
- Identity and Access Management in Amazon Kinesis Data Analytics for Java Applications (p. 110)
- Monitoring Amazon Kinesis Data Analytics (p. 113)
- Compliance Validation for Amazon Kinesis Data Analytics for Java Applications (p. 113)
- Resilience in Amazon Kinesis Data Analytics for Java Applications (p. 113)
- Infrastructure Security in Kinesis Data Analytics for Java Applications (p. 114)
- Security Best Practices for Kinesis Data Analytics for Java Applications (p. 115)

Data Protection in Amazon Kinesis Data Analytics for Java 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 Analytics, Kinesis Data Firehose, and Amazon S3.

**Data Encryption in Kinesis Data Analytics for Java Applications**

**Encryption at Rest**

Note the following about encrypting data at rest with Kinesis Data Analytics for Java Applications:
• 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 Kinesis Data 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 AWS KMS–Managed Keys (SSE-KMS).

• A Kinesis Data Analytics for Java application can read from any streaming source, and write to any streaming or database destination. Ensure that your sources and destinations encrypt all data in transit and data at rest.

• Your application’s code 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 Kinesis Data Firehose.

**Key Management**

Data encryption in Kinesis Data Analytics uses service-managed keys. Customer-managed keys are not supported.

**Identity and Access Management in Amazon Kinesis Data Analytics for Java Applications**

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

**Note**

You must create a permissions policy and role for your application. If you do not create these AWS Identity and Access Management (IAM) resources, your application cannot access its data sources, data destinations, and log streams.

You can grant these permissions by creating an IAM role that Kinesis Data Analytics can assume. Permissions that you grant to this role determine what 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 then. The console uses the following naming convention for IAM roles that it creates.
After the role is created, you can review the role and attached policies in the AWS Identity and Access Management (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.

Topics
- Trust Policy (p. 111)
- Permissions Policy (p. 111)

Trust Policy

To grant 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 Kinesis Data Analytics to read from an application's streaming source, you must grant permissions for relevant read actions. Examples of streaming sources include a Kinesis data stream, an Amazon Kinesis Data Firehose delivery stream, or a reference source in an Amazon Simple Storage Service (Amazon S3) bucket. Depending on your source, you can attach the following permissions policy.

Permissions Policy for Reading a Kinesis Data Stream

In the following example, replace each user input placeholder with your own information.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputKinesis",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": [
                "arn:aws:kinesis:aws-region:aws-account-id:stream/inputStreamName"
            ]
        }
    ]
}
```
Permissions Policy for Writing to a Kinesis Data Stream

In the following example, replace each user input placeholder with your own information.

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

Permissions Policy for Writing to a Kinesis Data Firehose Delivery Stream

In the following example, replace each user input placeholder with your own information.

```json
{
    "Version": "2012-10-17",
    "Statement": [ 
        { 
            "Sid": "WriteOutputFirehose",
            "Effect": "Allow",
            "Action": [ 
                "firehose:DescribeDeliveryStream",
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Resource": [ 
                "arn:aws:firehose:aws-region:aws-account-id:deliverystream/output-firehose-name"
            ]
        }
    ]
}
```

Permissions Policy for Reading from an Amazon S3 Bucket

```json
{
    "Version": "2012-10-17",
    "Statement": [ 
        { 
            "Effect": "Allow",
            "Action": [ 
                "s3:Get*",
                "s3:List*"
            ]
        }
    ]
}
```
Monitoring Amazon Kinesis Data Analytics

Kinesis Data Analytics provides monitoring functionality for your applications. For more information, see *Logging and Monitoring* (p. 116).

Compliance Validation for Amazon Kinesis Data Analytics for Java Applications

Third-party auditors assess the security and compliance of Amazon Kinesis Data Analytics for Java Applications 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 *AWS 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 for Java Applications 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 for Java Applications 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** – 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 for Java Applications

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 for Java Applications 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. Kinesis Data Analytics achieves this through multiple, redundant mechanisms. Each Kinesis Data Analytics application using Apache Flink runs in a single-tenant Apache Flink cluster. The Apache Flink cluster is run with the JobManager in high availability mode using Zookeeper across multiple availability zones. Kinesis Data Analytics deploys Apache Flink using Amazon EKS. Multiple Kubernetes pods are used in Amazon EKS for each AWS region across availability zones. In the event of a failure, Kinesis Data Analytics first tries to recover the application within the running Apache Flink cluster using your application's checkpoints, if available.

Kinesis Data Analytics for Java Applications backs up application state using **Checkpoints** and **Snapshots**:

- **Checkpoints** are backups of application state that Kinesis Data Analytics automatically creates periodically and uses to restore from faults.
- **Snapshots** are backups of application state that you create and restore from manually.

For more information about checkpoints and snapshots, see Fault Tolerance (p. 13).

**Versioning**

Stored versions of application state are versioned as follows:

- **Checkpoints** are versioned automatically by the service. If the service uses a checkpoint to restart the application, the latest checkpoint will be used.
- **Savepoints** are versioned using the **SnapshotName** parameter of the CreateApplicationSnapshot action.

Kinesis Data Analytics encrypts data stored in checkpoints and savepoints.

**Infrastructure Security in Kinesis Data Analytics for Java Applications**

As a managed service, Amazon Kinesis Data Analytics for Java Applications 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 for Java Applications through the network. All API calls to Kinesis Data Analytics are secured via Transport Layer Security (TLS) and authenticated via IAM. Clients must support TLS 1.0 or later. We recommend 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 for Java Applications

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.

Implement least privilege access

When granting permissions, you decide who is getting what permissions to which Kinesis Data Analytics resources. You enable specific actions that you want to allow on those resources. Therefore you should grant only the permissions that are required to perform a task. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

Use IAM roles to access other AWS services

Your Kinesis Data Analytics application must have valid credentials to access resources in other services, such as Kinesis data streams, Kinesis Data 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 (such as a user name and password or access keys) 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, Kinesis Data Firehose delivery streams, and Amazon S3 buckets. For more information on implementing server-side encryption in dependent resources, see Data Protection (p. 109).

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 AWS 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” (p. 132).
Logging and Monitoring in Amazon Kinesis Data Analytics for Java Applications

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Kinesis Data Analytics and your Kinesis Data Analytics for Java applications. 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 Kinesis Data Analytics, 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 Kinesis Data Analytics performance in your environment. You do this by measuring performance at various times and under different load conditions. As you monitor Amazon Kinesis Data Analytics, you can store historical monitoring data. You can then compare it with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

Topics
- Setting Up Application Logging (p. 116)
- Analyzing Logs with CloudWatch Logs Insights (p. 123)
- Viewing Amazon Kinesis Data Analytics Metrics and Dimensions (p. 125)
- Writing Custom Messages to CloudWatch Logs (p. 131)
- Logging Kinesis Data Analytics API Calls with AWS CloudTrail (p. 132)

Setting Up Application Logging

By adding an Amazon CloudWatch logging option to your Amazon Kinesis Data Analytics for Java application, you can monitor for application events or configuration problems.

This topic describes how to configure your application to write application events to a CloudWatch Logs stream. A CloudWatch logging option is a collection of application settings and permissions that your application uses to configure the way it writes application events to CloudWatch Logs. You can add and configure a CloudWatch logging option using either the AWS Management Console or the AWS Command Line Interface (AWS CLI).

Note the following about adding a CloudWatch logging option to your application:

- When you add a CloudWatch logging option using the console, Kinesis Data Analytics creates the CloudWatch log group and log stream for you and adds the permissions your application needs to write to the log stream.
When you add a CloudWatch logging option using the API, you must also create the application's log group and log stream, and add the permissions your application needs to write to the log stream.

This topic contains the following sections:
- Setting Up CloudWatch Logging Using the Console (p. 117)
- Setting Up CloudWatch Logging Using the CLI (p. 118)
- Application Monitoring Levels (p. 121)
- Logging Best Practices (p. 122)
- Logging Troubleshooting (p. 122)
- Next Step (p. 123)

Setting Up CloudWatch Logging Using the Console

When you enable CloudWatch logging for your application in the console, a CloudWatch log group and log stream is created for you. Also, your application's permissions policy is updated with permissions to write to the stream.

The following screenshot shows the **CloudWatch logging** setting in the **Configure application** page.

![CloudWatch logging setting in the Configure application page](image-url)
Kinesis Data Analytics creates a log group named using the following convention, where `ApplicationName` is your application's name.  

```
/aws/kinesis-analytics/ApplicationName
```

Kinesis Data Analytics creates a log stream in the new log group with the following name.

```
kinesis-analytics-log-stream
```

You set the application monitoring metrics level and monitoring log level using the Monitoring log level section of the Configure application page. For information about application log levels, see the section called "Application Monitoring Levels" (p. 121).

**Setting Up CloudWatch Logging Using the CLI**

To add a CloudWatch logging option using the AWS CLI, do the following:

- Create a CloudWatch log group and log stream.
- Add a logging option when you create an application by using the CreateApplication action, or add a logging option to an existing application using the AddApplicationCloudWatchLoggingOption action.
- Add permissions to your application's policy to write to the logs.

**This section contains the following topics:**

- Creating a CloudWatch Log Group and Log Stream (p. 118)
- Working with Application CloudWatch Logging Options (p. 118)
- Adding Permissions to Write to the CloudWatch Log Stream (p. 121)

**Creating a CloudWatch Log Group and Log Stream**

You create a CloudWatch log group and stream using either the CloudWatch Logs console or the API. For information about creating a CloudWatch log group and log stream, see Working with Log Groups and Log Streams.

**Working with Application CloudWatch Logging Options**

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. For information about how to use a JSON file for input for an API action, see Kinesis Data Analytics API Example Code (p. 175).

**Adding a CloudWatch Log Option When Creating an Application**

The following example demonstrates how to use the CreateApplication action to add a CloudWatch log option when you create an application. In the example, replace `Amazon Resource Name (ARN) of the CloudWatch Log stream to add to the new application` with your own information.

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "test-application-description",
    "RuntimeEnvironment": "FLINK-1_0",
    "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
    "ApplicationConfiguration": {
```

---

118
"ApplicationCodeConfiguration": {  
  "CodeContent": {  
    "S3ContentLocation":{  
      "BucketARN": "arn:aws:s3:::mybucket",  
      "FileKey": "myflink.jar"  
    },  
    "CodeContentType": "ZIPFILE"
  },  
  "CloudWatchLoggingOptions": [{  
    "LogStreamARN": "<Amazon Resource Name (ARN) of the CloudWatch log stream to add to the new application>"
  }]
}

Adding a CloudWatch Log Option to an Existing Application

The following example demonstrates how to use the AddApplicationCloudWatchLoggingOption action to add a CloudWatch log option to an existing application. In the example, replace each user input placeholder with your own information. For more information about the action, see AddApplicationCloudWatchLoggingOption.

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

Updating an Existing CloudWatch Log Option

The following example demonstrates how to use the UpdateApplication action to modify an existing CloudWatch log option. In the example, replace each user input placeholder with your own information. For more information about the action, see UpdateApplication.

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

Deleting a CloudWatch Log Option from an Application

The following example demonstrates how to use the DeleteApplicationCloudWatchLoggingOption action to delete an existing CloudWatch log option. In the example, replace each user input placeholder with your own information. For more information about the action, see DeleteApplicationCloudWatchLoggingOption.

```json
{
```

119
Setting the Application Logging Level

To set the level of application logging, use the `MonitoringConfiguration` parameter of the `CreateApplication` action or the `MonitoringConfigurationUpdate` parameter of the `UpdateApplication` action.

For information about application log levels, see the section called "Application Monitoring Levels" (p. 121).

