

---

# Amazon Kinesis Data Analytics

## Amazon Kinesis Data Analytics Developer Guide



# **Amazon Kinesis Data Analytics: Amazon Kinesis Data Analytics Developer Guide**

Copyright © Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

## Table of Contents

What Is Kinesis Data Analytics for Apache Flink? .....	1
Getting Started .....	1
How It Works .....	2
Programming Your Apache Flink Application .....	2
DataStream API .....	2
Table API .....	3
Creating Your Kinesis Data Analytics Application .....	3
Creating Applications .....	3
Building your Kinesis Data Analytics Application Code .....	4
Creating your Kinesis Data Analytics Application .....	5
Starting your Kinesis Data Analytics Application .....	5
Verifying your Kinesis Data Analytics Application .....	6
Using Apache Beam .....	6
Running Applications .....	7
Application and Job Status .....	7
Application Resources .....	8
Kinesis Data Analytics Application Resources .....	8
Apache Flink Application Resources .....	8
DataStream API .....	9
DataStream API Connectors .....	9
DataStream API Operators .....	14
DataStream API Timestamps .....	14
Table API .....	15
Table API Connectors .....	15
Table API Time Attributes .....	16
Using Python .....	17
Programming an Application .....	17
Creating an Application .....	19
Monitoring .....	20
Runtime Properties .....	21
Working with Runtime Properties in the Console .....	21
Working with Runtime Properties in the CLI .....	21
Accessing Runtime Properties in a Kinesis Data Analytics Application .....	23
Fault Tolerance .....	24
Configuring Checkpointing .....	24
Checkpointing API Examples .....	25
Snapshots .....	26
Scaling .....	30
Configuring Application Parallelism and ParallelismPerKPU .....	30
Allocating Kinesis Processing Units .....	30
Updating Your Application's Parallelism .....	31
Automatic Scaling .....	32
Tagging .....	33
Adding Tags when an Application is Created .....	33
Adding or Updating Tags for an Existing Application .....	34
Listing Tags for an Application .....	34
Removing Tags from an Application .....	34
Apache Flink Dashboard .....	34
Accessing Your Application's Apache Flink Dashboard .....	35
Studio notebooks .....	37
Creating a Studio notebook .....	38
Interactive analysis of streaming data .....	38
Flink interpreters .....	39
Apache Flink table environment variables .....	39

Deploying as an application with durable state .....	40
Scala/Python criteria .....	41
SQL criteria .....	41
IAM Permissions .....	41
Connectors and dependencies .....	42
Default connectors .....	42
Dependencies and custom connectors .....	43
User-Defined Functions .....	43
Enabling Checkpointing .....	44
Setting the checkpointing interval .....	44
Setting the checkpointing type .....	44
Working with AWS Glue .....	44
Table properties .....	45
Examples and Tutorials .....	46
Creating a Studio notebook Tutorial .....	46
Deploying as an Application with Durable State Tutorial .....	60
Examples .....	63
Studio Notebooks vs. SQL Applications .....	70
Troubleshooting .....	70
Stopping a stuck application .....	70
Canceling jobs .....	70
Restarting the Apache Flink interpreter .....	71
Appendix: Creating custom IAM policies .....	71
AWS Glue .....	71
CloudWatch Logs .....	72
Kinesis streams .....	72
Amazon MSK clusters .....	74
Getting Started (DataStream API) .....	75
Application Components .....	75
Prerequisites .....	75
Step 1: Set Up an Account .....	76
Sign Up for AWS .....	76
Create an IAM User .....	76
Next Step .....	78
Step 2: Set Up the AWS CLI .....	78
Next Step .....	79
Step 3: Create an Application .....	79
Create Two Amazon Kinesis Data Streams .....	79
Write Sample Records to the Input Stream .....	80
Download and Examine the Apache Flink Streaming Java Code .....	81
Compile the Application Code .....	81
Upload the Apache Flink Streaming Java Code .....	82
Create and Run the Kinesis Data Analytics Application .....	82
Next Step .....	90
Step 4: Clean Up .....	90
Delete Your Kinesis Data Analytics Application .....	90
Delete Your Kinesis Data Streams .....	90
Delete Your Amazon S3 Object and Bucket .....	90
Delete Your IAM Resources .....	91
Delete Your CloudWatch Resources .....	91
Next Step .....	91
Step 5: Next Steps .....	91
Getting Started (Table API) .....	93
Application Components .....	93
Prerequisites .....	93
Create an Application .....	94
Create Dependent Resources .....	94

Write Sample Records to the Input Stream .....	95
Download and Examine the Apache Flink Streaming Java Code .....	96
Compile the Application Code .....	97
Upload the Apache Flink Streaming Java Code .....	97
Create and Run the Kinesis Data Analytics Application .....	98
Next Step .....	101
Clean Up .....	101
Delete Your Kinesis Data Analytics Application .....	101
Delete Your Amazon MSK Cluster .....	101
Delete Your VPC .....	101
Delete Your Amazon S3 Objects and Bucket .....	102
Delete Your IAM Resources .....	102
Delete Your CloudWatch Resources .....	102
Next Step .....	102
Next Steps .....	102
Getting Started (Python) .....	103
Application Components .....	103
Prerequisites .....	103
Create an Application .....	104
Create Dependent Resources .....	104
Write Sample Records to the Input Stream .....	105
Create and Examine the Apache Flink Streaming Python Code .....	106
Upload the Apache Flink Streaming Python Code .....	107
Create and Run the Kinesis Data Analytics Application .....	108
Next Step .....	111
Clean Up .....	111
Delete Your Kinesis Data Analytics Application .....	111
Delete Your Kinesis Data Streams .....	111
Delete Your Amazon S3 Objects and Bucket .....	112
Delete Your IAM Resources .....	112
Delete Your CloudWatch Resources .....	112
Examples .....	113
DataStream API Examples .....	113
Tumbling Window .....	113
Sliding Window .....	120
S3 Sink .....	126
MSK Replication .....	135
EFO Consumer .....	139
Kinesis Data Firehose Sink .....	146
Cross-Account .....	157
Custom Truststore .....	163
Apache Beam .....	168
Python Examples .....	174
Tumbling Window .....	175
Sliding Window .....	181
S3 Sink .....	188
Kinesis Data Analytics Solutions .....	195
AWS Streaming Data Solution .....	196
Clickstream Lab .....	196
Custom Scaling .....	196
CloudWatch Dashboard .....	196
Amazon MSK .....	196
More Kinesis Data Analytics Solutions on GitHub .....	197
Security .....	198
Data Protection .....	198
Data Encryption .....	198
Identity and Access Management .....	199

Trust Policy .....	200
Permissions Policy .....	200
Monitoring .....	202
Compliance Validation .....	202
Resilience .....	202
Disaster Recovery .....	203
Versioning .....	203
Infrastructure Security .....	203
Security Best Practices .....	204
Implement least privilege access .....	204
Use IAM roles to access other Amazon services .....	204
Implement Server-Side Encryption in Dependent Resources .....	204
Use CloudTrail to Monitor API Calls .....	204
Logging and Monitoring .....	206
Setting Up Logging .....	206
Setting Up CloudWatch Logging Using the Console .....	207
Setting Up CloudWatch Logging Using the CLI .....	208
Application Monitoring Levels .....	211
Logging Best Practices .....	212
Logging Troubleshooting .....	212
Next Step .....	213
Analyzing Logs .....	213
Run a Sample Query .....	213
Example Queries .....	214
Metrics and Dimensions .....	215
Application Metrics .....	216
Kinesis Data Streams Connector Metrics .....	222
Amazon MSK Connector Metrics .....	223
Apache Zeppelin Metrics .....	224
Viewing CloudWatch Metrics .....	224
Metrics .....	225
Custom Metrics .....	226
Alarms .....	229
Writing Custom Messages .....	233
Write to CloudWatch Logs Using Log4J .....	233
Write to CloudWatch Logs Using SLF4J .....	234
Using AWS CloudTrail .....	235
Kinesis Data Analytics Information in CloudTrail .....	235
Understanding Kinesis Data Analytics Log File Entries .....	236
Performance .....	238
Troubleshooting Performance .....	238
The Data Path .....	238
Performance Troubleshooting Solutions .....	238
Performance Best Practices .....	240
Manage scaling properly .....	240
Monitor external dependency resource usage .....	241
Run your Apache Flink application locally .....	242
Monitoring Performance .....	242
Performance Monitoring using CloudWatch Metrics .....	242
Performance Monitoring using CloudWatch Logs and Alarms .....	242
Quota .....	243
Maintenance .....	244
Best Practices .....	246
Fault tolerance: checkpoints and savepoints .....	246
Performance and parallelism .....	246
Logging .....	247
Coding .....	247

Studio notebook refresh interval .....	247
Studio notebook optimum performance .....	247
Earlier Versions .....	249
Using the Apache Flink Kinesis Streams Connector with previous Apache Flink versions .....	249
Building Applications with Apache Flink 1.8.2 .....	250
Building Applications with Apache Flink 1.6.2 .....	251
Upgrading Applications .....	251
Available Connectors in Apache Flink 1.6.2 and 1.8.2 .....	252
Getting Started: Flink 1.11.3 .....	252
Application Components .....	252
Prerequisites .....	253
Step 1: Set Up an Account .....	253
Step 2: Set Up the AWS CLI .....	255
Step 3: Create an Application .....	256
Step 4: Clean Up .....	267
Step 5: Next Steps .....	268
Getting Started: Flink 1.8.2 .....	269
Application Components .....	75
Prerequisites .....	270
Step 1: Set Up an Account .....	270
Step 2: Set Up the AWS CLI .....	272
Step 3: Create an Application .....	273
Step 4: Clean Up .....	284
Getting Started: Flink 1.6.2 .....	285
Application Components .....	286
Prerequisites .....	286
Step 1: Set Up an Account .....	286
Step 2: Set Up the AWS CLI .....	289
Step 3: Create an Application .....	290
Step 4: Clean Up .....	300
Flink Settings .....	302
State Backend .....	302
Checkpointing .....	302
Savepointing .....	303
Heap Sizes .....	303
Using an Amazon VPC .....	304
Amazon VPC Concepts .....	304
VPC Application Permissions .....	305
Permissions Policy for Accessing an Amazon VPC .....	305
Internet and Service Access .....	305
Related Information .....	306
VPC API .....	306
CreateApplication .....	306
AddApplicationVpcConfiguration .....	307
DeleteApplicationVpcConfiguration .....	307
UpdateApplication .....	308
Example: Using a VPC .....	308
Troubleshooting .....	309
Development Troubleshooting .....	309
Enabling Flamegraps .....	309
Issue with EFO connector 1.13.2 .....	309
Compile Error: "Could not resolve dependencies for project" .....	309
Invalid Choice: "kinesisanalyticsv2" .....	310
UpdateApplication Action Isn't Reloading Application Code .....	310
Runtime Troubleshooting .....	310
Troubleshooting Tools .....	310
Application Issues .....	310

Application is Restarting .....	313
Throughput is Too Slow .....	315
Application State Data is Accumulating .....	316
Checkpointing Timing Out .....	316
Document History .....	318
API Example Code .....	321
AddApplicationCloudWatchLoggingOption .....	321
AddApplicationInput .....	322
AddApplicationInputProcessingConfiguration .....	322
AddApplicationOutput .....	323
AddApplicationReferenceDataSource .....	323
AddApplicationVpcConfiguration .....	324
CreateApplication .....	324
CreateApplicationSnapshot .....	325
DeleteApplication .....	325
DeleteApplicationCloudWatchLoggingOption .....	325
DeleteApplicationInputProcessingConfiguration .....	326
DeleteApplicationOutput .....	326
DeleteApplicationReferenceDataSource .....	326
DeleteApplicationSnapshot .....	326
DeleteApplicationVpcConfiguration .....	326
DescribeApplication .....	327
DescribeApplicationSnapshot .....	327
DiscoverInputSchema .....	327
ListApplications .....	328
ListApplicationSnapshots .....	328
StartApplication .....	328
StopApplication .....	328
UpdateApplication .....	329
API Reference .....	330



# What Is Amazon Kinesis Data Analytics for Apache Flink?

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

You can build Java and Scala 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.