Set the Application Logging Level when Creating an Application

The following example request for the `CreateApplication` action sets the application log level to `INFO`.

```json
{
    "ApplicationName": "MyApplication",
    "ApplicationDescription": "My Application Description",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::mybucket",
                    "FileKey": "myflink.jar",
                    "ObjectVersion": "AbCdEfGhIjKlMnOpQrStUvWxYz12345"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "FlinkApplicationConfiguration": {
            "MonitoringConfiguration": {
                "ConfigurationType": "CUSTOM",
                "LogLevel": "INFO"
            }
        },
        "RuntimeEnvironment": "FLINK-1_8",
        "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole"
    }
}
```

Update the Application Logging Level

The following example request for the `UpdateApplication` action sets the application log level to `INFO`.

```json
{
    "ApplicationConfigurationUpdate": {
        "FlinkApplicationConfigurationUpdate": {
            "MonitoringConfigurationUpdate": {
                "ConfigurationTypeUpdate": "CUSTOM",
                "LogLevelUpdate": "INFO"
            }
        }
    }
}
```
Adding Permissions to Write to the CloudWatch Log Stream

Kinesis Data Analytics needs permissions to write misconfiguration errors to CloudWatch. You can add these permissions to the AWS Identity and Access Management (IAM) role that Kinesis Data Analytics assumes.

For more information about using an IAM role for Kinesis Data Analytics, see Identity and Access Management in Amazon Kinesis Data Analytics for Java Applications (p. 110).

Trust Policy

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

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

Permissions Policy

To grant permissions to an application to write log events to CloudWatch from a Kinesis Data Analytics resource, you can use the following IAM permissions policy. Provide the correct Amazon Resource Names (ARNs) for your log group and stream.

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

Application Monitoring Levels

You control the generation of application log messages using the application's Monitoring Metrics Level and Monitoring Log Level.
The application's monitoring metrics level controls the granularity of log messages. Monitoring metrics levels are defined as follows:

- **Application**: Metrics are scoped to the entire application.
- **Task**: Metrics are scoped to each task. For information about tasks, see the section called “Scaling” (p. 19).
- **Operator**: Metrics are scoped to each operator. For information about operators, see the section called “Operators” (p. 9).
- **Parallelism**: Metrics are scoped to application parallelism. For information about parallelism, see the section called “Scaling” (p. 19).

The application's monitoring log level controls the verbosity of the application's log. Monitoring log levels are defined as follows:

- **Error**: Potential catastrophic events of the application.
- **Warn**: Potentially harmful situations of the application.
- **Info**: Informational and transient failure events of the application. We recommend that you use this logging level.
- **Debug**: Fine-grained informational events that are most useful to debug an application. *Note*: Only use this level for temporary debugging purposes.

**Logging Best Practices**

We recommend that your application use the **Info** logging level. We recommend this level to ensure that you see Apache Flink errors, which are logged at the **Info** level rather than the **Error** level.

We recommend that you use the **Debug** level only temporarily while investigating application issues. Switch back to the **Info** level when the issue is resolved. Using the **Debug** logging level will significantly affect your application's performance.

Excessive logging can also significantly impact application performance. We recommend that you do not write a log entry for every record processed, for example. Excessive logging can cause severe bottlenecks in data processing and can lead to back pressure in reading data from the sources.

**Logging Troubleshooting**

If application logs are not being written to the log stream, verify the following:

- Verify that your application's IAM role and policies are correct. Your application's policy needs the following permissions to access your log stream:
  - `logs:PutLogEvents`
  - `logs:DescribeLogGroups`
  - `logs:DescribeLogStreams`

  For more information, see the section called “Adding Permissions to Write to the CloudWatch Log Stream” (p. 121).

- Verify that your application is running. To check your application's status, view your application's page in the console, or use the `DescribeApplication` or `ListApplications` actions.

- Monitor CloudWatch metrics such as `down_time` to diagnose other application issues. For information about reading CloudWatch metrics, see *Metrics and Dimensions* (p. 125).
Next Step

After you have enabled CloudWatch logging in your application, you can use CloudWatch Logs Insights to analyze your application logs. For more information, see the section called “Analyzing Logs” (p. 123).

Analyzing Logs with CloudWatch Logs Insights

After you've added a CloudWatch logging option to your application as described in the previous section, you can use CloudWatch Logs Insights to query your log streams for specific events or errors.

CloudWatch Logs Insights enables you to interactively search and analyze your log data in CloudWatch Logs.

For information on getting started with CloudWatch Logs Insights, see Analyze Log Data with CloudWatch Logs Insights.

Run a Sample Query

This section describes how to run a sample CloudWatch Logs Insights query.

Prerequisites

- Existing log groups and log streams set up in CloudWatch Logs.
- Existing logs stored in CloudWatch Logs.

If you use services such as AWS CloudTrail, Amazon Route 53, or Amazon VPC, you've probably already set up logs from those services to go to CloudWatch Logs. For more information about sending logs to CloudWatch Logs, see Getting Started with CloudWatch Logs.

Queries in CloudWatch Logs Insights return either a set of fields from log events, or the result of a mathematical aggregation or other operation performed on log events. This section demonstrates a query that returns a list of log events.

To run a CloudWatch Logs Insights sample query

2. In the navigation pane, choose Insights.
3. The query editor near the top of the screen contains a default query that returns the 20 most recent log events. Above the query editor, select a log group to query.

   When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in Discovered fields in the right pane. It also displays a bar graph of log events in this log group over time. This bar graph shows the distribution of events in the log group that matches your query and time range, not just the events displayed in the table.

4. Choose Run query.

   The results of the query appear. In this example, the results are the most recent 20 log events of any type.

5. To see all of the fields for one of the returned log events, choose the arrow to the left of that log event.

For more information about how to run and modify CloudWatch Logs Insights queries, see Run and Modify a Sample Query.
Example Queries

This section contains CloudWatch Logs Insights example queries for analyzing Kinesis Data Analytics application logs. These queries search for several example error conditions, and serve as templates for writing queries that find other error conditions.

**Note**
Replace the Region (us-west-2), Account ID (012345678901) and application name (YourApplication) in the following query examples with your application's Region and your Account ID.

This topic contains the following sections:
- Analyze Operations: Distribution of Tasks (p. 124)
- Analyze Operations: Change in Parallelism (p. 124)
- Analyze Errors: Access Denied (p. 125)
- Analyze Errors: Source or Sink Not Found (p. 125)
- Analyze Errors: Application Task-Related Failures (p. 125)

### Analyze Operations: Distribution of Tasks

The following CloudWatch Logs Insights query returns the number of tasks the Apache Flink Job Manager distributes between Task Managers. You need to set the query's time frame to match one job run so that the query doesn't return tasks from previous jobs. For more information about Parallelism, see Scaling (p. 19).

```sql
fields @timestamp, message
| filter message like /Deploying/
| parse message " to flink-taskmanager-" as @tmid
| stats count(*) by @tmid
| sort @timestamp desc
| limit 2000
```

The following CloudWatch Logs Insights query returns the subtasks assigned to each Task Manager. The total number of subtasks is the sum of every task's parallelism. Task parallelism is derived from operator parallelism, and is the same as the application's parallelism by default, unless you change it in code by specifying `setParallelism`. For more information about setting operator parallelism, see Setting the Parallelism: Operator Level in the Apache Flink documentation.

```sql
fields @timestamp, @tmid, @subtask
| filter message like /Deploying/
| parse message "Deploying * to flink-taskmanager-" as @subtask, @tmid
| sort @timestamp desc
| limit 2000
```

For more information about task scheduling, see Jobs and Scheduling in the Apache Flink documentation.

### Analyze Operations: Change in Parallelism

The following CloudWatch Logs Insights query returns changes to an application's parallelism (for example, due to automatic scaling). This query also returns manual changes to the application's parallelism. For more information about automatic scaling, see the section called “Automatic Scaling” (p. 20).
fields @timestamp, @parallelism
| filter message like /property: parallelism.default, /
| parse message "default, ** as @parallelism
| sort @timestamp asc

Analyze Errors: Access Denied

The following CloudWatch Logs Insights query returns Access Denied logs.

fields @timestamp, @message, @messageType
| filter applicationARN like /arn:aws:kinesisanalytics:us-west-2:012345678901:application /YourApplication/
| filter @message like /AccessDenied/
| sort @timestamp desc

Analyze Errors: Source or Sink Not Found

The following CloudWatch Logs Insights query returns ResourceNotFound logs. ResourceNotFound logs result if a Kinesis source or sink is not found.

fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalytics:us-west-2:012345678901:application /YourApplication/
| filter @message like /ResourceNotFoundException/
| sort @timestamp desc

Analyze Errors: Application Task-Related Failures

The following CloudWatch Logs Insights query returns an application's task-related failure logs. These logs result if an application's state switches from RUNNING to FAILED.

fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalytics:us-west-2:012345678901:application /YourApplication/
| filter @message like /switched from RUNNING to FAILED/
| sort @timestamp desc

Viewing Amazon Kinesis Data Analytics Metrics and Dimensions

This topic contains the following sections:

- Application Metrics (p. 126)
- Kinesis Data Streams Connector Metrics (p. 128)
- Amazon MSK Connector Metrics (p. 128)
- Viewing CloudWatch Metrics (p. 129)
- Setting CloudWatch Metrics Reporting Levels (p. 130)

When your Kinesis Data Analytics for Java application processes a data source, Kinesis Data Analytics reports the following metrics and dimensions to Amazon CloudWatch.
# Application Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
<th>Level</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>downtime</td>
<td>Milliseconds</td>
<td>For jobs currently in a failing/recovering situation, the time elapsed during this outage.</td>
<td>Application</td>
<td>This metric measures the time elapsed while a job is failing or recovering. This metric returns 0 for running jobs and -1 for completed jobs. If this metric is not 0 or -1, this indicates that the Apache Flink job for the application failed to run.</td>
</tr>
<tr>
<td>lastCheckpointDuration</td>
<td>Milliseconds</td>
<td>The time it took to complete the last checkpoint</td>
<td>Application</td>
<td>This metric measures the time it took to complete the most recent checkpoint. If this metric is increasing in value, this may indicate that there is an issue with your application, such as a memory leak or bottleneck. In some cases, you can troubleshoot this issue by disabling checkpointing.</td>
</tr>
<tr>
<td>lastCheckpointSize</td>
<td>Bytes</td>
<td>The total size of the last checkpoint</td>
<td>Application</td>
<td>You can use this metric to determine running application storage utilization. Determine the application's storage utilization as follows:</td>
</tr>
</tbody>
</table>

126
<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
<th>Level</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>numRecordsIn</td>
<td>Count</td>
<td>The total number of records this operator or task has received.</td>
<td>Task</td>
<td></td>
</tr>
<tr>
<td>numRecordsInPerSecond</td>
<td>Count/Second</td>
<td>The total number of records this operator or task has received per second.</td>
<td>Task, Operator</td>
<td></td>
</tr>
<tr>
<td>numRecordsOut</td>
<td>Count</td>
<td>The total number of records this operator or task has emitted.</td>
<td>Task, Operator</td>
<td>You can use this metric to determine the total data sent by the task over time.</td>
</tr>
<tr>
<td>numRecordsOutPerSecond</td>
<td>Count/Second</td>
<td>The total number of records this operator or task has emitted per second.</td>
<td>Task, Operator</td>
<td></td>
</tr>
<tr>
<td>numLateRecordsDropped</td>
<td>Count</td>
<td>The number of records this operator or task has dropped due to arriving late.</td>
<td>Task, Operator</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Usage Notes:} \frac{(<\text{lastCheckpointSize}> + <\text{application's disk usage}>)}{(<\text{Number of KPs}> \times 50)}
\]

If this metric is increasing in value, this may indicate that there is an issue with your application, such as a memory leak or bottleneck.
### Kinesis Data Streams Connector Metrics

AWS emits all records for Kinesis Data Streams in addition to the following:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
<th>Level</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentInputWatermark</td>
<td>Milliseconds</td>
<td>The last watermark this operator/tasks has received</td>
<td>Task, Operator</td>
<td>This record is only emitted for operators or tasks with two inputs. This is the minimum value of the last received watermarks.</td>
</tr>
<tr>
<td>millisBehindLatest</td>
<td>Milliseconds</td>
<td>The number of milliseconds the consumer is behind the head of the stream, indicating how far behind current time the consumer is.</td>
<td>Stream, ShardId</td>
<td>• A value of 0 indicates that record processing is caught up, and there are no new records to process at this moment. A particular shard’s metric can be specified by stream name and shard id. • A value of -1 indicates that the service has not yet reported a value for the metric.</td>
</tr>
<tr>
<td>bytesRequestedPerFetch</td>
<td>Bytes</td>
<td>The bytes requested in a single call to getRecords.</td>
<td>Stream, ShardId</td>
<td></td>
</tr>
</tbody>
</table>