**Note**

Although Kinesis Data Analytics supports Apache Flink applications written in Scala version 2.12, this guide only contains code examples written in Java.

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, functions, 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 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 Apache Flink: How It Works \(p. 2\)](#)
- [Getting Started with Amazon Kinesis Data Analytics for Apache Flink \(DataStream API\) \(p. 75\)](#)

# Kinesis Data Analytics for Apache Flink: How It Works

Kinesis Data Analytics for Apache Flink is a fully managed Amazon service that enables you to use an Apache Flink application to process streaming data.

## Programming Your Apache Flink Application

An Apache Flink application is a Java or Scala application that is created with the Apache Flink framework. You author and build your Apache Flink application locally.

Applications primarily use either the [DataStream API](#) or the [Table API](#). The other Apache Flink APIs are also available for you to use, but they are less commonly used in building streaming applications.

The features of the two APIs are as follows:

### [DataStream API](#)

The Apache Flink DataStream API programming model is based on two components:

- **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.

Applications created with the DataStream API do the following:

- Read data from a Data Source (such as a Kinesis stream or Amazon MSK topic).
- Apply transformations to the data, such as filtering, aggregation, or enrichment.
- Write the transformed data to a Data Sink.

Applications that use the DataStream API can be written in Java or Scala, and can read from a Kinesis data stream, a Amazon MSK topic, or a custom source.

Your application processes data by using a *connector*. Apache Flink uses the following types of connectors:

- **Source:** A connector used to read external data.
- **Sink:** A connector used to write to external locations.
- **Operator:** A connector used to process data within the application.

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.

For more information about using the DataStream API, see [DataStream API \(p. 9\)](#).

## Table API

The Apache Flink Table API programming model is based on the following components:

- **Table Environment:** An interface to underlying data that you use to create and host one or more tables.
- **Table:** An object providing access to a SQL table or view.
- **Table Source:** Used to read data from an external source, such as an Amazon MSK topic.
- **Table Function:** A SQL query or API call used to transform data.
- **Table Sink:** Used to write data to an external location, such as an Amazon S3 bucket.

Applications created with the Table API do the following:

- Create a `TableEnvironment` by connecting to a `Table Source`.
- Create a table in the `TableEnvironment` using either SQL queries or Table API functions.
- Run a query on the table using either Table API or SQL
- Apply transformations on the results of the query using Table Functions or SQL queries.
- Write the query or function results to a `Table Sink`.

Applications that use the Table API can be written in Java or Scala, and can query data using either API calls or SQL queries.

For more information about using the Table API, see [Table API \(p. 15\)](#).

## Creating Your Kinesis Data Analytics Application

A Kinesis Data Analytics application is an AWS resource that is hosted by the Kinesis Data Analytics service. Your Kinesis Data Analytics application hosts your Apache Flink application and provides it with the following settings:

- **Runtime Properties (p. 21):** Parameters that you can provide to your application. You can change these parameters without recompiling your application code.
- **Fault Tolerance (p. 24):** How your application recovers from interrupts and restarts.
- **Logging and Monitoring (p. 206):** How your application logs events to CloudWatch Logs.
- **Scaling (p. 30):** How your application provisions computing resources.

You create your Kinesis Data Analytics application using either the console or the AWS CLI. To get started creating a Kinesis Data Analytics application, see [Getting Started \(DataStream API\) \(p. 75\)](#).

## Creating a Kinesis Data Analytics for Apache Flink Application

This topic contains information about creating a Kinesis Data Analytics for Apache Flink application.

**This topic contains the following sections:**

- [Building your Kinesis Data Analytics Application Code \(p. 4\)](#)

- [Creating your Kinesis Data Analytics Application \(p. 5\)](#)
- [Starting your Kinesis Data Analytics Application \(p. 5\)](#)
- [Verifying your Kinesis Data Analytics Application \(p. 6\)](#)
- [Creating Kinesis Data Analytics applications with Apache Beam \(p. 6\)](#)

## Building your Kinesis Data Analytics Application Code

This section describes the components you use to build the application code for your Kinesis Data Analytics 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 **1.13.2**. For information about upgrading Kinesis Data Analytics applications, see [Upgrading Applications \(p. 251\)](#).

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:

Component	Version
Java	11 (recommended)
Scala	2.12
Kinesis Data Analytics for Flink Runtime (aws-kinesisanalytics-runtime)	1.2.0
Kinesis Data Analytics Flink Connectors (aws-kinesisanalytics-flink)	2.0.0
AWS Kinesis Connector (flink-connector-kinesis)	<a href="#">1.13.2</a>
Apache Beam (Beam Applications Only)	2.33.0, with Jackson version 2.12.2

For an example of a `pom.xml` file for a Kinesis Data Analytics application that uses Apache Flink version 1.13.2, see the [Kinesis Data Analytics Getting Started Application](#).

For information about creating a Kinesis Data Analytics application that uses **Apache Beam**, see [Using Apache Beam \(p. 6\)](#).

## Specifying your Application's Apache Flink Version

When using Kinesis Data Analytics for Flink Runtime version 1.1.0 and later, 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=1.13.2
```

For building applications with older versions of Apache Flink, see [Earlier Versions \(p. 249\)](#).

## 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” \(p. 82\)](#) in the [Getting Started \(DataStream API\) \(p. 75\)](#) 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 \(Console\)” \(p. 82\)](#) in the [Getting Started \(DataStream API\) \(p. 75\)](#) 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.

For a tutorial about how to use the CLI to create an application, see [the section called “Create and Run an Application Using the CLI” \(p. 85\)](#) in the [Getting Started \(DataStream API\) \(p. 75\)](#) tutorial.

### Note

You cannot change the `RuntimeEnvironment` of an existing application. If you need to change the `RuntimeEnvironment` of an existing application, you must delete the application and create it again.

## 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 \(p. 206\)](#).
- **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.

## Creating Kinesis Data Analytics applications with Apache Beam

You can use the [Apache Beam](#) framework with your Kinesis Data Analytics application to process streaming data. Kinesis Data Analytics applications that use Apache Beam use [Apache Flink runner](#) to execute Beam pipelines.

For a tutorial about how to use Apache Beam in a Kinesis Data Analytics application, see [Apache Beam \(p. 168\)](#).

**This topic contains the following sections:**

- [Using Apache Beam with Kinesis Data Analytics \(p. 6\)](#)
- [Beam Capabilities \(p. 6\)](#)

## Using Apache Beam with Kinesis Data Analytics

Note the following about using the Apache Flink runner with Kinesis Data Analytics:

- Apache Beam metrics are not viewable in the Kinesis Data Analytics console.
- **Apache Beam is only supported with Kinesis Data Analytics applications that use Apache Flink version 1.8 and above. Apache Beam is not supported with Kinesis Data Analytics applications that use Apache Flink version 1.6.**

## Beam Capabilities

Kinesis Data Analytics supports the same Apache Beam capabilities as the Apache Flink runner. For information about what features are supported with the Apache Flink runner, see the [Beam Compatibility Matrix](#).

We recommend that you test your Apache Flink application in the Kinesis Data Analytics service to verify that we support all the features that your application needs.

# Running a Kinesis Data Analytics for Apache Flink Application

This topic contains information about running a Kinesis Data Analytics for Apache Flink application.

When you run your Kinesis Data Analytics application, the Kinesis Data Analytics service creates an Apache Flink job. An Apache Flink job is the execution lifecycle of your Kinesis Data Analytics application. The execution of the job, and the resources it uses, are managed by the Job Manager. The Job Manager separates the execution of the application into tasks. Each task is managed by a Task Manager. When you monitor your application's performance, you can examine the performance of each Task Manager, or of the Job Manager as a whole.

For information about Apache Flink jobs, see [Jobs and Scheduling](#) in the [Apache Flink Documentation](#).

## Application and Job Status

Both your application and the application's job have a current execution status:

- **Application status:** Your application has a current status that describes its phase of execution. Application statuses include the following:
  - **Steady application statuses:** Your application typically stays in these statuses until you make a status change:
    - **READY:** A new or stopped application is in the READY status until you run it.
    - **RUNNING:** An application that has successfully started is in the RUNNING status.
  - **Transient application statuses:** An application in these statuses is typically in the process of transitioning to another status. If an application stays in a transient status for a length of time, you can stop the application using the [StopApplication](#) action with the `Force` parameter set to `true`. These statuses include the following:
    - **STARTING:** Occurs after the [StartApplication](#) action. The application is transitioning from the READY to the RUNNING status.
    - **STOPPING:** Occurs after the [StopApplication](#) action. The application is transitioning from the RUNNING to the READY status.
    - **DELETING:** Occurs after the [DeleteApplication](#) action. The application is in the process of being deleted.
    - **UPDATING:** Occurs after the [UpdateApplication](#) action. The application is updating, and will transition back to the RUNNING or READY status.
    - **AUTOSCALING:** The application has the `AutoScalingEnabled` property of the [ParallelismConfiguration](#) set to `true`, and the service is increasing the parallelism of the application. When the application is in this status, the only valid API action you can use is the [StopApplication](#) action with the `Force` parameter set to `true`. For information about automatic scaling, see [Automatic Scaling](#) (p. 32).
    - **FORCE\_STOPPING:** Occurs after the [StopApplication](#) action is called with the `Force` parameter set to `true`. The application is in the process of being force stopped. The application transitions from the STARTING, UPDATING, STOPPING, or AUTOSCALING status to the READY status.
    - **ROLLING\_BACK:** Occurs after the [RollbackApplication](#) action is called. The application is in the process of being rolled back to a previous version. The application transitions from the UPDATING or AUTOSCALING status to the RUNNING status.
    - **ROLLED\_BACK:** When you successfully roll back an application, this becomes the status of the version that you rolled back from. For information about rolling back an application, see [RollbackApplication](#).
    - **MAINTENANCE:** Occurs while Kinesis Data Analytics applies patches to your application. For more information, see [Maintenance](#) (p. 244).

You can check your application's status using the console, or by using the [DescribeApplication](#) action.

- **Job status:** When your application is in the `RUNNING` status, your job has a status that describes its current execution phase. A job starts in the `CREATED` status, and then proceeds to the `RUNNING` status when it has started. If error conditions occur, your application enters the following status:
  - For applications using Apache Flink 1.11 and later, your application enters the `RESTARTING` status.
  - For applications using Apache Flink 1.8 and prior, your application enters the `FAILING` status.

The application then proceeds to either the `RESTARTING` or `FAILED` status, depending on whether the job can be restarted.

You can check the job's status by examining your application's CloudWatch log for status changes.

## Application Resources

This section describes the system resources that your application uses. Understanding how Kinesis Data Analytics provisions and uses resources will help you design, create, and maintain a performant and stable Kinesis Data Analytics application.

### Kinesis Data Analytics Application Resources

Kinesis Data Analytics is an AWS service that creates an environment for hosting your Apache Flink application. The Kinesis Data Analytics service provides resources using units called **Kinesis Processing Units (KPU)**.

One KPU represents the following system resources:

- One CPU core
- 4 GB of memory, of which one GB is native memory and three GB are heap memory
- 50 GB of disk space

KPUs run applications in distinct execution units called **tasks** and **subtasks**. You can think of a subtask as the equivalent of a thread.

The number of KPUs available to an application is equal to the application's `Parallelism` setting, divided by the application's `ParallelismPerKPU` setting.

For more information about application parallelism, see [Scaling](#) (p. 30).

### Apache Flink Application Resources

The Apache Flink environment allocates resources for your application using units called **task slots**. When Kinesis Data Analytics allocates resources for your application, it assigns one or more Apache Flink task slots to a single KPU. The number of slots assigned to a single KPU is equal to your application's `ParallelismPerKPU` setting. For more information about task slots, see [Job Scheduling](#) in the [Apache Flink documentation](#).

### Operator Parallelism

You can set the maximum number of subtasks that an operator can use. This value is called **Operator Parallelism**. By default, the parallelism of each operator in your application is equal to the application's parallelism. This means that by default, each operator in your application can use all of the available subtasks in the application if needed.



You can set the parallelism of the operators in your application using the `setParallelism` method. Using this method, you can control the number of subtasks each operator can use at one time.

For more information about operator chaining, see [Task chaining and resource groups](#) in the [Apache Flink Documentation](#).

## Operator Chaining

Normally, each operator uses a separate subtask to execute, but if several operators always execute in sequence, the runtime can assign them all to the same task. This process is called **Operator Chaining**.

Several sequential operators can be chained into a single task if they all operate on the same data. The following are some of the criteria needed for this to be true:

- The operators do 1-to-1 simple forwarding.
- The operators all have the same operator parallelism.

When your application chains operators into a single subtask, it conserves system resources, because the service doesn't need to perform network operations and allocate subtasks for each operator. To determine if your application is using operator chaining, look at the job graph in the Kinesis Data Analytics console. Each vertex in the application represents one or more operators. The graph shows operators that have been chained as a single vertex.

## DataStream API

Your Apache Flink application uses the [Apache Flink DataStream API](#) to transform data in a data stream.

This section contains the following topics:

- [Using Connectors to Move Data in Kinesis Data Analytics for Apache Flink With the DataStream API \(p. 9\)](#): These components move data between your application and external data sources and destinations.
- [Transforming Data Using Operators in Kinesis Data Analytics for Apache Flink With the DataStream API \(p. 14\)](#): These components transform or group data elements within your application.
- [Tracking Events in Kinesis Data Analytics for Apache Flink Using the DataStream API \(p. 14\)](#): This topic describes how Kinesis Data Analytics tracks events when using the DataStream API.

## Using Connectors to Move Data in Kinesis Data Analytics for Apache Flink With the DataStream API

In the Amazon Kinesis Data Analytics for Apache Flink DataStream API, *connectors* are software components that move data into and out of a 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 Amazon services and third-party systems.

Types of connectors include the following:

- [Sources \(p. 10\)](#): Provide data to your application from a Kinesis data stream, file, or other data source.
- [Sinks \(p. 11\)](#): Send data from your application to a Kinesis data stream, Kinesis Data Firehose delivery stream, or other data destination.
- [Asynchronous I/O \(p. 13\)](#): 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 Apache Flink

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 Amazon 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`:

```
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 \(p. 81\)](#).

#### Creating a `FlinkKinesisConsumer` that uses an EFO consumer

The `FlinkKinesisConsumer` now supports [Enhanced Fan-Out \(EFO\)](#).

If a Kinesis consumer uses EFO, the Kinesis Data Streams service gives it its own dedicated bandwidth, rather than having the consumer share the fixed bandwidth of the stream with the other consumers reading from the stream.

For more information about using EFO with the Kinesis consumer, see [FLIP-128: Enhanced Fan Out for AWS Kinesis Consumers](#).

You enable the EFO consumer by setting the following parameters on the Kinesis consumer:

- **RECORD\_PUBLISHER\_TYPE:** Set this parameter to **EFO** for your application to use an EFO consumer to access the Kinesis Data Stream data.
- **EFO\_CONSUMER\_NAME:** Set this parameter to a string value that is unique among the consumers of this stream. Re-using a consumer name in the same Kinesis Data Stream will cause the previous consumer using that name to be terminated.

To configure a `FlinkKinesisConsumer` to use EFO, add the following parameters to the consumer:

```
consumerConfig.putIfAbsent(RECORD_PUBLISHER_TYPE, "EFO");
consumerConfig.putIfAbsent(EFO_CONSUMER_NAME, "basic-efo-flink-app");
```

For an example of a Kinesis Data Analytics application that uses an EFO consumer, see [EFO Consumer \(p. 139\)](#).

## Amazon MSK

The `FlinkKafkaConsumer` source provides streaming data to your application from an Amazon MSK topic.

### Creating a `FlinkKafkaConsumer`

The following code example demonstrates creating a `FlinkKafkaConsumer`:

```
Properties inputProperties = new Properties();
inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
inputProperties.setProperty("bootstrap.servers", "Cluster Bootstrap Broker String");
inputProperties.setProperty("security.protocol", "SSL");
inputProperties.setProperty("ssl.truststore.location", "/usr/lib/jvm/java-11-amazon-
corretto/lib/security/cacerts");
inputProperties.setProperty("ssl.truststore.password", "changeit");

DataStream<string> input = env.addSource(new FlinkKafkaConsumer<>("MyMSKTopic", new
SimpleStringSchema(), inputProperties));
```

For more information about using a `FlinkKafkaConsumer`, see [MSK Replication](#) (p. 135).

## Writing Data Using Sinks in Kinesis Data Analytics for Apache Flink

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:

### Kinesis Data Streams

Apache Flink provides information about the [Kinesis Data Streams Connector](#) in the Apache Flink documentation.

For an example of an application that uses a Kinesis data stream for input and output, see [Getting Started \(DataStream API\)](#) (p. 75).

### 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. 126).

### 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. 11)
- [FlinkKinesisFirehoseProducer Code Example](#) (p. 12)

### Creating a `FlinkKinesisFirehoseProducer`

The following code example demonstrates creating a `FlinkKinesisFirehoseProducer`:

```
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.

```
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.ConsumerConfigConstants;

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";

    private 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());

    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” \(p. 146\)](#).

## Using Asynchronous I/O in Kinesis Data Analytics for Apache Flink

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 Apache Flink With the DataStream API

To transform incoming data in a Kinesis Data Analytics for Apache Flink 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 \(p. 14\)](#)
- [Aggregation Operators \(p. 14\)](#)

### 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 \(DataStream API\) \(p. 75\)](#). 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 Apache Flink Using the DataStream API

Kinesis Data Analytics 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](#).

## Table API

Your Apache Flink application uses the [Apache Flink Table API](#) to interact with data in a stream using a relational model. You use the Table API to access data using Table sources, and then use Table functions to transform and filter table data. You can transform and filter tabular data using either API functions or SQL commands.

This section contains the following topics:

- [Table API Connectors \(p. 15\)](#): These components move data between your application and external data sources and destinations.
- [Table API Time Attributes \(p. 16\)](#): This topic describes how Kinesis Data Analytics tracks events when using the Table API.

## Table API Connectors

In the Apache Flink programming model, connectors are components that your application uses to read or write data from external sources, such as other AWS services.

With the Apache Flink Table API, you can use the following types of connectors:

- [Table API Sources \(p. 15\)](#): You use Table API source connectors to create tables within your `TableEnvironment` using either API calls or SQL queries.
- [Table API Sinks \(p. 16\)](#): You use SQL commands to write table data to external sources such as an Amazon MSK topic or an Amazon S3 bucket.

## Table API Sources

You create a table source from a data stream. The following code creates a table from an Amazon MSK topic:

```
//create the table
final FlinkKafkaConsumer<StockRecord> consumer = new
FlinkKafkaConsumer<StockRecord>(kafkaTopic, new KafkaEventDeserializationSchema(),
kafkaProperties);
consumer.setStartFromEarliest();
//Obtain stream
DataStream<StockRecord> events = env.addSource(consumer);
```

```
Table table = streamTableEnvironment.fromDataStream(events);
```

For more information about table sources, see [Table & Connectors](#) in the [Apache Flink documentation](#).

## Table API Sinks

To write table data to a sink, you create the sink in SQL, and then run the SQL-based sink on the `StreamTableEnvironment` object.

The following code example demonstrates how to write table data to an Amazon S3 sink:

```
final String s3Sink = "CREATE TABLE sink_table (" +  
    "event_time TIMESTAMP," +  
    "ticker STRING," +  
    "price DOUBLE," +  
    "dt STRING," +  
    "hr STRING" +  
    ")" +  
    " PARTITIONED BY (ticker,dt,hr)" +  
    " WITH" +  
    "(" +  
    " 'connector' = 'filesystem'," +  
    " 'path' = '" + s3Path + "'," +  
    " 'format' = 'json'" +  
    ")" +  
    ";  
  
//send to s3  
streamTableEnvironment.executeSql(s3Sink);  
filteredTable.executeInsert("sink_table");
```

You can use the `format` parameter to control what format Kinesis Data Analytics uses to write the output to the sink. For information about formats, see [Formats](#) in the [Apache Flink documentation](#).

For more information about table sinks, see [Table & Connectors](#) in the [Apache Flink documentation](#).

## User-Defined Sources and Sinks

You can use existing Apache Kafka connectors for sending data to and from other AWS services, such as Amazon MSK and Amazon S3. For interacting with other data sources and destinations, you can define your own sources and sinks. For more information, see [User-Defined Sources and Sinks](#) in the [Apache Flink documentation](#).

## Table API Time Attributes

Each record in a data stream has several timestamps that define when events related to the record occurred:

- **Event Time:** A user-defined timestamp that defines when the event that created the record occurred.
- **Ingestion Time:** The time when your application retrieved the record from the data stream.
- **Processing Time:** The time when your application processed the record.

When the Apache Flink Table API creates windows based on record times, you define which of these timestamps it uses by using the `setStreamTimeCharacteristic` method.

For more information about using timestamps with the Table API, see [Time Attributes](#) in the [Apache Flink documentation](#).



## Using Python with Kinesis Data Analytics

Apache Flink version 1.13.2 includes support for creating applications using Python version 3.8, using the [PyFlink](#) library. You create a Kinesis Data Analytics application using Python by doing the following:

- Create your Python application code as a text file with a `main` method.
- Bundle your application code file and any Python or Java dependencies into a zip file, and upload it to an Amazon S3 bucket.
- Create your Kinesis Data Analytics application, specifying your Amazon S3 code location, application properties, and application settings.

At a high level, the Python Table API is a wrapper around the Java Table API. For information about the Python Table API, see [Intro to the Python Table API](#) in the [Apache Flink Documentation](#).

## Programming Your Kinesis Data Analytics for Python Application

You code your Kinesis Data Analytics for Python application using the Apache Flink Python Table API. The Apache Flink engine translates Python Table API statements (running in the Python VM) into Java Table API statements (running in the Java VM).

You use the Python Table API by doing the following:

- Create a reference to the `StreamTableEnvironment`.
- Create table objects from your source streaming data by executing queries on the `StreamTableEnvironment` reference.
- Execute queries on your table objects to create output tables.
- Write your output tables to your destinations using a `StatementSet`.

To get started using the Python Table API in Kinesis Data Analytics, see [Getting Started with Amazon Kinesis Data Analytics for Apache Flink for Python \(p. 103\)](#).

## Reading and Writing Streaming Data

To read and write streaming data, you execute SQL queries on the table environment.

### Creating a Table

The following code example demonstrates a user-defined function that creates a SQL query. The SQL query creates a table that interacts with a Kinesis stream:

```
def create_table(table_name, stream_name, region, stream_initpos):
    return """ CREATE TABLE {0} (
        `record_id` VARCHAR(64) NOT NULL,
        `event_time` BIGINT NOT NULL,
        `record_number` BIGINT NOT NULL,
        `num_retries` BIGINT NOT NULL,
        `verified` BOOLEAN NOT NULL
    )
    PARTITIONED BY (record_id)
    WITH (
        'connector' = 'kinesis',
        'stream' = '{1}',
        'aws.region' = '{2}',
        'scan.stream.initpos' = '{3}',
```

```
'sink.partitionner-field-delimiter' = ';',  
'sink.producer.collection-max-count' = '100',  
'format' = 'json',  
'json.timestamp-format.standard' = 'ISO-8601'  
) """.format(table_name, stream_name, region, stream_initpos)
```

## Reading Streaming Data

The following code example demonstrates how to use preceding CreateTableSQL query on a table environment reference to read data:

```
table_env.execute_sql(create_table(input_table, input_stream, input_region,  
stream_initpos))
```

## Writing Streaming Data

The following code example demonstrates how to use the SQL query from the CreateTable example to create an output table reference, and how to use a StatementSet to interact with the tables to write data to a destination Kinesis stream:

```
table_result = table_env.execute_sql("INSERT INTO {0} SELECT * FROM {1}"  
                                     .format(output_table_name, input_table_name))
```

## Reading Runtime Properties

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

You specify application properties for your application the same way as with a Kinesis Data Analytics for Java application. You can specify runtime properties in the following ways:

- Using the [CreateApplication](#) action.
- Using the [UpdateApplication](#) action.
- Configuring your application by using the console.

You retrieve application properties in code by reading a json file called `application_properties.json` that the Kinesis Data Analytics runtime creates.

The following code example demonstrates reading application properties from the `application_properties.json` file:

```
file_path = '/etc/flink/application_properties.json'  
if os.path.isfile(file_path):  
    with open(file_path, 'r') as file:  
        contents = file.read()  
        properties = json.loads(contents)
```

The following user-defined function code example demonstrates reading a property group from the application properties object: retrieves:

```
def property_map(properties, property_group_id):  
    for prop in props:  
        if prop["PropertyGroupId"] == property_group_id:  
            return prop["PropertyMap"]
```

The following code example demonstrates reading a property called `INPUT_STREAM_KEY` from a property group that the previous example returns:

```
input_stream = input_property_map[INPUT_STREAM_KEY]
```

## Creating your application's code package

Once you have created your Python application, you bundle your code file and dependencies into a zip file.

Your zip file must contain a python script with a `main` method, and can optionally contain the following:

- Additional Python code files
- User-defined Java code in JAR files
- Java libraries in JAR files

### Note

Your application zip file must contain all of the dependencies for your application. You can't reference libraries from other sources for your application.

## Creating Your Python Kinesis Data Analytics Application

### Specifying your Code Files

Once you have created your application's code package, you upload it to an Amazon S3 bucket. You then create your application using either the console or the [CreateApplication](#) action.

When you create your application using the [CreateApplication](#) action, you specify the code files and archives in your zip file using a special application property group called `kinesis.analytics.flink.run.options`. You can define the following types files:

- **python:** A text file containing a Python main method.
- **jarfile:** A Java JAR file containing Java user-defined functions.
- **pyFiles:** A Python resource file containing resources to be used by the application.
- **pyArchives:** A zip file containing resource files for the application.

For more information about Apache Flink Python code file types, see [Command Line Usage](#) in the [Apache Flink Documentation](#).

### Note

Kinesis Data Analytics does not support the `pyModule`, `pyExecutable`, or `pyRequirements` file types. All of the code, requirements, and dependencies must be in your zip file. You can't specify dependencies to be installed using `pip`.

The following example json snippet demonstrates how to specify file locations within your application's zip file:

```
"ApplicationConfiguration": {
  "EnvironmentProperties": {
    "PropertyGroups": [
      {
        "PropertyGroupId": "kinesis.analytics.flink.run.options",
        "PropertyMap": {
          "python": "MyApplication/main.py",
          "jarfile": "MyApplication/lib/myJarFile.jar",
          "pyFiles": "MyApplication/lib/myDependentFile.py",
          "pyArchives": "MyApplication/lib/myArchive.zip"
        }
      }
    ]
  }
}
```

```
}  
},
```

## Monitoring Your Python Kinesis Data Analytics Application

You use your application's CloudWatch log to monitor your Python Kinesis Data Analytics application.

Kinesis Data Analytics logs the following messages for Python applications:

- Messages written to the console using `print()` in the application's main method.
- Messages sent in user-defined functions using the logging package. The following code example demonstrates writing to the application log from a user-defined function:

```
import logging  
  
@udf(input_types=[DataTypes.BIGINT()], result_type=DataTypes.BIGINT())  
def doNothingUdf(i):  
    logging.info("Got {} in the doNothingUdf".format(str(i)))  
    return i
```

- Error messages thrown by the application.

If the application throws an exception in the main function, it will appear in your application's logs.

The following example demonstrates a log entry for an exception thrown from Python code:

```
2021-03-15 16:21:20.000 ----- Python Process Started  
-----  
2021-03-15 16:21:21.000 Traceback (most recent call last):  
2021-03-15 16:21:21.000 " File "/tmp/flink-web-6118109b-1cd2-439c-9dcd-218874197fa9/  
flink-web-upload/4390b233-75cb-4205-a532-441a2de83db3_code/PythonKinesisSink/  
PythonUdfUndeclared.py", line 101, in <module>"  
2021-03-15 16:21:21.000     main()  
2021-03-15 16:21:21.000 " File "/tmp/flink-web-6118109b-1cd2-439c-9dcd-218874197fa9/  
flink-web-upload/4390b233-75cb-4205-a532-441a2de83db3_code/PythonKinesisSink/  
PythonUdfUndeclared.py", line 54, in main"  
2021-03-15 16:21:21.000 "     table_env.register_function("doNothingUdf",  
doNothingUdf)"  
2021-03-15 16:21:21.000 NameError: name 'doNothingUdf' is not defined  
2021-03-15 16:21:21.000 ----- Python Process Exited  
-----  
2021-03-15 16:21:21.000 Run python process failed  
2021-03-15 16:21:21.000 Error occurred when trying to start the job
```

### Note

Due to performance issues, we recommend that you only use custom log messages during application development.

## Querying Logs with CloudWatch Insights

The following CloudWatch Insights query searches for logs created by the Python entrypoint while executing the main function of your application:

```
fields @timestamp, message  
| sort @timestamp asc  
| filter logger like /PythonDriver/
```

| limit 1000

# Runtime Properties in Kinesis Data Analytics for Apache Flink

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 \(p. 21\)](#)
- [Working with Runtime Properties in the CLI \(p. 21\)](#)
- [Accessing Runtime Properties in a Kinesis Data Analytics Application \(p. 23\)](#)

## Working with Runtime Properties in the Console

You can add, update, or remove runtime properties from your Kinesis Data Analytics 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 application

1. Open the Kinesis Data Analytics console at <https://console.aws.amazon.com/kinesisanalytics>.
2. Choose your Kinesis Data Analytics 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 \(p. 321\)](#).

### 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_13",
```

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

**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:

```
{
  "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 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:

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

You retrieve a property group (as a `Java.Util.Properties` object) as follows:

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

You typically configure an Apache 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:

```
private static FlinkKinesisProducer<String> createSinkFromApplicationProperties() throws
    IOException {
    Map<String, Properties> applicationProperties =
        KinesisAnalyticsRuntime.getApplicationProperties();
    FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<String>(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 \(DataStream API\)](#) (p. 75). Source code for the Getting Started application is available at [Getting Started](#) in the [Kinesis Data Analytics Java Examples](#) GitHub repository.

## Implementing Fault Tolerance in Kinesis Data Analytics for Apache Flink

Checkpointing is the method that is used for implementing fault tolerance in Amazon Kinesis Data Analytics for Apache Flink. 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 [Checkpoints](#) 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. 26).

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 Apache Flink

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 Command Line Interface, 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 \(p. 321\)](#).

### Configure Checkpointing for a New Application

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

```
{
  "ApplicationName": "MyApplication",
  "RuntimeEnvironment": "FLINK-1_13",
  "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::mybucket",
          "FileKey": "myflink.jar",
          "ObjectVersion": "AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      }
    },
    "FlinkApplicationConfiguration": {
      "CheckpointConfiguration": {
        "CheckpointingEnabled": "true",
        "CheckpointInterval": 20000,
        "ConfigurationType": "CUSTOM",
        "MinPauseBetweenCheckpoints": 10000
      }
    }
  }
}
```

### Disable Checkpointing for a New Application

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

```
{
  "ApplicationName": "MyApplication",
  "RuntimeEnvironment": "FLINK-1_13",
  "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"
      }
    }
  }
}
```

```
}
```

## Configure Checkpointing for an Existing Application

The following example request for the [UpdateApplication](#) action configures checkpointing for an existing application:

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

## Disable Checkpointing for an Existing Application

The following example request for the [UpdateApplication](#) action disables checkpointing for an existing application:

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

We recommend that your application create a snapshot several times a day to restart properly with correct state data. The correct frequency for your snapshots depends on your application's business logic. Taking frequent snapshots allows you to recover more recent data, but increases cost and requires more system resources.

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

- [CreateApplicationSnapshot](#)
- [DeleteApplicationSnapshot](#)
- [DescribeApplicationSnapshot](#)

- [ListApplicationSnapshots](#)

For the per-application limit on the number of snapshots, see [Quota \(p. 243\)](#). If your application reaches the limit on snapshots, then manually creating a snapshot fails with a `LimitExceededException`.

Kinesis Data Analytics never deletes 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 \(p. 27\)](#)
- [Restoring From a Snapshot That Contains Incompatible State Data \(p. 27\)](#)
- [Snapshot API Examples \(p. 28\)](#)

## Automatic Snapshot Creation

If `SnapshotsEnabled` is set to `true` in the [ApplicationSnapshotConfiguration](#) for the application, Kinesis Data Analytics automatically creates and uses snapshots 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.

Automatically created snapshots have the following qualities:

- The snapshot is managed by the service, but you can see the snapshot using the [ListApplicationSnapshots](#) action. Automatically created snapshots count against your snapshot limit.
- If your application exceeds the snapshot limit, manually created snapshots will fail, but the Kinesis Data Analytics service will still successfully create snapshots when the application is updated, scaled, or stopped. You must manually delete snapshots using the [DeleteApplicationSnapshot](#) action before creating more snapshots manually.

## 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 or later 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 \(p. 321\)](#).

### Enable Snapshots for an Application

The following example request for the [UpdateApplication](#) action enables snapshots for an application:

```
{
  "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:

```
{
  "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:

```
{
  "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:

```
{
  "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](#):

```
{
  "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:

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

## 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 Apache Flink

You can configure the parallel execution of tasks and the allocation of resources for Amazon Kinesis Data Analytics for Apache Flink 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 \(p. 30\)](#)
- [Allocating Kinesis Processing Units \(p. 30\)](#)
- [Updating Your Application's Parallelism \(p. 31\)](#)
- [Automatic Scaling \(p. 32\)](#)

## Configuring Application Parallelism and ParallelismPerKPU

You configure the parallel execution for your Kinesis Data Analytics 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 [Service Quotas](#).

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 \(p. 31\)](#).

The default limit for KPIs for your application is 32. For instructions on how to request an increase to this limit, see "To request a limit increase" in [Service Quotas](#).