### Amazon MSK Connector Metrics

AWS emits all records for Amazon MSK in addition to the following:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
<th>Level</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentoffsets</td>
<td>N/A</td>
<td>The consumer’s current read</td>
<td>Topic, Partition</td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>Unit</td>
<td>Description</td>
<td>Level</td>
<td>Usage Notes</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>offset, for each partition. A particular partition's metric can be specified by topic name and partition id.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>committedoffsets</td>
<td>N/A</td>
<td>The last successfully committed offsets to Kafka, for each partition. A particular partition's metric can be specified by topic name and partition id.</td>
<td>Topic, Partition</td>
<td></td>
</tr>
<tr>
<td>records_lag_max</td>
<td>Count</td>
<td>The maximum lag in terms of number of records for any partition in this window</td>
<td>Operator</td>
<td></td>
</tr>
<tr>
<td>KafkaConsumer_records_lag_max</td>
<td>Count</td>
<td>The maximum lag in terms of number of records for a Kafka consumer connector for any partition in this window</td>
<td>Operator</td>
<td>This metric is deprecated, and will be removed from a future Apache Flink release.</td>
</tr>
<tr>
<td>bytes_consumed_rate</td>
<td>Bytes</td>
<td>The average number of bytes consumed per second for a topic</td>
<td>Operator</td>
<td></td>
</tr>
<tr>
<td>KafkaConsumer_bytes_consumed_rate</td>
<td>Bytes</td>
<td>The average number of bytes consumed per second for a Kafka consumer connector for a topic</td>
<td>Operator</td>
<td>This metric is deprecated, and will be removed from a future Apache Flink release.</td>
</tr>
</tbody>
</table>

### Viewing CloudWatch Metrics

You can view CloudWatch metrics for your application using the Amazon CloudWatch console or the AWS CLI.

**To view metrics using the CloudWatch console**

2. In the navigation pane, choose **Metrics**.
3. In the **CloudWatch Metrics by Category** pane for Amazon Kinesis Data Analytics, 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.

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

**Setting CloudWatch Metrics Reporting Levels**

You can control the level of application metrics that your application creates. Kinesis Data Analytics for Java Applications supports the following metrics levels:

- **Application**: The application only reports the highest level of metrics for each application.
- **Task**: The application reports task-specific throughput metrics, such as number of records in and out of the application per second.
- **Operator**: The application reports operator-level metrics, such as metrics for each filter or map operation.
- **Parallelism**: The application reports Task and Operator level metrics for each execution thread. This reporting level is not recommended for applications with a Parallelism setting above 64 due to excessive costs.

The default level is **Application**. The application reports metrics at the current level and all higher levels. For example, if the reporting level is set to **Operator**, the application reports Application, Task, and Operator metrics.

You set the CloudWatch metrics reporting level using the `MonitoringConfiguration` parameter of the **CreateApplication** action, or the `MonitoringConfigurationUpdate` parameter of the **UpdateApplication** action. The following example request for the **UpdateApplication** action sets the CloudWatch metrics reporting level to **Task**:

```json
{
    "ApplicationName": "MyApplication",
    "CurrentApplicationVersionId": 4,
    "ApplicationConfigurationUpdate": {
        "FlinkApplicationConfigurationUpdate": {
            "MonitoringConfigurationUpdate": {
                "ConfigurationTypeUpdate": "CUSTOM",
                "MetricsLevelUpdate": "TASK"
            }
        }
    }
}
```

You can also configure the logging level using the `LogLevel` parameter of the **CreateApplication** action or the `LogLevelUpdate` parameter of the **UpdateApplication** action. You can use the following log levels:

- **ERROR**: Logs potentially recoverable error events.
- **WARN**: Logs warning events that might lead to an error.
INFO: Logs informational events.
DEBUG: Logs general debugging events.

For more information about Log4j logging levels, see Level in the Apache Log4j documentation.

Writing Custom Messages to CloudWatch Logs

You can write custom messages to your Amazon Kinesis Data Analytics application's CloudWatch log. You do this by using the Apache log4j library or the Simple Logging Facade for Java (SLF4J) library.

Topics
• Write to CloudWatch Logs Using Log4J (p. 131)
• Write to CloudWatch Logs Using SLF4J (p. 132)

Write to CloudWatch Logs Using Log4J

1. Add the following repository to your application's pom.xml file:

```xml
<repository>
  <id>log4j</id>
  <url>https://mvnrepository.com/artifact/log4j/log4j</url>
</repository>
```

2. Add the following dependencies to your application's pom.xml file:

```xml
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
  <scope>compile</scope>
</dependency>
```

3. Include the object from the library:

```java
import org.apache.logging.log4j.Logger;
```

4. Instantiate the Logger object, passing in your application class:

```java
private static final Logger log = Logger.getLogger(YourApplicationClass.class);
```

5. Write to the log using log.info. A large number of messages are written to the application log. To make your custom messages easier to filter, use the INFO application log level.

```java
log.info("This message will be written to the application's CloudWatch log");
```

The application writes a record to the log with a message similar to the following:

```
{
  "locationInformation":
  "com.amazonaws.services.kinesisanalytics.streamingjob.main(StreamingJob.java:95)",
  "logger": "com.amazonaws.services.kinesisanalytics.streamingjob",
```
Write to CloudWatch Logs Using SLF4J

1. Add the following dependency to your application's `pom.xml` file:

   ```xml
   <dependency>
     <groupId>org.slf4j</groupId>
     <artifactId>slf4j-log4j12</artifactId>
     <version>1.7.7</version>
     <scope>runtime</scope>
   </dependency>
   
   2. Include the objects from the library:

   ```java
   import org.slf4j.Logger;
   import org.slf4j.LoggerFactory;
   ```

   3. Instantiate the `Logger` object, passing in your application class:

   ```java
   private static final Logger log = LoggerFactory.getLogger(YourApplicationClass.class);
   ```

   4. Write to the log using `log.info`. A large number of messages are written to the application log. To make your custom messages easier to filter, use the INFO application log level.

   ```java
   log.info("This message will be written to the application's CloudWatch log");
   ```

The application writes a record to the log with a message similar to the following:

```json
{
   "locationInformation": "com.amazonaws.services.kinesisanalytics.StreamingJob.main(StreamingJob.java:95)",
   "logger": "com.amazonaws.services.kinesisanalytics.StreamingJob",
   "message": "This message will be written to the application's CloudWatch log",
   "threadName": "Flink-DispatcherRestEndpoint-thread-2",
   "applicationARN": "arn:aws:kinesisanalytics:us-east-1:123456789012:application/test",
   "applicationVersionId": "1", "messageSchemaVersion": "1",
   "messageType": "INFO"
}
```
Kinesis Data Analytics Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Kinesis Data Analytics, 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 Kinesis Data Analytics, 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 AWS Regions. 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 Kinesis Data Analytics actions are logged by CloudTrail and are documented in the Kinesis Data Analytics 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 root or AWS Identity and Access Management (IAM) 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 Kinesis Data Analytics 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.

```json
{
    "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": "2019-03-07T01:19:47Z",
    "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": {
        "applicationName": "cloudtrail-test",
        "currentApplicationVersionId": 1,
        "cloudWatchLoggingOption": {
        }
    },
    "responseElements": {
        "cloudWatchLoggingOptionDescriptions": [
            {
                "cloudWatchLoggingOptionId": "2.1",
            }
        ],
        "applicationVersionId": 2,
        "applicationARN": "arn:aws:kinesisanalytics:us-east-1:012345678910:application/cloudtrail-test"
    },
    "requestID": "18dfb315-4077-11e9-afd3-67f7af21e34f",
    "eventID": "d3c9e467-db1d-4cab-a628-c21258385124",
    "eventType": "AwsApiCall",
    "apiVersion": "2018-05-23",
    "recipientAccountId": "012345678910"
},
{
    "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": "2019-03-12T02:40:48Z",
    "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": "sample-app"
    },
    "responseElements": null,
    "requestID": "3e82dc3e-4470-11e9-9d01-e8994e9a3ca",
    "eventID": "90ffe8e4-9e47-48c9-84e1-4f2d427d98a5",
    "eventType": "AwsApiCall",
    "apiVersion": "2018-05-23"}
"recipientAccountId": "012345678910"
Kinesis Data Analytics for Java Applications Limits

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

- You can create up to 50 Kinesis Data Analytics applications per 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.

  For a list of regions that support Kinesis Data Analytics, see Kinesis Data Analytics Regions and Endpoints.

- The number of Kinesis processing units (KPU) is limited to 32 by default. For instructions on how to request an increase to this limit, see To request a limit increase in AWS Service Limits.

  With Kinesis Data Analytics, your AWS account is charged for allocated resources, rather than resources that your application uses. You are charged an hourly rate based on the maximum 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. For each KPU, the service also provisions 50 GB of running application storage.

- You can create up to 1,000 Kinesis Data Analytics Snapshots (p. 16) per application.

- You can assign up to 50 tags per application.

- The maximum size for an application JAR file is 512 MB. If you exceed this limit, your application will fail to start.
Best Practices for Kinesis Data Analytics for Java Applications

This section contains information and recommendations for developing stable, performant Amazon Kinesis Data Analytics applications.

Topics
- Fault Tolerance: Checkpoints and Savepoints (p. 137)
- Performance and Parallelism (p. 137)
- Logging (p. 138)
- Coding (p. 138)

Fault Tolerance: Checkpoints and Savepoints

Use checkpoints and savepoints to implement fault tolerance in your Kinesis Data Analytics for Java Applications application. Keep the following in mind when developing and maintaining your application:

- We recommend that you leave checkpointing enabled for your application. Checkpointing provides fault tolerance for your application in case of unexpected failures due to service issues, application dependency failures, and other issues.
- Set ApplicationSnapshotConfiguration::SnapshotsEnabled to false during application development or troubleshooting. A snapshot is created during every application stop, which may cause issues if the application is in an unhealthy state or isn't performant. Set SnapshotsEnabled to true after the application is in production and is stable.

For your application to restart properly with correct state data, we recommend that your application create a snapshot every 5-10 minutes.

For information about monitoring application downtime, see Metrics and Dimensions (p. 125).

For more information about implementing fault tolerance, see Fault Tolerance (p. 13).

Performance and Parallelism

Your application can scale to meet any throughput level by tuning your application parallelism, and avoiding performance pitfalls. Keep the following in mind when developing and maintaining your application:

- Verify that all of your application sources and sinks are sufficiently provisioned and are not being throttled. If the sources and sinks are other AWS services, monitor those services using CloudWatch.
- For applications with very high parallelism, check if the high levels of parallelism are applied to all operators in the application. By default, Apache Flink applies the same application parallelism for all operators in the application graph. This can lead to either provisioning issues on sources or sinks, or bottlenecks in operator data processing. You can change the parallelism of each operator in code with setParallelism.


• Understand the meaning of the parallelism settings for the operators in your application. If you change the parallelism for an operator, you may not be able to restore the application from a snapshot created when the operator had a parallelism that is incompatible with the current settings. For more information about setting operator parallelism, see Set maximum parallelism for operators explicitly.

For more information about implementing scaling, see Scaling (p. 19).

Logging

You can monitor your application's performance and error conditions using CloudWatch Logs. Keep the following in mind when configuring logging for your application:

• Enable CloudWatch logging for the application so that any runtime issues can be debugged.
• Do not create a log entry for every record being processed in the application. This causes severe bottlenecks during processing and might lead to backpressure in processing of data.

For more information about implementing logging, see Logging and Monitoring (p. 116).