## 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 \(p. 321\)](#).

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

```
{
  "ApplicationName": "string",
  "RuntimeEnvironment": "FLINK-1_13",
  "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:

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 4,
  "ApplicationConfigurationUpdate": {
    "FlinkApplicationConfigurationUpdate": {
      "ParallelismConfigurationUpdate": {
        "AutoScalingEnabledUpdate": "true",
        "ConfigurationTypeUpdate": "CUSTOM",
        "ParallelismPerKPUUpdate": 4,
        "ParallelismUpdate": 4
      }
    }
  }
}
```

The following example request for the [UpdateApplication](#) action disables parallelism for an existing application:

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 4,
  "ApplicationConfigurationUpdate": {
    "FlinkApplicationConfigurationUpdate": {
      "ParallelismConfigurationUpdate": {
        "AutoScalingEnabledUpdate": "false"
      }
    }
  }
}
```

## 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 up or down accordingly:

- Your application scales up (increases parallelism) when your CPU usage remains at 75 percent or above for 15 minutes.
- Your application scales down (decreases parallelism) when your CPU usage remains below 10 percent for six hours.

Kinesis Data Analytics will not reduce your application's `CurrentParallelism` value to less than your application's `Parallelism` setting.

When the Kinesis Data Analytics service is scaling your application, it will be in the `AUTOSCALING` status. You can check your current application status using the [DescribeApplication](#) or [ListApplications](#) actions. While the service is scaling your application, the only valid API action you can use is [StopApplication](#) with the `Force` parameter set to `true`.

You can use the `AutoScalingEnabled` property (part of [FlinkApplicationConfiguration](#)) to enable or disable auto scaling behavior. Your AWS account is charged for KPIs 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](#).

Note the following about application scaling:

- Automatic scaling is enabled by default.
- Scaling doesn't apply to Studio notebooks. However, if you deploy a Studio notebook as an application with durable state, then scaling will apply to the deployed application.
- Your application has a default limit of 32 KPUs. For more information, see [Quota \(p. 243\)](#).
- 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" \(p. 31\)](#) 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" \(p. 215\)](#).



## maxParallelism considerations

- Autoscale logic will prevent scaling a Flink job to a parallelism that will cause interference with the job and operator `maxParallelism`. For example, if a simple job with only a source and a sink where the source has `maxParallelism` 16 and the sink has 8, we will not autoscale the job to above 8.
- If `maxParallelism` is not set for a job, Flink will default to 128. Therefore, if you think that a job will need to run at a higher parallelism than 128, you will have to set that number for your application.
- If you expect to see your job autoscale but are not seeing it, ensure your `maxParallelism` values allow for it.

## 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 *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. 33)
- [Adding or Updating Tags for an Existing Application](#) (p. 34)
- [Listing Tags for an Application](#) (p. 34)
- [Removing Tags from an Application](#) (p. 34)

## Adding Tags when an Application is Created

You add tags when creating an application using the `tags` parameter of the [CreateApplication](#) action.

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

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

## Adding or Updating Tags for an Existing Application

You add tags to an application using the [TagResource](#) action. You cannot add tags to an application using the [UpdateApplication](#) action.

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

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

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

## Listing Tags for an Application

To list existing tags, you use the [ListTagsForResource](#) action.

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

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

## Removing Tags from an Application

To remove tags from an application, you use the [UntagResource](#) action.

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

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

# Using the Apache Flink Dashboard with Amazon Kinesis Data Analytics

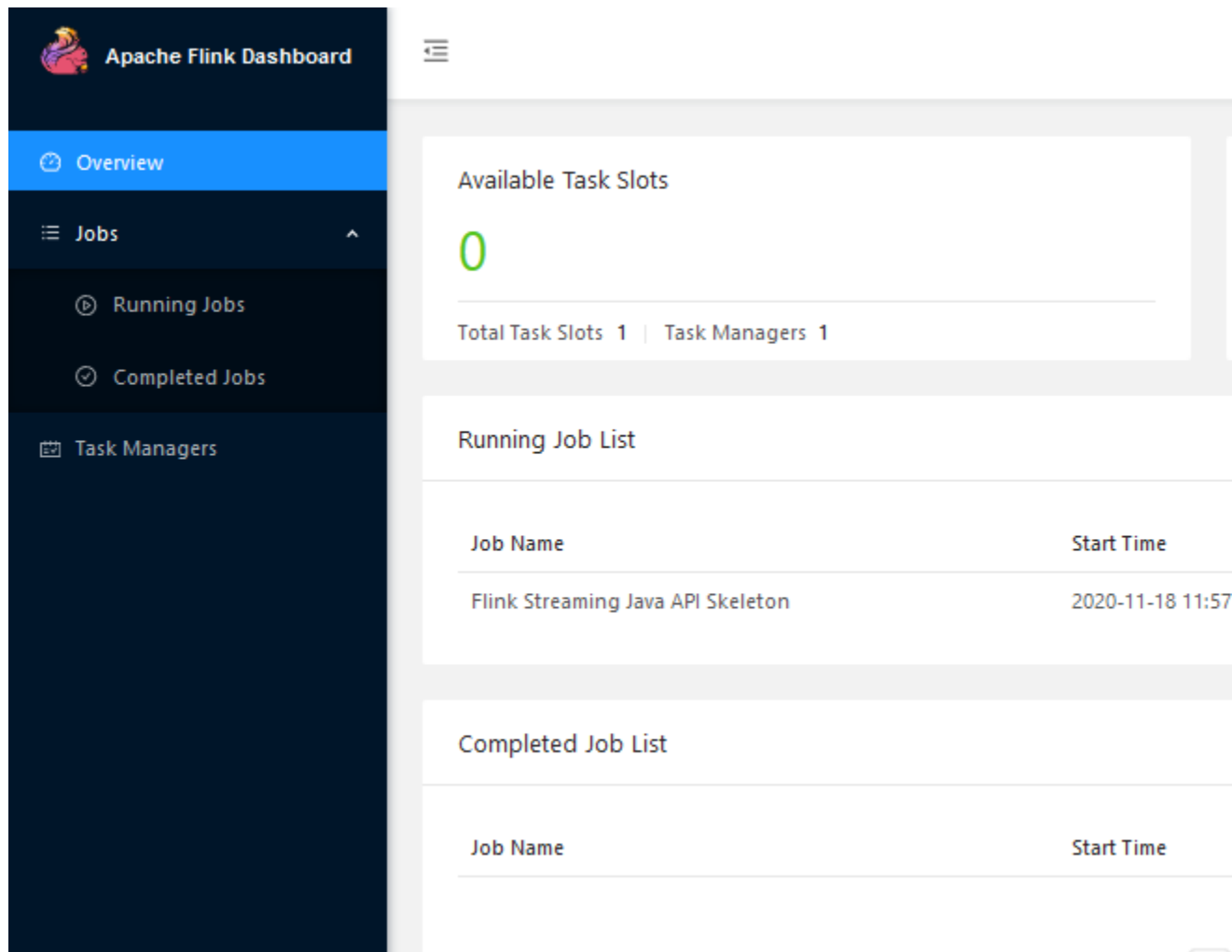
You can use your application's Apache Flink Dashboard to monitor your Kinesis Data Analytics application's health. Your application's dashboard shows the following information:

- Resources in use, including Task Managers and Task Slots.
- Information about Jobs, including those that are running, completed, canceled, and failed.

For information about Apache Flink Task Managers, Task Slots, and Jobs, see [Apache Flink Architecture](#) on the Apache Flink website.

Note the following about using the Apache Flink Dashboard with Kinesis Data Analytics applications:

- The Apache Flink Dashboard for Kinesis Data Analytics applications is read-only. You can't make changes to your Kinesis Data Analytics application using the Apache Flink Dashboard.
- The Apache Flink Dashboard is not compatible with Microsoft Internet Explorer.



## Accessing Your Application's Apache Flink Dashboard


You can access your application's Apache Flink Dashboard either through the Kinesis Data Analytics console, or by requesting a secure URL endpoint using the CLI.

### Accessing Your Application's Apache Flink Dashboard Using the Kinesis Data Analytics Console

To access your application's Apache Flink Dashboard from the console, choose **Apache Flink Dashboard** on your application's page.

Kinesis Data Analytics applications > MyApplication


## MyApplication

Apache Flink dashboard 

Run

Actions ▼

Configure

Choose **Configure** to add your application code. If you'd like a starting point for writing your Apache Flink application, see the [getting started tutorial](#) .

Application status: Running

### Application graph

The application graph is a visual representation of the data flow consisting of operators and intermediate results.

#### Note

When you open the dashboard from the Kinesis Data Analytics console, the URL that the console generates will be valid for 12 hours.

## Accessing your application's Apache Flink Dashboard using the Kinesis Data Analytics CLI

You can use the Kinesis Data Analytics CLI to generate a URL to access your application dashboard. The URL that you generate is valid for a specified amount of time.

#### Note

If you don't access the generated URL within three minutes, it will no longer be valid.

You generate your dashboard URL using the [CreateApplicationPresignedUrl](#) action. You specify the following parameters for the action:

- The application name
- The time in seconds that the URL will be valid
- You specify `FLINK_DASHBOARD_URL` as the URL type.

# Using a Studio notebook with Kinesis Data Analytics for Apache Flink

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

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

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

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

To get started using Kinesis Data Analytics and Apache Zeppelin, see [Creating a Studio notebook Tutorial \(p. 46\)](#). For more information about Apache Zeppelin, see the [Apache Zeppelin documentation](#).

With a notebook, you model queries using the Apache Flink [Table API & SQL](#) in SQL, Python, or Scala, or [DataStream API](#) in Scala. With a few clicks, you can then promote the Studio notebook to a continuously-running, non-interactive, Kinesis Data Analytics stream-processing application for your production workloads.

**This topic contains the following sections:**

- [Creating a Studio notebook \(p. 38\)](#)
- [Interactive analysis of streaming data \(p. 38\)](#)
- [Deploying as an application with durable state \(p. 40\)](#)
- [IAM permissions for Studio notebooks \(p. 41\)](#)
- [Connectors and dependencies \(p. 42\)](#)
- [User-defined functions \(p. 43\)](#)
- [Enabling Checkpointing \(p. 44\)](#)
- [Working with AWS Glue \(p. 44\)](#)
- [Examples and tutorials \(p. 46\)](#)
- [Comparing Studio notebooks and Kinesis Data Analytics for SQL Applications \(p. 70\)](#)
- [Troubleshooting \(p. 70\)](#)
- [Appendix: Creating custom IAM policies \(p. 71\)](#)

## Creating a Studio notebook

A Studio notebook contains queries or programs written in SQL, Python, or Scala that runs on streaming data and returns analytic results. You create your application using either the console or the CLI, and provide queries for analyzing the data from your data source.

Your application has the following components:

- A data source, such as an Amazon MSK cluster, a Kinesis data stream, or an Amazon S3 bucket.
- An AWS Glue database. This database contains tables, which store your data source and destination schemas and endpoints. For more information, see [Working with AWS Glue \(p. 44\)](#).
- Your application code. Your code implements your analytics query or program.
- Your application settings and runtime properties. For information about application settings and runtime properties, see the following topics in the [Developer Guide for Apache Flink Applications](#):
  - **Application Parallelism and Scaling:** You use your application's Parallelism setting to control the number of queries that your application can execute simultaneously. Your queries can also take advantage of increased parallelism if they have multiple paths of execution, such as in the following circumstances:
    - When processing multiple shards of a Kinesis data stream
    - When partitioning data using the `KeyBy` operator.
    - When using multiple window operators

For more information about application scaling, see [Application Scaling in Kinesis Data Analytics for Apache Flink](#).

- **Logging and Monitoring:** For information about application logging and monitoring, see [Logging and Monitoring in Amazon Kinesis Data Analytics for Apache Flink](#).
- Your application uses checkpoints and savepoints for fault tolerance. Checkpoints and savepoints are not enabled by default for Studio notebooks.

You can create your Studio notebook using either the AWS Management Console or the AWS CLI.

When creating the application from the console, you have the following options:

- In the Amazon MSK console choose your cluster, then choose **Process data in real time**.
- In the Kinesis Data Streams console choose your data stream, then on the **Applications** tab choose **Process data in real time**.
- In the Kinesis Data Analytics console choose the **Studio** tab, then choose **Create Studio notebook**.

For a tutorial about how to create a Studio notebook using either the AWS Management Console or the AWS CLI, see [Tutorial: Creating a Studio notebook in Kinesis Data Analytics \(p. 46\)](#).

## Interactive analysis of streaming data

You use a serverless notebook powered by Apache Zeppelin to interact with your streaming data. Your notebook can have multiple notes, and each note can have one or more paragraphs where you can write your code.

The following example SQL query shows how to retrieve data from a data source:

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

For more examples of Flink Streaming SQL queries, see [Examples and tutorials \(p. 46\)](#) following, and [Queries](#) in the [Apache Flink documentation](#).

You can use Flink SQL queries in the Studio notebook to query streaming data. You may also use Python (Table API) and Scala (Table and Datastream APIs) to write programs to query your streaming data interactively. You can view the results of your queries or programs, update them in seconds, and re-run them to view updated results.

## Flink interpreters

You specify which language Kinesis Data Analytics uses to run your application by using an *interpreter*. You can use the following interpreters with Kinesis Data Analytics:

Name	Class	Description
%flink	FlinkInterpreter	Creates ExecutionEnvironment/StreamExecutionEnvironment/BatchTableEnvironment/StreamTableEnvironment and provides a Scala environment
%flink.pyflink	PyFlinkInterpreter	Provides a python environment
%flink.ipynlink	IPyFlinkInterpreter	Provides an ipython environment
%flink.ssql	FlinkStreamSqlInterpreter	Provides a stream sql environment
%flink.bsql	FlinkBatchSqlInterpreter	Provides a batch sql environment

For more information about Flink interpreters, see [Flink interpreter for Apache Zeppelin](#).

## Apache Flink table environment variables

Apache Zeppelin provides access to table environment resources using environment variables.

You access Scala table environment resources with the following variables:

Variable	Resource
se <code>env</code>	StreamingTableEnvironment
be <code>nv</code>	ExecutionEnvironment
ste <code>nv</code>	StreamTableEnvironment for blink planner
bte <code>nv</code>	BatchTableEnvironment for blink planner
ste <code>nv</code> _2	StreamTableEnvironment for flink planner
bte <code>nv</code> _2	BatchTableEnvironment for flink planner

You access Python table environment resources with the following variables:

Variable	Resource
<code>s_env</code>	<code>StreamingTableEnvironment</code>
<code>b_env</code>	<code>ExecutionEnvironment</code>
<code>st_env</code>	<code>StreamTableEnvironment</code> for blink planner
<code>bt_env</code>	<code>BatchTableEnvironment</code> for blink planner
<code>st_env_2</code>	<code>StreamTableEnvironment</code> for flink planner
<code>bt_env_2</code>	<code>BatchTableEnvironment</code> for flink planner

For more information about using table environments, see [Create a TableEnvironment](#) in the [Apache Flink documentation](#).

## Deploying as an application with durable state

You can build your code and export it to Amazon S3. You can promote the code that you wrote in your note to a continuously running stream processing application. There are two modes of running an Apache Flink application on Kinesis Data Analytics: With a Studio notebook, you have the ability to develop your code interactively, view results of your code in real time, and visualize it within your note. After you deploy a note to run in streaming mode, Kinesis Data Analytics creates an application for you that runs continuously, reads data from your sources, writes to your destinations, maintains long-running application state, and autoscales automatically based on the throughput of your source streams.

### Note

The S3 bucket to which you export your application code must be in the same Region as your Studio notebook.

You can only deploy a note from your Studio notebook if it meets the following criteria:

- Paragraphs must be ordered sequentially. When you deploy your application, all paragraphs within a note will be executed sequentially (left-to-right, top-to-bottom) as they appear in your note. You can check this order by choosing **Run All Paragraphs** in your note.
- Your code is a combination of Python and SQL or Scala and SQL. We do not support Python and Scala together at this time for deploy-as-application.
- Your note should have only the following interpreters: `%flink`, `%flink.ssql`, `%flink.pyflink`, `%flink.ipynlink`, `%md`.
- The use of the [Zeppelin context](#) object `z` is not supported. Methods that return nothing will do nothing except log a warning. Other methods will raise Python exceptions or fail to compile in Scala.
- A note must result in a single Apache Flink job.
- Notes with [dynamic forms](#) are unsupported for deploying as an application.
- `%md` ([Markdown](#)) paragraphs will be skipped in deploying as an application, as these are expected to contain human-readable documentation that is unsuitable for running as part of the resulting application.
- Paragraphs disabled for running within Zeppelin will be skipped in deploying as an application. Even if a disabled paragraph uses an incompatible interpreter, for example, `%flink.ipynlink` in a note with



`%flink` and `%flink.sql` interpreters, it will be skipped while deploying the note as an application, and will not result in an error.

- There must be at least one paragraph present with source code (Flink SQL, PyFlink or Flink Scala) that is enabled for running for the application deployment to succeed.
- Setting parallelism in the interpreter directive within a paragraph (e.g. `%flink.sql(parallelism=32)`) will be ignored in applications deployed from a note. Instead, you can update the deployed application through the AWS Management Console, AWS Command Line Interface or AWS API to change the Parallelism and/or ParallelismPerKPU settings according to the level of parallelism your application requires, or you can enable autoscaling for your deployed application.

## Scala/Python criteria

- Your Scala or Python code can't use a `BatchExecutionEnvironment` or `BatchTableEnvironment` (`benv`, `btenv`, `btenv_2` for Scala; `b_env`, `bt_env`, `bt_env_2` for Python).
- In your Scala or Python code, use the [Blink planner](#) (`senv`, `stenv` for Scala; `s_env`, `st_env` for Python) and not the older "Flink" planner (`stenv_2` for Scala, `st_env_2` for Python). The Apache Flink project recommends the use of the Blink planner for production use cases, and this is the default planner in Zeppelin and in Flink.
- Your Python paragraphs must not use [shell invocations/assignments](#) using `!` or [IPython magic commands](#) like `%timeit` or `%conda` in notes meant to be deployed as applications.
- You can't use Scala case classes as parameters of functions passed to higher-order dataflow operators like `map` and `filter`. For information about Scala case classes, see [CASE CLASSES](#) in the Scala documentation.
- You can deploy an application with a UDF written in Java or Scala, but not in Python.

## SQL criteria

- Simple SELECT statements are not permitted, as there's nowhere equivalent to a paragraph's output section where the data can be delivered.
- In any given paragraph, DDL statements (`USE`, `CREATE`, `ALTER`, `DROP`, `SET`, `RESET`) must precede DML (`INSERT`) statements. This is because DML statements in a paragraph must be submitted together as a single Flink job.
- There should be at most one paragraph that has DML statements in it. This is because, for the deploy-as-application feature, we only support submitting a single job to Flink.

## IAM permissions for Studio notebooks

Kinesis Data Analytics creates an IAM role for you when you create a Studio notebook through the AWS Management Console. It also associates with that role a policy that allows the following access:

Service	Access
CloudWatch Logs	List
Amazon EC2	List
AWS Glue	Read, Write
Kinesis Data Analytics	Read

Service	Access
Kinesis Data Analytics V2	Read
Amazon S3	Read, Write

## Connectors and dependencies

Connectors enable you to read and write data across various technologies. Kinesis Data Analytics bundles three default connectors with your Studio notebook. You can also use custom connectors. For more information about connectors, see [Table & SQL Connectors](#) in the Apache Flink documentation.

### Default connectors

If you use the AWS Management Console to create your Studio notebook, Kinesis Data Analytics includes the following custom connectors by default: `flink-sql-connector-flink`, `flink-connector-kafka_2.12` and `aws-msk-iam-auth`. To create a Studio notebook through the console without these custom connectors, choose the **Create with custom settings** option. Then, when you get to the **Configurations** page, clear the checkboxes next to the two connectors.

If you use the [CreateApplication](#) API to create your Studio notebook, the `flink-sql-connector-flink` and `flink-connector-kafka` connectors aren't included by default. To add them, specify them as a `MavenReference` in the `CustomArtifactsConfiguration` data type as shown in the following examples.

The `aws-msk-iam-auth` connector is the connector to use with Amazon MSK that includes the feature to automatically authenticate with IAM.

#### Note

The connector versions shown in the following example are the only versions that we support.