Coding

You can make your application performant and stable by using recommended programming practices. Keep the following in mind when writing application code:

• Do not use \texttt{system.exit()} in your application code, in either your application's \texttt{main} method or in user-defined functions. If you want to shut down your application from within code, throw an exception derived from \texttt{Exception} or \texttt{RuntimeException}, containing a message about what went wrong with the application.

Note the following about how the service handles this exception:

• If the exception is thrown from your application's \texttt{main} method, the service will wrap it in a \texttt{ProgramInvocationException} when the application transitions to the \texttt{RUNNING} state, and the job manager will fail to submit the job.
• If the exception is thrown from a user-defined function, the job manager will fail the job and restart it, and details of the exception will be written to the exception log.
• Consider shading your application JAR file and its included dependencies. Shading is recommended when there are potential conflicts in package names between your application and the Apache Flink runtime. If a conflict occurs, your application logs may contain an exception of type \texttt{java.util.concurrent.ExecutionException}. For more information about shading your application JAR file, see Apache Maven Shade Plugin.
Earlier Version Information for Kinesis Data Analytics for Java Applications

This topic contains information about using Kinesis Data Analytics for Java Applications with older versions of Apache Flink. The versions of Apache Flink that Kinesis Data Analytics supports are 1.8.2 (recommended) and 1.6.2.

This topic contains the following sections:

- Building Applications with Apache Flink 1.6.2 (p. 139)
- Upgrading Applications from Apache Flink 1.6.2 to 1.8.2 (p. 140)
- Available Connectors in Apache Flink 1.6.2 (p. 140)
- Getting Started: Flink 1.6.2 (p. 140)

Building Applications with Apache Flink 1.6.2

This section contains information about components that you use for building Kinesis Data Analytics applications that work with Apache Flink 1.6.2.

Use the following component versions for Kinesis Data Analytics applications:

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.8 (recommended)</td>
</tr>
<tr>
<td>AWS Java SDK</td>
<td>1.11.379</td>
</tr>
<tr>
<td>Apache Flink</td>
<td>1.6.2</td>
</tr>
<tr>
<td>Kinesis Data Analytics for Flink Runtime (aws-kinesisanalytics-runtime)</td>
<td>1.0.1</td>
</tr>
<tr>
<td>Kinesis Data Analytics Flink Connectors (aws-kinesisanalytics-flink)</td>
<td>1.0.1</td>
</tr>
<tr>
<td>Apache Maven</td>
<td>3.1</td>
</tr>
</tbody>
</table>

Note
When using Kinesis Data Analytics Runtime version 1.0.1, you specify the version of Apache Flink in your pom.xml file rather than using the -Dflink.version parameter when compiling your application code.

For an example of a pom.xml file for a Kinesis Data Analytics application that uses Apache Flink version 1.6.2, see the Kinesis Data Analytics for Flink 1.6.2 Getting Started Application.

For information about how to build and use application code for a Kinesis Data Analytics application, see Creating Applications (p. 2).
Upgrading Applications from Apache Flink 1.6.2 to 1.8.2

To upgrade an Kinesis Data Analytics application from Apache Flink 1.6.2 to 1.8.2, do the following:

- Change the versions of the Kinesis Data Analytics Runtime and Kinesis Data Analytics Flink connectors (aws-kinesisanalytics-flink) in your application's pom.xml file to 1.1.0.
- Remove the flink.version property from your application's pom.xml file. You will provide this parameter when you compile the application code in the next step.
- Recompile your application code using the following command:

  ```bash
  mvn package -Dflink.version=1.8.2
  ```

- Delete your existing application. Create your application again, and choose Apache Flink 1.8 (Recommended version) for the application's Runtime.

Available Connectors in Apache Flink 1.6.2

The Apache Flink framework contains connectors for accessing data from a variety of sources. For information about connectors available in the Apache Flink 1.6.2 framework, see Connectors in the Apache Flink documentation.

Getting Started: Flink 1.6.2

This topic contains a version of the Getting Started (p. 24) Tutorial that uses Flink 1.6.2.

Topics

- Components of a Kinesis Data Analytics for Java Application (p. 140)
- Prerequisites for Completing the Exercises (p. 141)
- Step 1: Set Up an AWS Account and Create an Administrator User (p. 141)
- Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 143)
- Step 3: Create and Run a Kinesis Data Analytics for Java Application (p. 144)
- Step 4: Clean Up AWS Resources (p. 158)

Components of a Kinesis Data Analytics for Java Application

To process data, your Kinesis Data Analytics application uses a Java/Apache Maven application that processes input and produces output using the Apache Flink runtime.

A Kinesis Data Analytics for Java application has the following components:

- **Runtime properties:** You can use runtime properties to configure your application without recompiling your application code.
- **Source:** The application consumes data by using a source. A source connector reads data from a Kinesis data stream, an Amazon S3 bucket, etc. For more information, see Sources (p. 6).
- **Operators:** The application processes data by using one or more operators. An operator can transform, enrich, or aggregate data. For more information, see Operators (p. 9).
- **Sink:** The application produces data to external sources by using sinks. A sink connector writes data to a Kinesis data stream, a Kinesis Data Firehose delivery stream, an Amazon S3 bucket, etc. For more information, see Sinks (p. 6).

After you create, compile, and package your Java application, you upload the code package to an Amazon Simple Storage Service (Amazon S3) bucket. You then create a Kinesis Data Analytics application. You pass in the code package location, a Kinesis data stream as the streaming data source, and typically a streaming or file location that receives the application's processed data.

**Prerequisites for Completing the Exercises**

To complete the steps in this guide, you must have the following:

- **Java Development Kit** (JDK) version 8. Set the `JAVA_HOME` environment variable to point to your JDK install location.
- We recommend that you use a development environment (such as Eclipse Java Neon or IntelliJ Idea) to develop and compile your application.
- **Git Client.** Install the Git client if you haven't already.
- **Apache Maven Compiler Plugin.** Maven must be in your working path. To test your Apache Maven installation, enter the following:

  ```
  $ mvn --version
  ```

**Note**

Kinesis Data Analytics for Java Applications only supports Java applications that are built with Apache Maven.

To get started, go to Step 1: Set Up an AWS Account and Create an Administrator User (p. 141).

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

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

1. **Sign Up for AWS** (p. 141)
2. **Create an IAM User** (p. 142)

**Sign Up for AWS**

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

With Kinesis Data Analytics, you pay only for the resources that 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 Tier.

If you already have an AWS account, skip to the next task. If you don't have an AWS account, follow these steps to create one.

**To create an AWS account**

2. Follow the online instructions.

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

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

**Create an IAM User**

Services in AWS, such as Amazon Kinesis Data Analytics, require that you provide credentials when you access them. This is so that the service can determine whether you have permissions to access the resources that are owned by that service. The AWS Management Console requires that you enter your password.

You can create access keys for your AWS account to access the AWS Command Line Interface (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 permissions. Follow the procedure to create adminuser in your account.

**To create a group for administrators**

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Groups, and then choose Create New Group.
3. For Group Name, enter a name for your group, such as Administrators, and then choose Next Step.
4. In the list of policies, select the check box next to the AdministratorAccess policy. You can use the Filter menu and the Search box to filter the list of policies.
5. Choose Next Step, and then choose Create Group.

Your new group is listed under Group Name.

**To create an IAM user for yourself, add it to the Administrators group, and create a password**

1. In the navigation pane, choose Users, and then choose Add user.
2. In the User name box, enter a user name.
3. Choose both Programmatic access and AWS Management Console access.
4. Choose Next: Permissions.
5. Select the check box next to the Administrators group. Then choose Next: Review.
6. Choose Create user.

**To sign in as the new IAM user**

1. Sign out of the AWS Management Console.
2. Use the following URL format to sign in to the console:
https://aws_account_number.signin.aws.amazon.com/console/

The **aws_account_number** is your AWS account ID without any hyphens. For example, if your AWS account ID is 1234-5678-9012, replace **aws_account_number** with **123456789012**. For information about how to find your account number, see Your AWS Account ID and Its Alias in the IAM User Guide.

3. Enter the IAM user name and password that you just created. When you're signed in, the navigation bar displays **your_user_name @ your_aws_account_id**.

**Note**
If you don’t want the URL for your sign-in page to contain your AWS account ID, you can create an account alias.

**To create or remove an account alias**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. On the navigation pane, choose Dashboard.
3. Find the IAM users sign-in link.
4. To create the alias, choose Customize. Enter the name you want to use for your alias, and then choose Yes, Create.
5. To remove the alias, choose Customize, and then choose Yes, Delete. The sign-in URL reverts to using your AWS account ID.

To sign in after you create an account alias, use the following URL:

https://your_account_alias.signin.aws.amazon.com/console/

To verify the sign-in link for IAM users for your account, open the IAM console and check under IAM users sign-in link on the dashboard.

For more information about IAM, see the following:

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

**Next Step**

Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 143)

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

In this step, you download and configure the AWS CLI to use with Amazon Kinesis Data Analytics for Java Applications.

**Note**
The getting started exercises in this guide assume that you are using administrator credentials (adminuser) in your account to perform the operations.

**Note**
If you already have the AWS CLI installed, you might need to upgrade to get the latest functionality. For more information, see Installing the AWS Command Line Interface in the
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:
   - Installing the AWS Command Line Interface
   - Configuring the AWS CLI
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 AWS Regions and Endpoints in the Amazon Web Services General Reference.

   **Note**
   The example code and commands in this tutorial use the US West (Oregon) Region. To use a different AWS Region, change the Region in the code and commands for this tutorial to the Region you want to use.

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

   ```
   aws help
   ```

After you set up an AWS account and the AWS CLI, you can try the next exercise, in which you configure a sample application and test the end-to-end setup.

**Next Step**

**Step 3: Create and Run a Kinesis Data Analytics for Java Application (p. 144)**

**Step 3: Create and Run a Kinesis Data Analytics for Java Application**

In this exercise, you create a Kinesis Data Analytics for Java application with data streams as a source and a sink.

This section contains the following steps:

- Create Two Amazon Kinesis Data Streams (p. 145)
- Write Sample Records to the Input Stream (p. 145)
- Download and Examine the Apache Flink Streaming Java Code (p. 146)
Create Two Amazon Kinesis Data Streams

Before you create a Kinesis Data Analytics for Java application for this exercise, create two Kinesis data streams (ExampleInputStream and ExampleOutputStream). Your application uses these streams for the application source and destination streams.

You can create these streams using either the Amazon Kinesis console or the following AWS CLI command. For console instructions, see Creating and Updating Data Streams in the Amazon Kinesis Data Streams Developer Guide.

To create the data streams (AWS CLI)

1. To create the first stream (ExampleInputStream), use the following Amazon Kinesis create-stream AWS CLI command.

   ```
   $ aws kinesis create-stream \
   --stream-name ExampleInputStream \
   --shard-count 1 \
   --region us-west-2 \
   --profile adminuser
   ```

2. To create the second stream that the application uses to write output, run the same command, changing the stream name to ExampleOutputStream.

   ```
   $ aws kinesis create-stream \
   --stream-name ExampleOutputStream \
   --shard-count 1 \
   --region us-west-2 \
   --profile adminuser
   ```

Write Sample Records to the Input Stream

In this section, you use a Python script to write sample records to the stream for the application to process.

Note
This section requires the AWS SDK for Python (Boto).

1. Create a file named stock.py with the following contents:

   ```python
   import json
   import boto3
   import random
   import datetime
   kinesis = boto3.client('kinesis')
   def getReferrer():
       data = {}
       now = datetime.datetime.now()
       str_now = now.isoformat()
       data['EVENT_TIME'] = str_now
       data['TICKER'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
   ```
Step 3: Create an Application

```python
    price = random.random() * 100
    data['PRICE'] = round(price, 2)
    return data

    while True:
        data = json.dumps(getReferrer())
        print(data)
        kinesis.put_record(
            StreamName="ExampleInputStream",
            Data=data,
            PartitionKey="partitionkey")
```

2. Later in the tutorial, you run the `stock.py` script to send data to the application.

```bash
    # python stock.py
```

Download and Examine the Apache Flink Streaming Java Code

The Java application code for this example is available from GitHub. To download the application code, do the following:

1. Clone the remote repository using the following command:

    ```bash
    git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-examples
    ```

2. Navigate to the `amazon-kinesis-data-analytics-java-examples/GettingStarted_1_6` directory.

Note the following about the application code:

- A **Project Object Model (pom.xml)** file contains information about the application's configuration and dependencies, including the Kinesis Data Analytics for Java Applications libraries.
- The **BasicStreamingJob.java** file contains the `main` method that defines the application's functionality.
- The application uses a Kinesis source to read from the source stream. The following snippet creates the Kinesis source:

    ```java
    return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
    new SimpleStringSchema(), inputProperties));
    ```

- Your application creates source and sink connectors to access external resources using a `StreamExecutionEnvironment` object.
- The application creates source and sink connectors using static properties. To use dynamic application properties, use the `createSourceFromApplicationProperties` and `createSinkFromApplicationProperties` methods to create the connectors. These methods read the application's properties to configure the connectors.

For more information about runtime properties, see **Runtime Properties (p. 10)**.

Compile the Application Code

In this section, you use the Apache Maven compiler to create the Java code for the application. For information about installing Apache Maven and the Java Development Kit (JDK), see **Prerequisites for Completing the Exercises (p. 141)**.
Note
In order to use the Kinesis connector for the following application, you need to download the source code for the connector and build it as described in the Apache Flink documentation.

To compile the application code

1. To use your application code, you compile and package it into a JAR file. You can compile and package your code in one of two ways:
   - Use the command-line Maven tool. Create your JAR file by running the following command in the directory that contains the pom.xml file:
     
     ```
     mvn package
     ```
   
   **Note**
   The -Dflink.version parameter is not required for Kinesis Data Analytics Runtime version 1.0.1; it is only required for version 1.1.0 and later. For more information, see the section called “Specifying your Application's Apache Flink Version” (p. 3).
   
   • Use your development environment. See your development environment documentation for details.
   
   You can either upload your package as a JAR file, or you can compress your package and upload it as a ZIP file. If you create your application using the AWS CLI, you specify your code content type (JAR or ZIP).
   
2. If there are errors while compiling, verify that your JAVA_HOME environment variable is correctly set.

If the application compiles successfully, the following file is created:

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Upload the Apache Flink Streaming Java Code

In this section, you create an Amazon Simple Storage Service (Amazon S3) bucket and upload your application code.

To upload the application code

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose Create bucket.
3. Enter `ka-app-code-<username>` in the Bucket name field. Add a suffix to the bucket name, such as your user name, to make it globally unique. Choose Next.
4. In the Configure options step, keep the settings as they are, and choose Next.
5. In the Set permissions step, keep the settings as they are, and choose Next.
6. Choose Create bucket.
7. In the Amazon S3 console, choose the `ka-app-code-<username>` bucket, and choose Upload.
8. In the Select files step, choose Add files. Navigate to the `aws-kinesis-analytics-java-apps-1.0.jar` file that you created in the previous step. Choose Next.
9. In the Set permissions step, keep the settings as they are. Choose Next.
10. In the Set properties step, keep the settings as they are. Choose Upload.

Your application code is now stored in an Amazon S3 bucket where your application can access it.
Create and Run the Kinesis Data Analytics Application

You can create and run a Kinesis Data Analytics for Java application using either the console or the AWS CLI.

**Note**
When you create the application using the console, your AWS Identity and Access Management (IAM) and Amazon CloudWatch Logs resources are created for you. When you create the application using the AWS CLI, you create these resources separately.

**Topics**
- Create and Run the Application (Console) (p. 148)
- Create and Run the Application (AWS CLI) (p. 153)

Create and Run the Application (Console)

Follow these steps to create, configure, update, and run the application using the console.

**Create the Application**

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. On the Amazon Kinesis Data Analytics dashboard, choose Create analytics application.
3. On the Kinesis Analytics - Create application page, provide the application details as follows:
   - For Application name, enter MyApplication.
   - For Description, enter My java test app.
   - For Runtime, choose Apache Flink.

   **Note**
   Kinesis Data Analytics for Java Applications uses Apache Flink version 1.8.2 or 1.6.2.
   - Change the version pulldown to Apache Flink 1.6.
5. Choose **Create application**.

**Note**

When you create a Kinesis Data Analytics for Java application using the console, you have the option of having an IAM role and policy created for your application. Your application uses this role and policy to access its dependent resources. These IAM resources are named using your application name and Region as follows:
Step 3: Create an Application

- Policy: kinesis-analytics-service-MyApplication-us-west-2
- Role: kinesis-analytics-MyApplication-us-west-2

**Edit the IAM Policy**

Edit the IAM policy to add permissions to access the Kinesis data streams.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Policies. Choose the kinesis-analytics-service-MyApplication-us-west-2 policy that the console created for you in the previous section.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (012345678901) with your account ID.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-started-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
            ]
        }
    ]
}
```
Configure the Application

1. On the **MyApplication** page, choose **Configure**.

2. On the **Configure application** page, provide the **Code location**:
   - For **Amazon S3 bucket**, enter `ka-app-code-<username>`.
   - For **Path to Amazon S3 object**, enter `java-getting-started-1.0.jar`.

3. Under **Access to application resources**, for **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
4. Under **Properties**, for **Group ID**, enter **ProducerConfigProperties**.
5. Enter the following application properties and values:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>flink.inputstream.initpos</td>
<td>LATEST</td>
</tr>
<tr>
<td>aws:region</td>
<td>us-west-2</td>
</tr>
<tr>
<td>AggregationEnabled</td>
<td>false</td>
</tr>
</tbody>
</table>

6. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
7. For **CloudWatch logging**, select the **Enable** check box.
8. Choose **Update**.
Note
When you choose to enable Amazon CloudWatch logging, Kinesis Data Analytics creates a log group and log stream for you. The names of these resources are as follows:

- Log group: /aws/kinesis-analytics/MyApplication
- Log stream: kinesis-analytics-log-stream

Run the Application


2. When the application is running, refresh the page. The console shows the Application graph.
Stop the Application

On the MyApplication page, choose Stop. Confirm the action.

Update the Application

Using the console, you can update application settings such as application properties, monitoring settings, and the location or file name of the application JAR. You can also reload the application JAR from the Amazon S3 bucket if you need to update the application code.

On the MyApplication page, choose Configure. Update the application settings and choose Update.

Create and Run the Application (AWS CLI)

In this section, you use the AWS CLI to create and run the Kinesis Data Analytics application. Kinesis Data Analytics for Java Applications uses the kinesisanalyticsv2 AWS CLI command to create and interact with Kinesis Data Analytics applications.

Create a Permissions Policy

First, you create a permissions policy with two statements: one that grants permissions for the read action on the source stream, and another that grants permissions for write actions on the sink stream. You then attach the policy to an IAM role (which you create in the next section). Thus, when Kinesis Data Analytics assumes the role, the service has the necessary permissions to read from the source stream and write to the sink stream.

Use the following code to create the KAReadSourceStreamWriteSinkStream permissions policy. Replace username with the user name that you used to create the Amazon S3 bucket to store the application code. Replace the account ID in the Amazon Resource Names (ARNs) (012345678901) with your account ID.

```json
{
    "Version": "2012-10-17",
    "Statement": [
```
Step 3: Create an Application

```json
{
  "Sid": "S3",
  "Effect": "Allow",
  "Action": [
    "s3:GetObject",
    "s3:GetObjectVersion"
  ],
  "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*" ]
},
{
  "Sid": "ReadInputStream",
  "Effect": "Allow",
  "Action": "kinesis:*",
},
{
  "Sid": "WriteOutputStream",
  "Effect": "Allow",
  "Action": "kinesis:*",
}
}
```

For step-by-step instructions to create a permissions policy, see Tutorial: Create and Attach Your First Customer Managed Policy in the IAM User Guide.

**Note**

To access other AWS services, you can use the AWS SDK for Java. Kinesis Data Analytics automatically sets the credentials required by the SDK to those of the service execution IAM role that is associated with your application. No additional steps are needed.

Create an IAM Role

In this section, you create an IAM role that the Kinesis Data Analytics for Java application can assume to read a source stream and write to the sink stream.

Kinesis Data Analytics cannot access your stream without permissions. You grant these permissions via an IAM role. Each IAM role has two policies attached. The trust policy grants Kinesis Data Analytics permission to assume the role, and the permissions policy determines what Kinesis Data Analytics can do after assuming the role.

You attach the permissions policy that you created in the preceding section to this role.

**To create an IAM role**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles, Create Role.
3. Under Select type of trusted identity, choose AWS Service. Under Choose the service that will use this role, choose Kinesis. Under Select your use case, choose Kinesis Analytics.
   
   Choose Next: Permissions.
4. On the Attach permissions policies page, choose Next: Review. You attach permissions policies after you create the role.
5. On the Create role page, enter **KA-stream-rw-role** for the Role name. Choose Create role.

   Now you have created a new IAM role called KA-stream-rw-role. Next, you update the trust and permissions policies for the role.
6. Attach the permissions policy to the role.

**Note**
For this exercise, Kinesis Data Analytics assumes this role for both reading data from a Kinesis data stream (source) and writing output to another Kinesis data stream. So you attach the policy that you created in the previous step, the section called “Create a Permissions Policy” (p. 153).

a. On the **Summary** page, choose the **Permissions** tab.
b. Choose **Attach Policies**.
c. In the search box, enter **KAReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
d. Choose the **KAReadInputStreamWriteOutputStream** policy, and choose **Attach policy**.

You now have created the service execution role that your application uses to access resources. Make a note of the ARN of the new role.

For step-by-step instructions for creating a role, see **Creating an IAM Role (Console)** in the **IAM User Guide**.

**Create the Kinesis Data Analytics Application**

1. Save the following JSON code to a file named `create_request.json`. Replace the sample role ARN with the ARN for the role that you created previously. Replace the bucket ARN suffix (`username`) with the suffix that you chose in the previous section. Replace the sample account ID (012345678901) in the service execution role with your account ID.

```json
{
  "ApplicationName": "test",
  "ApplicationDescription": "my java test app",
  "RuntimeEnvironment": "FLINK-1_6",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/KA-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "java-getting-started-1.0.jar"
        },
        "CodeContentType": "ZIPFILE"
      },
      "EnvironmentProperties": {
        "PropertyGroups": [
          {
            "PropertyGroupId": "ProducerConfigProperties",
            "PropertyMap": {
              "flink.stream.initpos": "LATEST",
              "aws.region": "us-west-2",
              "AggregationEnabled": "false"
            }
          },
          {
            "PropertyGroupId": "ConsumerConfigProperties",
            "PropertyMap": {
              "aws.region": "us-west-2"
            }
          }
        ]
      }
    }
  }
}
```
2. Execute the `CreateApplication` action with the preceding request to create the application:

```bash
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

The application is now created. You start the application in the next step.

**Start the Application**

In this section, you use the `StartApplication` action to start the application.

**To start the application**

1. Save the following JSON code to a file named `start_request.json`.

```json
{
  "ApplicationName": "test",
  "RunConfiguration": {
    "ApplicationRestoreConfiguration": {
      "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
    }
  }
}
```

2. Execute the `StartApplication` action with the preceding request to start the application:

```bash
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

The application is now running. You can check the Kinesis Data Analytics metrics on the Amazon CloudWatch console to verify that the application is working.

**Stop the Application**

In this section, you use the `StopApplication` action to stop the application.

**To stop the application**

1. Save the following JSON code to a file named `stop_request.json`.

```json
{"ApplicationName": "test"}
```

2. Execute the `StopApplication` action with the following request to stop the application:

```bash
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

The application is now stopped.

**Add a CloudWatch Logging Option**

You can use the AWS CLI to add an Amazon CloudWatch log stream to your application. For information about using CloudWatch Logs with your application, see the section called “Setting Up Logging” (p. 116).
Update Environment Properties

In this section, you use the `UpdateApplication` action to change the environment properties for the application without recompiling the application code. In this example, you change the AWS Region of the source and destination streams.