```
For the Kinesis connector:

"CustomArtifactsConfiguration": [{
  "ArtifactType": "DEPENDENCY_JAR",
  "MavenReference": {
    "GroupId": "org.apache.flink",
    "ArtifactId": "flink-sql-connector-kinesis-2.12",
    "Version": "1.13.2"
  }
}]

For the Apache MSK connector:

"CustomArtifactsConfiguration": [{
  "ArtifactType": "DEPENDENCY_JAR",
  "MavenReference": {
    "GroupId": "software.amazon.msk",
    "ArtifactId": "aws-msk-iam-auth",
    "Version": "1.1.0"
  }
}]

For the Apache Kafka connector:

"CustomArtifactsConfiguration": [{
  "ArtifactType": "DEPENDENCY_JAR",
  "MavenReference": {
```

```
"GroupId": "org.apache.flink",  
"ArtifactId": "flink-connector-kafka_2.12",  
"Version": "1.13.2"  
}  
}]
```

To add these connectors to an existing notebook, use the [UpdateApplication](#) API operation and specify them as a `MavenReference` in the `CustomArtifactsConfigurationUpdate` data type.

**Note**

You can set `failOnError` to true for the `flink-sql-connector-kinesis` connector in the table API.

## Dependencies and custom connectors

To use the AWS Management Console to add a dependency or a custom connector to your Studio notebook, follow these steps:

1. Upload your custom connector's file to Amazon S3.
2. In the AWS Management Console, choose the **Custom create** option for creating your Studio notebook.
3. Follow the Studio notebook creation workflow until you get to the **Configurations** step.
4. In the **Custom connectors** section, choose **Add custom connector**.
5. Specify the Amazon S3 location of the dependency or the custom connector.
6. Choose **Save changes**.

To add a dependency JAR or a custom connector when you create a new Studio notebook using the [CreateApplication](#) API, specify the Amazon S3 location of the dependency JAR or the custom connector in the `CustomArtifactsConfiguration` data type. To add a dependency or a custom connector to an existing Studio notebook, invoke the [UpdateApplication](#) API operation and specify the Amazon S3 location of the dependency JAR or the custom connector in the `CustomArtifactsConfigurationUpdate` data type.

**Note**

When you include a dependency or a custom connector, you must also include all its transitive dependencies that aren't bundled within it.

## User-defined functions

User-defined functions (UDFs) are extension points that allow you to call frequently-used logic or custom logic that can't be expressed otherwise in queries. You can use Python or a JVM language like Java or Scala to implement your UDFs in paragraphs inside your Studio notebook. You can also add to your Studio notebook external JAR files that contain UDFs implemented in a JVM language.

To use the console to add UDF JAR files to your Studio notebook, follow these steps:

1. Upload your UDF JAR file to Amazon S3.
2. In the AWS Management Console, choose the **Custom create** option for creating your Studio notebook.
3. Follow the Studio notebook creation workflow until you get to the **Configurations** step.
4. In the **User-defined functions** section, choose **Add user-defined function**.

5. Specify the Amazon S3 location of the JAR file or the ZIP file that has the implementation of your UDF.
6. Choose **Save changes**.

To add a UDF JAR when you create a new Studio notebook using the [CreateApplication](#) API, specify the JAR location in the `CustomArtifactConfiguration` data type. To add a UDF JAR to an existing Studio notebook, invoke the [UpdateApplication](#) API operation and specify the JAR location in the `CustomArtifactsConfigurationUpdate` data type. Alternatively, you can use the AWS Management Console to add UDF JAR files to your Studio notebook.

## Enabling Checkpointing

You enable checkpointing by using environment settings. For information about checkpointing, see [Fault Tolerance](#) in the [Kinesis Data Analytics Developer Guide](#).

### Setting the checkpointing interval

The following Scala code example sets your application's checkpoint interval to one minute:

```
// start a checkpoint every 1 minute
stenv.enableCheckpointing(60000)
```

The following Python code example sets your application's checkpoint interval to one minute:

```
st_env.get_config().get_configuration().set_string(
    "execution.checkpointing.interval", "1min"
)
```

### Setting the checkpointing type

The following Scala code example sets your application's checkpoint mode to `EXACTLY_ONCE` (the default):

```
// set mode to exactly-once (this is the default)
stenv.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE)
```

The following Python code example sets your application's checkpoint mode to `EXACTLY_ONCE` (the default):