To update environment properties for the application

1. Save the following JSON code to a file named `update_properties_request.json`.

   ```json
   {
   "ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
   "EnvironmentPropertyUpdates": {
   "PropertyGroups": [
   {
   "PropertyGroupId": "ProducerConfigProperties",
   "PropertyMap": {
   "flink.stream.initpos": "LATEST",
   "aws.region": "us-west-2",
   "AggregationEnabled": "false"
   }
   },
   {
   "PropertyGroupId": "ConsumerConfigProperties",
   "PropertyMap": {
   "aws.region": "us-west-2"
   }
   }
   ]
   }
   }
   }
   }
   }
   ```

2. Execute the `UpdateApplication` action with the preceding request to update environment properties:

   ```bash
   aws kinesisanalyticsv2 update-application --cli-input-json file://update_properties_request.json
   ```

Update the Application Code

When you need to update your application code with a new version of your code package, you use the `UpdateApplication` AWS CLI action.

To use the AWS CLI, delete your previous code package from your Amazon S3 bucket, upload the new version, and call `UpdateApplication`, specifying the same Amazon S3 bucket and object name. The application will restart with the new code package.

The following sample request for the `UpdateApplication` action reloads the application code and restarts the application. Update the `CurrentApplicationVersionId` to the current application version. You can check the current application version using the `ListApplications` or `DescribeApplication` actions. Update the bucket name suffix (<username>) with the suffix that you chose in the section called “Create Two Amazon Kinesis Data Streams” (p. 145) section.

```json
{
   "ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
   "ApplicationCodeConfigurationUpdate": {
   ```
Step 4: Clean Up AWS Resources

This section includes procedures for cleaning up AWS resources created in the Getting Started tutorial.

This topic contains the following sections:
- Delete Your Kinesis Data Analytics Application (p. 158)
- Delete Your Kinesis Data Streams (p. 158)
- Delete Your Amazon S3 Object and Bucket (p. 158)
- Delete Your IAM Resources (p. 158)
- Delete Your CloudWatch Resources (p. 159)

Delete Your Kinesis Data Analytics Application

2. In the Kinesis Data Analytics panel, choose MyApplication.
3. Choose Configure.
4. In the Snapshots section, choose Disable and then choose Update.
5. In the application's page, choose Delete and then confirm the deletion.

Delete Your Kinesis Data Streams

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. In the Kinesis Data Streams panel, choose ExampleInputStream.
3. In the ExampleInputStream page, choose Delete Kinesis Stream and then confirm the deletion.
4. In the Kinesis streams page, choose the ExampleOutputStream, choose Actions, choose Delete, and then confirm the deletion.

Delete Your Amazon S3 Object and Bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the ka-app-code-<username> bucket.
3. Choose Delete and then enter the bucket name to confirm deletion.

Delete Your IAM Resources

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation bar, choose Policies.
3. In the filter control, enter kinesis.
4. Choose the `kinesis-analytics-service-MyApplication-<your-region>` policy.
5. Choose Policy Actions and then choose Delete.
6. In the navigation bar, choose Roles.
7. Choose the `kinesis-analytics-MyApplication-<your-region>` role.
8. Choose Delete role and then confirm the deletion.

Delete Your CloudWatch Resources

2. In the navigation bar, choose Logs.
3. Choose the `/aws/kinesis-analytics/MyApplication` log group.
4. Choose Delete Log Group and then confirm the deletion.
Apache Flink Settings

Kinesis Data Analytics for Java Applications is an implementation of the Apache Flink framework. Kinesis Data Analytics uses the default values described in this section. Some of these values can be set by Kinesis Data Analytics applications in Java code, and others cannot be changed.

This topic contains the following sections:
- State Backend (p. 160)
- Checkpointing (p. 160)
- Savepointing (p. 161)
- Heap Sizes (p. 161)

State Backend

Kinesis Data Analytics stores transient data in a state backend. Kinesis Data Analytics uses the RocksDBStateBackend. Calling setStateBackend to set a different backend has no effect.

We enable the following features on the state backend:
- Incremental state backend snapshots
- Asynchronous state backend snapshots
- Local recovery of checkpoints

In Kinesis Data Analytics, the state.backend.rocksdb.ttl.compaction.filter.enabled configuration is enabled by default. Using this filter, you can update your application code to enable the compaction cleanup strategy. For more information, see State TTL in Flink 1.8.0 in the Apache Flink documentation.

For more information about state backends, see State Backends in the Apache Flink documentation.

Checkpointing

Kinesis Data Analytics for Java Applications uses a default checkpoint configuration with the following values. Some of these values can be changed. You must set CheckpointConfiguration.ConfigurationType to CUSTOM for Kinesis Data Analytics to use modified checkpointing values.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Can be modified?</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckpointingEnabled</td>
<td>Modifiable</td>
<td>True</td>
</tr>
<tr>
<td>CheckpointInterval</td>
<td>Modifiable</td>
<td>60000</td>
</tr>
<tr>
<td>MinPauseBetweenCheckpoints</td>
<td>Modifiable</td>
<td>5000</td>
</tr>
<tr>
<td>Number of Concurrent Checkpoints</td>
<td>Not Modifiable</td>
<td>1</td>
</tr>
<tr>
<td>Checkpointing Mode</td>
<td>Not Modifiable</td>
<td>Exactly Once</td>
</tr>
</tbody>
</table>
Savepointing

By default, when restoring from a savepoint, the resume operation will try to map all state of the savepoint back to the program you are restoring with. If you dropped an operator, by default, restoring from a savepoint that has data that corresponds to the missing operator will fail. You can allow the operation to succeed by setting the `AllowNonRestoredState` parameter of the application’s `FlinkRunConfiguration` to `true`. This will allow the resume operation to skip state that cannot be mapped to the new program.

For more information, see Allowing Non-Restored State in the Apache Flink documentation.

Heap Sizes

Kinesis Data Analytics for Java Applications allocates each KPU 3 GiB of JVM heap, and reserves 1 GiB for native code allocations. For information about increasing your application capacity, see the section called “Scaling” (p. 19).

For more information about JVM heap sizes, see Configuration in the Apache Flink documentation.
Configuring Kinesis Data Analytics for Java Applications to access Resources in an Amazon VPC

You can configure a Kinesis Data Analytics application to connect to private subnets in a virtual private cloud (VPC) in your account. Use Amazon Virtual Private Cloud (Amazon VPC) to create a private network for resources such as databases, cache instances, or internal services. Connect your application to the VPC to access private resources during execution.

This topic contains the following sections:
- Amazon VPC Concepts (p. 162)
- VPC Application Permissions (p. 163)
- Internet and Service Access for a VPC-Connected Kinesis Data Analytics for Java application (p. 163)
- Kinesis Data Analytics VPC API (p. 164)
- Example: Using a VPC to Access Data in an Amazon MSK Cluster (p. 166)
- Kinesis Data Analytics VPC Limitations (p. 166)

Amazon VPC Concepts

Amazon VPC is the networking layer for Amazon EC2. If you're new to Amazon EC2, see What is Amazon EC2? in the Amazon EC2 User Guide for Linux Instances to get a brief overview.

The following are the key concepts for VPCs:
- A virtual private cloud (VPC) is a virtual network dedicated to your AWS account.
- A subnet is a range of IP addresses in your VPC.
- A route table contains a set of rules, called routes, that are used to determine where network traffic is directed.
- An internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between instances in your VPC and the internet. It therefore imposes no availability risks or bandwidth constraints on your network traffic.
- A VPC endpoint enables you to privately connect your VPC to supported AWS services and VPC endpoint services powered by PrivateLink without requiring an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC do not require public IP addresses to communicate with resources in the service. Traffic between your VPC and the other service does not leave the Amazon network.

For more information about the Amazon VPC service, see the Amazon Virtual Private Cloud User Guide. Kinesis Data Analytics creates elastic network interfaces in one of the subnets provided in your VPC configuration for the application. The number of elastic network interfaces created in your VPC subnets may vary, depending on the parallelism and parallelism per KPU of the application. For more information about application scaling, see Scaling (p. 19).

Note
VPC configurations are not supported for SQL applications.
**VPC Application Permissions**

This section describes the permission policies your application will need to work with your VPC. For more information about using permissions policies, see Identity and Access Management (p. 110).

The following permissions policy grants your application the necessary permissions to interact with a VPC. To use this permission policy, add it to your application's execution role.

### Permissions Policy for Accessing an Amazon VPC

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VPCReadOnlyPermissions",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeDhcpOptions"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ENIReadWritePermissions",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": "*"
        }
    ]
}
```

**Note**

When you specify application resources using the console (such as CloudWatch Logs or an Amazon VPC), the console modifies your application execution role to grant permission to access those resources. You only need to manually modify your application's execution role if you create your application without using the console.

### Internet and Service Access for a VPC-Connected Kinesis Data Analytics for Java application

By default, when you connect a Kinesis Data Analytics for Java application to a VPC in your account, it does not have access to the internet unless the VPC provides access. If the application needs internet access, the following need to be true:
• The Kinesis Data Analytics for Java application should only be configured with private subnets.
• The VPC must contain a NAT gateway or instance in a public subnet.
• A route must exist for outbound traffic from the private subnets to the NAT gateway in a public subnet.

Note
Several services offer VPC endpoints. You can use VPC endpoints to connect to AWS services from within a VPC without internet access.

Whether a subnet is public or private depends on its route table. Every route table has a default route, which determines the next hop for packets that have a public destination.
• For a Private subnet: The default route points to a NAT gateway (nat-...) or NAT instance (eni-...).
• For a Public subnet: The default route points to an internet gateway (igw-...).

Once you configure your VPC with a public subnet (with a NAT) and one or more private subnets, do the following to identify your private and public subnets:
• In the VPC console, from the navigation pane, choose Subnets.
• Select a subnet, and then choose the Route Table tab. Verify the default route:
  • Public subnet: Destination: 0.0.0.0/0, Target: igw-…
  • Private subnet: Destination: 0.0.0.0/0, Target: nat-… or eni-…

To associate the Kinesis Data Analytics for Java application with private subnets:
• Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
• On the Kinesis Analytics applications page, choose your application, and choose Application details.
• On the page for your application, choose Configure.
• In the VPC Connectivity section, choose the VPC to associate with your application. Choose the subnets and security group associated with your VPC that you want the application to use to access VPC resources.
• Choose Update.

Related Information
Creating a VPC with Public and Private Subnets
NAT gateway basics

Kinesis Data Analytics VPC API

Use the following Kinesis Data Analytics API operations to manage VPCs for your application. For information on using the Kinesis Data Analytics API, see API Example Code (p. 175).

CreateApplication

Use the CreateApplication action to add a VPC configuration to your application during creation.

The following example request code for the CreateApplication action includes a VPC configuration when the application is created:
AddApplicationVpcConfiguration

Use the AddApplicationVpcConfiguration action to add a VPC configuration to your application after it has been created.

The following example request code for the AddApplicationVpcConfiguration action adds a VPC configuration to an existing application:

```json
{
    "ApplicationName": "MyApplication",
    "CurrentApplicationVersionId": 9,
    "VpcConfiguration": {
        "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
        "SubnetIds": [ "subnet-0123456789abcdef0" ]
    }
}
```

DeleteApplicationVpcConfiguration

Use the DeleteApplicationVpcConfiguration action to remove a VPC configuration from your application.

The following example request code for the DeleteApplicationVpcConfiguration action removes an existing VPC configuration from an application:

```json
{
    "ApplicationName": "MyApplication",
    "CurrentApplicationVersionId": 9,
    "VpcConfiguration": {
        "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
        "SubnetIds": [ "subnet-0123456789abcdef0" ]
    }
}
```
UpdateApplication

Use the UpdateApplication action to update all of an application's VPC configurations at once.

The following example request code for the UpdateApplication action updates all of the VPC configurations for an application:

```
{
  "ApplicationConfigurationUpdate": {
    "VpcConfigurationUpdates": [
      {
        "SecurityGroupIdUpdates": [ "sg-0123456789abcdef0" ],
        "SubnetIdUpdates": [ "subnet-0123456789abcdef0" ],
        "VpcConfigurationId": "2.1"
      }
    ],
    "ApplicationName": "MyApplication",
    "CurrentApplicationVersionId": 9
  }
}
```

Example: Using a VPC to Access Data in an Amazon MSK Cluster

For a complete tutorial about how to access data from an Amazon MSK Cluster in a VPC, see MSK Replication (p. 71).

Kinesis Data Analytics VPC Limitations

Kinesis Data Analytics currently does not provide support for VPCs with the following CIDR ranges:

- 192.168.0.0/16
- 172.17.0.0/16
- 10.100.0.0/16
Troubleshooting Kinesis Data Analytics for Java Applications

The following can help you troubleshoot problems that you might encounter with Amazon Kinesis Data Analytics for Java Applications.

Topics
- General Troubleshooting: Analyze Logs (p. 167)
- Cannot access resources in a VPC (p. 167)
- Compile error: "Could not resolve dependencies for project" (p. 168)
- Invalid Choice: 'kinesisanalyticsv2' (p. 168)
- Data Is Lost When Writing to an Amazon S3 Bucket (p. 168)
- Application Is in RUNNING State but Not Processing Data (p. 168)
- Application Fails to enter RUNNING state (p. 169)
- Snapshot Fails to Be Created (p. 169)
- Restoring from a Snapshot Fails (p. 170)
- Throughput Is Too Slow or MillisBehindLatest Is Increasing (p. 170)
- Snapshot, Application Update, or Application Stop Error: InvalidApplicationConfigurationException (p. 171)
- Downtime Is Not Zero (p. 171)
- UpdateApplication action not reloading application code (p. 172)

General Troubleshooting: Analyze Logs

You can investigate issues with your application by querying your application's CloudWatch logs.

We recommend that you set your log level to INFO. This log level writes sufficient information to your logs to troubleshoot most issues. You can use the DEBUG log level for short periods of time while troubleshooting issues, but it can create significant performance issues for your application.

For information about setting up and analyzing CloudWatch logs, see Logging and Monitoring (p. 116).

If your application is not writing entries to your CloudWatch log, see the section called "Logging Troubleshooting" (p. 122).

Cannot access resources in a VPC

If your application uses an Amazon VPC, do the following to verify that your application has access to its resources:

- Check your CloudWatch logs for the following error. This error indicates that your application cannot access resources in your VPC:

```java
org.apache.kafka.common.errors.TimeoutException: Failed to update metadata after 60000 ms.
```
If you see this error, verify that your route tables are set up correctly, and that your connectors have the correct connection settings.

For information about setting up and analyzing CloudWatch logs, see Logging and Monitoring (p. 116).

- Verify that you are not using restricted CIDRs in the subnets in your Amazon VPC. For more information, see Limitations (p. 166).

**Compile error: "Could not resolve dependencies for project"**

In order to compile the Kinesis Data Analytics for Java Applications sample applications, you must first download and compile the Apache Flink Kinesis connector and add it to your local Maven repository. If the connector hasn’t been added to your repository, a compile error similar to the following appears:

```
Could not resolve dependencies for project your project name: Failure to find org.apache.flink:flink-connector-kinesis_2.11:jar:1.8.2 in https://repo.maven.apache.org/maven2 was cached in the local repository, resolution will not be reattempted until the update interval of central has elapsed or updates are forced
```

To resolve this error, you must download the Apache Flink source code (version 1.8.2 from https://flink.apache.org/downloads.html) for the connector. For instructions about how to download, compile, and install the Apache Flink source code, see the section called "Using the Apache Flink Kinesis Streams Connector" (p. 3).

**Invalid Choice: 'kinesisanalyticsv2'**

To use v2 of the Kinesis Data Analytics API, you need the latest version of the AWS Command Line Interface (AWS CLI).

For information about upgrading the AWS CLI, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide.

**Data Is Lost When Writing to an Amazon S3 Bucket**

Some data loss may occur when writing to an Amazon S3 bucket using Apache Flink version 1.6.2. We recommend using Apache Flink version 1.8.2 when using S3 for output directly. To write to an Amazon S3 bucket using Apache Flink 1.6.2, we recommend using Kinesis Data Firehose. For more information about using Kinesis Data Firehose with Kinesis Data Analytics, see Kinesis Data Firehose Sink (p. 79).

**Application Is in RUNNING State but Not Processing Data**

You can check your application state using either the ListApplications or the DescribeApplication actions. If your application enters the RUNNING state but is not writing data to your sink, you can troubleshoot the issue by adding an Amazon CloudWatch log stream to your
application. For more information, see Working with Application CloudWatch Logging Options (p. 118). The log stream contains messages that you can use to troubleshoot application issues.

Application Fails to enter RUNNING state

You can check your application state using either the ListApplications or the DescribeApplication actions. If your application fails to enter the RUNNING state, check the following:

- Verify that your application JAR file is smaller than 512 MB. JAR files larger than 512 MB are not supported.
- Check your application's CloudWatch logs for errors. For information about setting up CloudWatch logs for your application, see the section called “Setting Up Logging” (p. 116).

Snapshot Fails to Be Created

Kinesis Data Analytics takes a snapshot of the application during an UpdateApplication or StopApplication request. The service then uses this snapshot state and restores the application using the updated application configuration to provide exactly-once processing semantics.

The service can't take a snapshot of the application under the following circumstances:

- The application exceeded the snapshot limit. The limit for snapshots is 1,000. For more information, see Snapshots (p. 16).
- The application is not in a healthy state.
- The application does not have permissions to access its source or sink.
- The application code is not functioning properly.
- The application is experiencing other configuration issues.

If you get an exception while taking a snapshot during an application update or while stopping the application, check the application's CloudWatch logs for errors, and retry the request. You can also retry the request by setting the SnapshotsEnabled property of your application's ApplicationSnapshotConfiguration to false.

After the application returns to a healthy state, we recommend that you set the SnapshotsEnabled property to true.

You can set the SnapshotsEnabled property using the UpdateApplication action. The following UpdateApplication example sets the SnapshotsEnabled property to true:

```
aws kinesisanalyticsv2 update-application \
--application-name MyApplication \
--current-application-version-id 10 \
--application-configuration-update '{"ApplicationSnapshotConfigurationUpdate": "SnapshotsEnabledUpdate":true}''
```

You can also update the SnapshotsEnabled property using the console.

Update the SnapshotsEnabled Property Using the Console

1. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. In the Kinesis Data Analytics console, choose your application.
3. In your application's page, choose **Configure**.
4. In the **Snapshots** section, choose **Enable**.

## Restoring from a Snapshot Fails

If you remove or change an operator in an application update and attempt to restore from a snapshot, the restore will fail by default if the snapshot contains state data for the missing operator. In addition, the application will be stuck in either the **STOPPED** or **UPDATING** state. To change this behavior and allow the restore to succeed, change the **AllowNonRestoredState** parameter of the application's **FlinkRunConfiguration** to `true`. This will allow the resume operation to skip state data that cannot be mapped to the new program.

For more information, see the section called "Restoring From a Snapshot That Contains Incompatible State Data" (p. 17).

## Throughput Is Too Slow or MillisBehindLatest Is Increasing

If the application metrics are showing that throughput is too slow or the **MillisBehindLatest** metric is steadily increasing, do the following:

- Enable auto scaling if it is disabled, or increase application parallelism. For more information, see **Scaling** (p. 19).
- Check if the application is logging an entry for every record being processed. Writing a log entry for each record during times when the application has high throughput will cause severe bottlenecks in data processing. To check for this condition, query your logs for log entries that your application writes with every record it processes. For more information, see the section called “Analyzing Logs” (p. 123).
- Increase the application's parallelism. You update the application's parallelism using the **ParallelismConfigurationUpdate** parameter of the **UpdateApplication** action.

The maximum KPUs for an application is 32 by default, and can be increased by requesting a limit increase.
• Verify that your application's workload isn't distributed unevenly. To control the distribution of workload across your application's worker processes, use the Parallelism setting and the KeyBy operator. For more information, see the following topics in the Apache Flink documentation:
  • Parallel Execution
  • DataStream Transformations
• Examine your application logic for inefficient or non-performant operations, such as accessing an external dependency (such as a database or a web service), accessing application state, etc. If you are using an external dependency to enrich or otherwise process incoming data, consider using asynchronous IO instead. For more information, see Async I/O in the Apache Flink documentation.
• Ensure that your application is not throwing exceptions that lead to job restarts. Any job restarts will slow down processing as the job is recycled and rebuilt and state data is persisted and restored. For information about logging application errors using CloudWatch Logs, see the section called “Setting Up Logging” (p. 116).

### Snapshot, Application Update, or Application Stop Error: InvalidApplicationConfigurationException

An error similar to the following may occur during a snapshot operation, or during an operation that creates a snapshot, such as updating or stopping an application:

```
An error occurred (InvalidApplicationConfigurationException) when calling the UpdateApplication operation:
Failed to take snapshot for the application xxxx at this moment. The application is currently experiencing downtime.
Please check the application's CloudWatch metrics or CloudWatch logs for any possible errors and retry the request.
You can also retry the request after disabling the snapshots in the Kinesis Data Analytics console or by updating the ApplicationSnapshotConfiguration through the AWS SDK
```

This error occurs when the application is unable to create a snapshot.

If you encounter this error during a snapshot operation or an operation that creates a snapshot, do the following:

• Disable snapshots for your application. You can do this either in the Kinesis Data Analytics console, or by using the SnapshotsEnabledUpdate parameter of the UpdateApplication action.
• Investigate why snapshots cannot be created. For more information, see the section called “Snapshot Fails to Be Created” (p. 169).
• Re-enable snapshots when the application returns to a healthy state.

### Downtime Is Not Zero

If the Downtime metric is not zero, the application is not healthy. Common causes of this condition include the following:

• Your application is under-provisioning sources and sinks. Check that any sources or sinks used in the application are well-provisioned, and are not experiencing read or write throttling.

If the source or sink is a Kinesis data stream, check the metrics for the stream for ReadProvisionedThroughputExceeded or WriteProvisionedThroughputExceeded errors.
You can investigate the causes of this condition by querying your application logs for changes from your application's state from RUNNING to FAILED. For more information, see the section called "Analyze Errors: Application Task-Related Failures" (p. 125).

- If any exception in an operator in your application is unhandled, the application fails over (by interpreting that the failure cannot be handled by operator) and the application will restart from the latest checkpoint to maintain "exactly-once" processing semantics. This will lead to Downtime being not zero during these restart periods. In order to prevent this from happening, we recommend that you handle any retryable exceptions in the application code.


The location of the SSL truststore was updated in a previous deployment. Use the following value for the ssl.truststore.location parameter instead:

/us/local/openjdk-8/jre/lib/security/cacerts

UpdateApplication action not reloading application code

The UpdateApplication action will not reload application code with the same filename if no S3 object version is specified. To reload application code with the same filename, enable versioning on your S3 bucket, and specify the new object version using the ObjectVersionUpdate parameter. For more information about enabling object versioning in an S3 bucket, see Enabling or Disabling Versioning.
# Document History for Amazon Kinesis Data Analytics for Java Applications

The following table describes the important changes to the documentation since the last release of Amazon Kinesis Data Analytics.

- **API version:** 2018-05-23
- **Latest documentation update:** March 30, 2020

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example: Read From a Kinesis Stream in a Different Account</td>
<td>Learn how to access a Kinesis stream in a different AWS account in your Kinesis Data Analytics application. For more information, see Cross-Account (p. 103).</td>
<td>March 30, 2020</td>
</tr>
<tr>
<td>Support for Apache Flink 1.8.2</td>
<td>Kinesis Data Analytics now supports Java applications that use Apache Flink 1.8.2. Use the Flink StreamingFileSink connector to write output directly to S3. For more information, see Creating Applications (p. 2).</td>
<td>December 17, 2019</td>
</tr>
<tr>
<td>Kinesis Data Analytics VPC</td>
<td>Configure a Kinesis Data Analytics application to connect to a virtual private cloud. For more information, see Using an Amazon VPC (p. 162).</td>
<td>November 25, 2019</td>
</tr>
<tr>
<td>Kinesis Data Analytics Best Practices</td>
<td>Best practices for creating and administering Kinesis Data Analytics applications. For more information, see Best Practices (p. 137).</td>
<td>October 14, 2019</td>
</tr>
<tr>
<td>Analyze Kinesis Data Analytics Application Logs</td>
<td>Use CloudWatch Logs Insights to monitor your Kinesis Data Analytics application. For more information, see Analyzing Logs (p. 123).</td>
<td>June 26, 2019</td>
</tr>
<tr>
<td>Kinesis Data Analytics Application Runtime Properties</td>
<td>Work with Runtime Properties in Kinesis Data Analytics for Java Applications. For more</td>
<td>June 24, 2019</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>Change</td>
<td>information, see Runtime Properties (p. 10).</td>
<td></td>
</tr>
<tr>
<td>Tagging Kinesis Data Analytics Applications</td>
<td>Use application tagging to determine per-application costs, control access, or for user-defined purposes. For more information, see Using Tagging (p. 22).</td>
<td>May 8, 2019</td>
</tr>
<tr>
<td>Kinesis Data Analytics Example Applications</td>
<td>Example applications for Amazon Kinesis Data Analytics demonstrating window operators and writing output to CloudWatch Logs. For more information, see Examples (p. 44).</td>
<td>May 1, 2019</td>
</tr>
<tr>
<td>Logging Kinesis Data Analytics API Calls with AWS CloudTrail</td>
<td>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 information, see Using AWS CloudTrail (p. 132).</td>
<td>March 22, 2019</td>
</tr>
<tr>
<td>Create an Application (Kinesis Data Firehose Sink)</td>
<td>Exercise to create a Kinesis Data Analytics for Java application with an Amazon Kinesis data stream as a source, and an Amazon Kinesis Data Firehose delivery stream as a sink. For more information, see Kinesis Data Firehose Sink (p. 79).</td>
<td>December 13, 2018</td>
</tr>
<tr>
<td>Public release</td>
<td>This is the initial release of the Kinesis Data Analytics Developer Guide for Java Applications.</td>
<td>November 27, 2018</td>
</tr>
</tbody>
</table>
Kinesis Data Analytics API Example Code

This topic contains example request blocks for Kinesis Data Analytics actions.

To use JSON as the input for an action with the AWS Command Line Interface (AWS CLI), save the request in a JSON file. Then pass the file name into the action using the 

\[-cli-input-json parameter.

The following example demonstrates how to use a JSON file with an action.

```
$ aws kinesisanalyticsv2 start-application --cli-input-json file://start.json
```

For more information about using JSON with the AWS CLI, see Generate CLI Skeleton and CLI Input JSON Parameters in the AWS Command Line Interface User Guide.

Topics

- AddApplicationCloudWatchLoggingOption (p. 175)
- AddApplicationInput (p. 176)
- AddApplicationInputProcessingConfiguration (p. 176)
- AddApplicationOutput (p. 177)
- AddApplicationReferenceDataSource (p. 177)
- AddApplicationVpcConfiguration (p. 178)
- CreateApplication (p. 178)
- CreateApplicationSnapshot (p. 179)
- DeleteApplication (p. 179)
- DeleteApplicationCloudWatchLoggingOption (p. 179)
- DeleteApplicationInputProcessingConfiguration (p. 180)
- DeleteApplicationOutput (p. 180)
- DeleteApplicationReferenceDataSource (p. 180)
- DeleteApplicationSnapshot (p. 180)
- DeleteApplicationVpcConfiguration (p. 180)
- DescribeApplication (p. 181)
- DescribeApplicationSnapshot (p. 181)
- DiscoverInputSchema (p. 181)
- ListApplications (p. 182)
- ListApplicationSnapshots (p. 182)
- StartApplication (p. 182)
- StopApplication (p. 182)
- UpdateApplication (p. 183)

AddApplicationCloudWatchLoggingOption

The following example request code for the AddApplicationCloudWatchLoggingOption action adds an Amazon CloudWatch logging option to a Kinesis Data Analytics application:
AddApplicationInput

The following example request code for the `AddApplicationInput` action adds an application input to a Kinesis Data Analytics application:

```json
{
    "ApplicationName": "MyApplication",
    "CurrentApplicationVersionId": 2,
    "Input": {
        "InputParallelism": {
            "Count": 2
        },
        "InputSchema": {
            "RecordColumns": [
                {
                    "Mapping": "$.TICKER",
                    "Name": "TICKER_SYMBOL",
                    "SqlType": "VARCHAR(50)"
                },
                {
                    "SqlType": "REAL",
                    "Name": "PRICE",
                    "Mapping": "$.PRICE"
                }
            ],
            "RecordEncoding": "UTF-8",
            "RecordFormat": {
                "MappingParameters": {
                    "JSONMappingParameters": {
                        "RecordRowPath": "$"
                    }
                },
                "RecordFormatType": "JSON"
            }
        },
        "KinesisStreamsInput": {
            "ResourceARN": "arn:aws:kinesis:us-east-1:012345678901:stream/ExampleInputStream"
        }
    }
}
```

AddApplicationInputProcessingConfiguration

The following example request code for the `AddApplicationInputProcessingConfiguration` action adds an application input processing configuration to a Kinesis Data Analytics application:

```json
{
}
```
AddApplicationOutput

The following example request code for the `AddApplicationOutput` action adds a Kinesis data stream as an application output to a Kinesis Data Analytics application:

```json
{
    "ApplicationName": "MyApplication",
    "CurrentApplicationVersionId": 2,
    "Output": {
        "DestinationSchema": {
            "RecordFormatType": "JSON"
        },
        "KinesisStreamsOutput": {
        },
        "Name": "DESTINATION_SQL_STREAM"
    }
}
```

AddApplicationReferenceDataSource

The following example request code for the `AddApplicationReferenceDataSource` action adds a CSV application reference data source to a Kinesis Data Analytics application:

```json
{
    "ApplicationName": "MyApplication",
    "CurrentApplicationVersionId": 5,
    "ReferenceDataSource": {
        "ReferenceSchema": {
            "RecordColumns": [
                {
                    "Mapping": "$\.TICKER",
                    "Name": "TICKER",
                    "SqlType": "VARCHAR(4)"
                },
                {
                    "Mapping": "$\.COMPANYNAME",
                    "Name": "COMPANY_NAME",
                    "SqlType": "VARCHAR(40)"
                }
            ],
            "RecordEncoding": "UTF-8",
            "RecordFormat": {
                "MappingParameters": {
                    "CSVMappingParameters": {
                        "RecordColumnDelimiter": ",",
                        "RecordRowDelimiter": "\r\n"
                    }
                }
            }
        }
    }
}
```
AddApplicationVpcConfiguration

The following example request code for the AddApplicationVpcConfiguration action adds a VPC configuration to an existing application:

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 9,
  "VpcConfiguration": {
    "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
    "SubnetIds": [ "subnet-0123456789abcdef0" ]
  }
}
```

CreateApplication

The following example request code for the CreateApplication action creates a Kinesis Data Analytics application:

```
{
  "ApplicationName": "MyApplication",
  "ApplicationDescription": "My-Application-Description",
  "RuntimeEnvironment": "FLINK-1_8",
  "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
  "CloudWatchLoggingOptions": [
    {
    }
  ],
  "ApplicationConfiguration": {
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-east-1",
            "flink.stream.initpos": "LATEST"
          }
        },
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-east-1"
          }
        }
      ]
    }
  }
}
```
CreateApplicationSnapshot

The following example request code for the CreateApplicationSnapshot action creates a snapshot of application state:

```json
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MySnapshot"
}
```

DeleteApplication

The following example request code for the DeleteApplication action deletes a Kinesis Data Analytics application:

```json
{"ApplicationName": "MyApplication",
 "CreateTimestamp": 12345678912}
```

DeleteApplicationCloudWatchLoggingOption

The following example request code for the DeleteApplicationCloudWatchLoggingOption action deletes an Amazon CloudWatch logging option from a Kinesis Data Analytics application:

```json
{
   "ApplicationName": "MyApplication",
   "CloudWatchLoggingOptionId": "3.1",
   "CurrentApplicationVersionId": 3
}
```
DeleteApplicationInputProcessingConfiguration

The following example request code for the **DeleteApplicationInputProcessingConfiguration** action removes an input processing configuration from a Kinesis Data Analytics application:

```json
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 4,
  "InputId": "2.1"
}
```

DeleteApplicationOutput

The following example request code for the **DeleteApplicationOutput** action removes an application output from a Kinesis Data Analytics application:

```json
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 4,
  "OutputId": "4.1"
}
```

DeleteApplicationReferenceDataSource

The following example request code for the **DeleteApplicationReferenceDataSource** action removes an application reference data source from a Kinesis Data Analytics application:

```json
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 5,
  "ReferenceId": "5.1"
}
```

DeleteApplicationSnapshot

The following example request code for the **DeleteApplicationSnapshot** action deletes a snapshot of application state:

```json
{
  "ApplicationName": "MyApplication",
  "SnapshotCreationTimestamp": 12345678912,
  "SnapshotName": "MySnapshot"
}
```

DeleteApplicationVpcConfiguration

The following example request code for the **DeleteApplicationVpcConfiguration** action removes an existing VPC configuration from an application:
DescribeApplication

The following example request code for the DescribeApplication action returns details about a Kinesis Data Analytics application:

```json
{"ApplicationName": "MyApplication"
}
```

DescribeApplicationSnapshot

The following example request code for the DescribeApplicationSnapshot action returns details about a snapshot of application state:

```json
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MySnapshot"
}
```

DiscoverInputSchema

The following example request code for the DiscoverInputSchema action generates a schema from a streaming source:

```json
{
   "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
      }
   },
   "InputStartingPositionConfiguration": {
      "InputStartingPosition": "NOW"
   },
   "S3Configuration": {
      "BucketARN": "string",
      "FileKey": "string"
   },
   "ServiceExecutionRole": "string"
}
```

The following example request code for the DiscoverInputSchema action generates a schema from a reference source:

```json
{
}
```
"S3Configuration": {
    "BucketARN": "arn:aws:s3:::mybucket",
    "FileKey": "TickerReference.csv"
},
"ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole"

## ListApplications

The following example request code for the ListApplications action returns a list of Kinesis Data Analytics applications in your account:

```json
{
    "ExclusiveStartApplicationName": "MyApplication",
    "Limit": 50
}
```

## ListApplicationSnapshots

The following example request code for the ListApplicationSnapshots action returns a list of snapshots of application state:

```json
{"ApplicationName": "MyApplication",
 "Limit": 50,
 "NextToken": "aBcDeFgHiJkLmNoPqRsTuVwXyZ0123"
}
```

## StartApplication

The following example request code for the StartApplication action starts a Kinesis Data Analytics application, and loads the application state from the latest snapshot (if any):

```json
{
    "ApplicationName": "MyApplication",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
            "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
        }
    }
}
```

## StopApplication

The following example request code for the API_StopApplication action stops a Kinesis Data Analytics application:

```json
{"ApplicationName": "MyApplication"}
```
The following example request code for the `UpdateApplication` action updates a Kinesis Data Analytics application to change the location of the application code:

```json
{"ApplicationName": "MyApplication",
"CurrentApplicationVersionId": 1,
"ApplicationConfigurationUpdate": {
  "ApplicationCodeConfigurationUpdate": {
    "CodeContentTypeUpdate": "ZIPFILE",
    "CodeContentUpdate": {
      "S3ContentLocationUpdate": {
        "BucketARNUpdate": "arn:aws:s3:::my_new_bucket",
        "FileKeyUpdate": "my_new_code.zip",
        "ObjectVersionUpdate": "2"
      }
    }
  }
}
```
Kinesis Data Analytics for Java Applications API Reference

For information about the CLI actions that Kinesis Data Analytics for Java Applications provides, see Kinesis Data Analytics CLI API Reference.