```
st_env.get_config().get_configuration().set_string(
    "execution.checkpointing.mode", "EXACTLY_ONCE"
)
```

## Working with AWS Glue

Your Studio notebook stores and gets information about its data sources and sinks from AWS Glue. When you create your Studio notebook, you specify the AWS Glue database that contains your connection information. When you access your data sources and sinks, you specify AWS Glue tables

contained in the database. Your AWS Glue tables provide access to the AWS Glue connections that define the locations, schemas, and parameters of your data sources and destinations.

Studio notebooks use table properties to store application-specific data. For more information, see [Table properties \(p. 45\)](#).

For an example of how to set up a AWS Glue connection, database, and table for use with Studio notebooks, see [Create an AWS Glue Database \(p. 47\)](#) in the [Creating a Studio notebook Tutorial \(p. 46\)](#) tutorial.

## Table properties

In addition to data fields, your AWS Glue tables provide other information to your Studio notebook using table properties. Kinesis Data Analytics uses the following AWS Glue table properties:

- [Using Apache Flink time values \(p. 45\)](#): These properties define how Kinesis Data Analytics emits Apache Flink internal data processing time values.
- [Using Flink Connector and format properties \(p. 46\)](#): These properties provide information about your data streams.

To add a property to an AWS Glue table, do the following:

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. From the list of tables, choose the table that your application uses to store its data connection information. Choose **Action**, **Edit table details**.
3. Under **Table Properties**, enter `kinesisanalytics.proctime` for **key** and `user_action_time` for **Value**.

## Using Apache Flink time values

Apache Flink provides time values that describe when stream processing events occurred, such as [Processing Time](#) and [Event Time](#). To include these values in your application output, you define properties on your AWS Glue table that tell the Kinesis Data Analytics runtime to emit these values into the specified fields.

The keys and values you use in your table properties are as follows:

Timestamp Type	Key	Value
<a href="#">Processing Time</a>	<code>kinesisanalytics.proctime</code>	The column name that AWS Glue will use to expose the value. This column name does not correspond to an existing table column.
<a href="#">Event Time</a>	<code>kinesisanalytics.rowtime</code>	The column name that AWS Glue will use to expose the value. This column name corresponds to an existing table column.
	<code>kinesisanalytics.watermark.column</code>	The watermark interval in milliseconds

## Using Flink Connector and format properties

You provide information about your data sources to your application's Flink connectors using AWS Glue table properties. Some examples of the properties that Kinesis Data Analytics uses for connectors are as follows:

Connector Type	Key	Value
Kafka	<code>format</code>	The format used to deserialize and serialize Kafka messages, e.g. <code>json</code> or <code>csv</code> .
	<code>scan.startup.mode</code>	The startup mode for the Kafka consumer, e.g. <code>earliest-offset</code> or <code>timestamp</code> .
Kinesis	<code>format</code>	The format used to deserialize and serialize Kinesis data stream records, e.g. <code>json</code> or <code>csv</code> .
	<code>aws.region</code>	The AWS region where the stream is defined.
S3 (Filesystem)	<code>format</code>	The format used to deserialize and serialize files, e.g. <code>json</code> or <code>csv</code> .
	<code>path</code>	The Amazon S3 path, e.g. <code>s3://mybucket/</code> .

For more information about other connectors besides Kinesis and Apache Kafka, see your connector's documentation.

## Examples and tutorials

### Topics

- [Tutorial: Creating a Studio notebook in Kinesis Data Analytics \(p. 46\)](#)
- [Tutorial: Deploying as an application with durable state \(p. 60\)](#)
- [Examples \(p. 63\)](#)

## Tutorial: Creating a Studio notebook in Kinesis Data Analytics

The following tutorial demonstrates how to create a Studio notebook that reads data from a Kinesis Data Stream or an Amazon MSK cluster.

### This tutorial contains the following sections:

- [Setup \(p. 47\)](#)
- [Create an AWS Glue Database \(p. 47\)](#)
- [Next Steps \(p. 47\)](#)
- [Creating a Studio notebook with Kinesis Data Streams \(p. 47\)](#)

- [Creating a Studio notebook with Amazon MSK \(p. 51\)](#)
- [Cleaning up your application and dependent resources \(p. 59\)](#)

## Setup

Ensure that your AWS CLI is version 2 or later. To install the latest AWS CLI, see [Installing, updating, and uninstalling the AWS CLI version 2](#).

## Create an AWS Glue Database

Your Studio notebook uses an [AWS Glue](#) database for metadata about your Amazon MSK data source.

### Create an AWS Glue Database

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. Choose **Add database**. In the **Add database** window, enter **default** for **Database name**. Choose **Create**.

## Next Steps

With this tutorial, you can create a Studio notebook that uses either Kinesis Data Streams or Amazon MSK:

- [Kinesis Data Streams \(p. 47\)](#) : With Kinesis Data Streams, you quickly create an application that uses a Kinesis data stream as a source. You only need to create a Kinesis data stream as a dependent resource.
- [Amazon MSK \(p. 51\)](#) : With Amazon MSK, you create an application that uses a Amazon MSK cluster as a source. You need to create an Amazon VPC, an Amazon EC2 client instance, and an Amazon MSK cluster as dependent resources.

## Creating a Studio notebook with Kinesis Data Streams

This tutorial describes how to create a Studio notebook that uses a Kinesis data stream as a source.

### This tutorial contains the following sections:

- [Setup \(p. 47\)](#)
- [Create an AWS Glue table \(p. 48\)](#)
- [Create a Studio notebook with Kinesis Data Streams \(p. 48\)](#)
- [Send data to your Kinesis data stream \(p. 50\)](#)
- [Test your Studio notebook \(p. 51\)](#)

## Setup

Before you create a Studio notebook, create a Kinesis data stream (`ExampleInputStream`). Your application uses this stream for the application source.

You can create this stream 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*. Name the stream `ExampleInputStream` and set the **Number of open shards** to **1**.

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

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

## Create an AWS Glue table

Your Studio notebook uses an [AWS Glue](#) database for metadata about your Kinesis Data Streams data source.

### Note

You can either manually create the database first or you can let Kinesis Data Analytics create it for you when you create the notebook. Similarly, you can either manually create the table as described in this section, or you can use the create table connector code for Kinesis Data Analytics in your notebook within Apache Zeppelin to create your table via a DDL statement. You can then check in AWS Glue to make sure the table was correctly created.

### Create a Table

1. Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. If you don't already have a AWS Glue database, choose **Databases** from the left navigation bar. Choose **Add Database**. In the **Add database** window, enter **default** for **Database name**. Choose **Create**.
3. In the left navigation bar, choose **Tables**. In the **Tables** page, choose **Add tables**, **Add table manually**.
4. In the **Set up your table's properties** page, enter **stock** for the **Table name**. Make sure you select the database you created previously. Choose **Next**.
5. In the **Add a data store** page, choose **Kinesis**. For the **Stream name**, enter **ExampleInputStream**. For **Kinesis source URL**, choose enter **https://kinesis.us-east-1.amazonaws.com**. If you copy and paste the **Kinesis source URL**, be sure to delete any leading or trailing spaces. Choose **Next**.
6. In the **Classification** page, choose **JSON**. Choose **Next**.
7. In the **Define a Schema** page, choose **Add Column** to add a column. Add columns with the following properties:

Column name	Data type
<b>ticker</b>	<b>string</b>
<b>price</b>	<b>double</b>

Choose **Next**.

8. On the next page, verify your settings, and choose **Finish**.
9. Choose your newly created table from the list of tables.
10. Choose **Edit table** and add a property with the key `kinesisanalytics.proctime` and the value `proctime`.
11. Choose **Apply**.

## Create a Studio notebook with Kinesis Data Streams

Now that you have created the resources your application uses, you create your Studio notebook.



To create your application, you can use either the **AWS Management Console** or the **AWS CLI**.

- [Create a Studio notebook using the AWS Management Console \(p. 49\)](#)
- [Create a Studio notebook using the AWS CLI \(p. 49\)](#)

### Create a Studio notebook using the AWS Management Console

1. Open the Kinesis Data Analytics console at <https://console.aws.amazon.com/kinesisanalytics/home?region=us-east-1#/applications/dashboard>.
2. In the **Kinesis Data Analytics applications** page, choose the **Studio** tab. Choose **Create Studio notebook**.

#### Note

You can also create a Studio notebook from the Amazon MSK or Kinesis Data Streams consoles by selecting your input Amazon MSK cluster or Kinesis data stream, and choosing **Process data in real time**.

3. In the **Create Studio notebook** page, provide the following information:
  - Enter **MyNotebook** for the name of the notebook.
  - Choose **default** for **AWS Glue database**.

Choose **Create Studio notebook**.

4. In the **MyNotebook** page, choose **Run**. Wait for the **Status** to show **Running**. Charges apply when the notebook is running.

### Create a Studio notebook using the AWS CLI

To create your Studio notebook using the AWS CLI, do the following:

1. Verify your account ID. You need this value to create your application.
2. Create the role `arn:aws:iam::AccountID:role/ZeppelinRole` and add the following permissions to the auto-created role by console.

```
"kinesis:GetShardIterator",
"kinesis:GetRecords",
"kinesis:ListShards"
```
3. Create a file called `create.json` with the following contents. Replace the placeholder values with your information.

```
{
  "ApplicationName": "MyNotebook",
  "RuntimeEnvironment": "ZEPPELIN-FLINK-2_0",
  "ApplicationMode": "INTERACTIVE",
  "ServiceExecutionRole": "arn:aws:iam::AccountID:role/ZeppelinRole",
  "ApplicationConfiguration": {
    "ApplicationSnapshotConfiguration": {
      "SnapshotsEnabled": false
    },
    "ZeppelinApplicationConfiguration": {
      "CatalogConfiguration": {
        "GlueDataCatalogConfiguration": {
          "DatabaseARN": "arn:aws:glue:us-east-1:AccountID:database/default"
        }
      }
    }
  }
}
```

```
}  
}
```

4. Run the following command to create your application:

```
aws kinesisanalyticstv2 create-application --cli-input-json file://create.json
```

5. When the command completes, you see output that shows the details for your new Studio notebook. The following is an example of the output.

```
{  
  "ApplicationDetail": {  
    "ApplicationARN": "arn:aws:kinesisanalytics:us-east-1:012345678901:application/  
MyNotebook",  
    "ApplicationName": "MyNotebook",  
    "RuntimeEnvironment": "ZEPPELIN-FLINK-2_0",  
    "ApplicationMode": "INTERACTIVE",  
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/ZepppelinRole",  
    ...  
  }  
}
```

6. Run the following command to start your application. Replace the sample value with your account ID.

```
aws kinesisanalyticstv2 start-application --application-arn arn:aws:kinesisanalytics:us-  
east-1:012345678901:application/MyNotebook\
```

## Send data to your Kinesis data stream

To send test data to your Kinesis data stream, do the following:

1. Open the [Kinesis Data Generator](#).
2. Choose **Create a Cognito User with CloudFormation**.
3. The AWS CloudFormation console opens with the Kinesis Data Generator template. Choose **Next**.
4. In the **Specify stack details** page, enter a username and password for your Cognito user. Choose **Next**.
5. In the **Configure stack options** page, choose **Next**.
6. In the **Review Kinesis-Data-Generator-Cognito-User** page, choose the **I acknowledge that AWS CloudFormation might create IAM resources** checkbox. Choose **Create Stack**.
7. Wait for the AWS CloudFormation stack to finish being created. After the stack is complete, open the **Kinesis-Data-Generator-Cognito-User** stack in the AWS CloudFormation console, and choose the **Outputs** tab. Open the URL listed for the **KinesisDataGeneratorUrl** output value.
8. In the **Amazon Kinesis Data Generator** page, log in with the credentials you created in step 4.
9. On the next page, provide the following values:

<b>Region</b>	<b>us-east-1</b>
<b>Stream/delivery stream</b>	<b>ExampleInputStream</b>
<b>Records per second</b>	<b>1</b>

For **Record Template**, paste the following code:

```
{
```

```
"ticker": "{{random.arrayElement(
  ["AMZN", "MSFT", "GOOG"]
)}}",
"price": {{random.number(
  {
    "min": 10,
    "max": 150
  }
)}}
}
```

10. Choose **Send data**.
11. The generator will send data to your Kinesis data stream.

Leave the generator running while you complete the next section.

## Test your Studio notebook

In this section, you use your Studio notebook to query data from your Kinesis data stream.

1. Open the Kinesis Data Analytics console at <https://console.aws.amazon.com/kinesisanalytics/home?region=us-east-1#/applications/dashboard>.
2. On the **Kinesis Data Analytics applications** page, choose the **Studio notebook** tab. Choose **MyNotebook**.
3. In the **MyNotebook** page, choose **Open in Apache Zeppelin**.

The Apache Zeppelin interface opens in a new tab.

4. In the **Welcome to Zeppelin!** page, choose **Zeppelin Note**.
5. In the **Zeppelin Note** page, enter the following query into a new note:

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

Choose the run icon.

After a short time, the note displays data from the Kinesis data stream.

To open the Apache Flink Dashboard for your application to view operational aspects, choose **FLINK JOB**. For more information about the Flink Dashboard, see [Apache Flink Dashboard](#) in the [Kinesis Data Analytics Developer Guide](#).

For more examples of Flink Streaming SQL queries, see [Queries](#) in the [Apache Flink documentation](#).

## Creating a Studio notebook with Amazon MSK

This tutorial describes how to create a Studio notebook that uses an Amazon MSK cluster as a source.

**This tutorial contains the following sections:**

- [Setup \(p. 52\)](#)
- [Add a NAT Gateway to your VPC \(p. 52\)](#)
- [Create an AWS Glue Connection and Table \(p. 54\)](#)
- [Create a Studio notebook with Amazon MSK \(p. 55\)](#)
- [Send data to your Amazon MSK cluster \(p. 57\)](#)
- [Test your Studio notebook \(p. 58\)](#)

## Setup

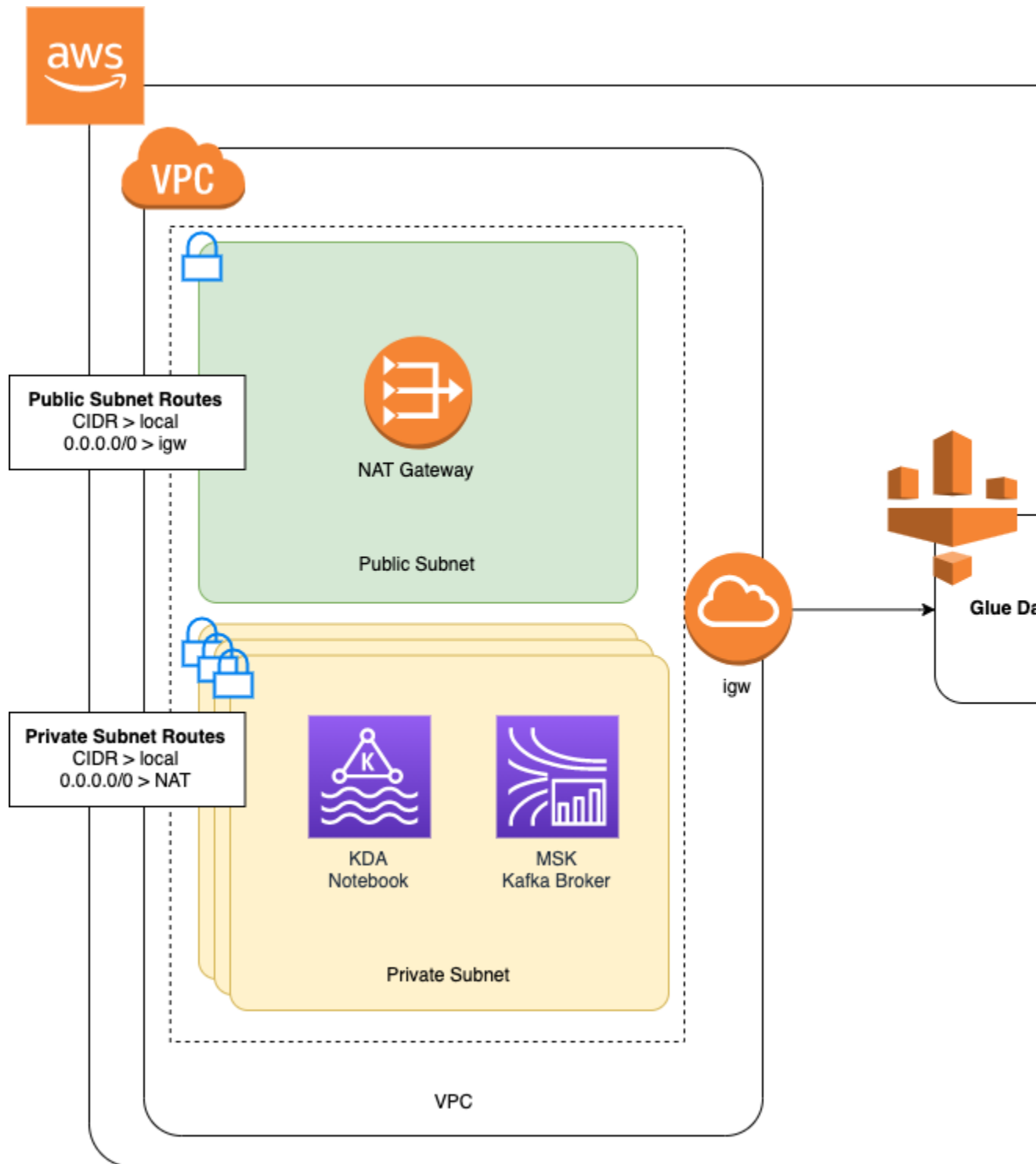
For this tutorial, you need an Amazon MSK cluster that allows plaintext access. If you don't have an Amazon MSK cluster set up already, follow the [Getting Started Using Amazon MSK](#) tutorial to create an Amazon VPC, an Amazon MSK cluster, a topic, and an Amazon EC2 client instance.

When following the tutorial, do the following:

- In [Step 3: Create an Amazon MSK Cluster](#), on step 4, change the `ClientBroker` value from `TLS` to **PLAINTEXT**.

## Add a NAT Gateway to your VPC

If you created an Amazon MSK cluster by following the [Getting Started Using Amazon MSK](#) tutorial, or if your existing Amazon VPC does not already have a NAT gateway for its private subnets, you must add a NAT Gateway to your Amazon VPC. The following diagram shows the architecture.



To create a NAT Gateway for your Amazon VPC, do the following:

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **NAT Gateways** from the left navigation bar.
3. On the **NAT Gateways** page, choose **Create NAT Gateway**.

- On the **Create NAT Gateway** page, provide the following values:

<b>Name - optional</b>	<b>ZeppelinGateway</b>
<b>Subnet</b>	<b>AWSKafkaTutorialSubnet1</b>
<b>Elastic IP allocation ID</b>	Choose an available Elastic IP. If there are no Elastic IPs available, choose <b>Allocate Elastic IP</b> , and then choose the Elastic IP that the console creates.

Choose **Create NAT Gateway**.

- On the left navigation bar, choose **Route Tables**.
  - Choose **Create Route Table**.
  - On the **Create route table** page, provide the following information:
    - Name tag:** **ZeppelinRouteTable**
    - VPC:** Choose your VPC (e.g. **AWSKafkaTutorialVPC**).
- Choose **Create**.
- In the list of route tables, choose **ZeppelinRouteTable**. Choose the **Routes** tab, and choose **Edit routes**.
  - In the **Edit Routes** page, choose **Add route**.
  - In the **For Destination**, enter **0.0.0.0/0**. For **Target**, choose **NAT Gateway, ZeppelinGateway**. Choose **Save Routes**. Choose **Close**.
  - On the Route Tables page, with **ZeppelinRouteTable** selected, choose the **Subnet associations** tab. Choose **Edit subnet associations**.
  - In the **Edit subnet associations** page, choose **AWSKafkaTutorialSubnet2** and **AWSKafkaTutorialSubnet3**. Choose **Save**.

## Create an AWS Glue Connection and Table

Your Studio notebook uses an [AWS Glue](#) database for metadata about your Amazon MSK data source. In this section, you create an AWS Glue connection that describes how to access your Amazon MSK cluster, and an AWS Glue table that describes how to present the data in your data source to clients such as your Studio notebook.

### Create a Connection

- Sign in to the AWS Management Console and open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
- If you don't already have a AWS Glue database, choose **Databases** from the left navigation bar. Choose **Add Database**. In the **Add database** window, enter **default** for **Database name**. Choose **Create**.
- Choose **Connections** from the left navigation bar. Choose **Add Connection**.
- In the **Add Connection** window, provide the following values:
  - For **Connection name**, enter **ZeppelinConnection**.
  - For **Connection type**, choose **Kafka**.
  - For **Kafka bootstrap server URLs**, provide the bootstrap broker string for your cluster. You can get the bootstrap brokers from either the MSK console, or by entering the following CLI command:

```
aws kafka get-bootstrap-brokers --region us-east-1 --cluster-arn ClusterArn
```

- Uncheck the **Require SSL connection** checkbox.

Choose **Next**.

5. In the **VPC** page, provide the following values:

- For **VPC**, choose the name of your VPC (e.g. **AWSKafkaTutorialVPC**.)
- For **Subnet**, choose **AWSKafkaTutorialSubnet2**.
- For **Security groups**, choose all available groups.

Choose **Next**.

6. In the **Connection properties / Connection access** page, choose **Finish**.

## Create a Table

### Note

You can either manually create the table as described in the following steps, or you can use the create table connector code for Kinesis Data Analytics in your notebook within Apache Zeppelin to create your table via a DDL statement. You can then check in AWS Glue to make sure the table was correctly created.

1. In the left navigation bar, choose **Tables**. In the **Tables** page, choose **Add tables, Add table manually**.
2. In the **Set up your table's properties** page, enter **stock** for the **Table name**. Make sure you select the database you created previously. Choose **Next**.
3. In the **Add a data store** page, choose **Kafka**. For the **Topic name**, enter your topic name (e.g. **AWSKafkaTutorialTopic**). For **Connection**, choose **ZeppelinConnection**.
4. In the **Classification** page, choose **JSON**. Choose **Next**.
5. In the **Define a Schema** page, choose **Add Column** to add a column. Add columns with the following properties:

Column name	Data type
<b>ticker</b>	<b>string</b>
<b>price</b>	<b>double</b>

Choose **Next**.

6. On the next page, verify your settings, and choose **Finish**.
7. Choose your newly created table from the list of tables.
8. Choose **Edit table** and add a property with the key `kinesisanalytics.proctime` and the value `proctime`.
9. Choose **Apply**.

## Create a Studio notebook with Amazon MSK

Now that you have created the resources your application uses, you create your Studio notebook.

You can create your application using either the **AWS Management Console** or the **AWS CLI**.

- [Create a Studio notebook using the AWS Management Console \(p. 56\)](#)
- [Create a Studio notebook using the AWS CLI \(p. 49\)](#)

#### Note

You can also create a Studio notebook from the Amazon MSK console by choosing an existing cluster, then choosing **Process data in real time**.

### Create a Studio notebook using the AWS Management Console

1. Open the Kinesis Data Analytics console at <https://console.aws.amazon.com/kinesisanalytics/home?region=us-east-1#/applications/dashboard>.
2. In the **Kinesis Data Analytics applications** page, choose the **Studio** tab. Choose **Create Studio notebook**.

#### Note

To create a Studio notebook from the Amazon MSK or Kinesis Data Streams consoles, select your input Amazon MSK cluster or Kinesis data stream, then choose **Process data in real time**.

3. In the **Create Studio notebook** page, provide the following information:
  - Enter **MyNotebook** for **Studio notebook Name**.
  - Choose **default** for **AWS Glue database**.

Choose **Create Studio notebook**.

4. In the **MyNotebook** page, choose the **Configuration** tab. In the **Networking** section, choose **Edit**.
5. In the **Edit networking for MyNotebook** page, choose **VPC configuration based on Amazon MSK cluster**. Choose your Amazon MSK cluster for **Amazon MSK Cluster**. Choose **Save changes**.
6. In the **MyNotebook** page, choose **Run**. Wait for the **Status** to show **Running**.

### Create a Studio notebook using the AWS CLI

To create your Studio notebook by using the AWS CLI, do the following:

1. Verify that you have the following information. You need these values to create your application.
  - Your account ID.
  - The subnet IDs and security group ID for the Amazon VPC that contains your Amazon MSK cluster.
2. Create a file called `create.json` with the following contents. Replace the placeholder values with your information.

```
{
  "ApplicationName": "MyNotebook",
  "RuntimeEnvironment": "ZEPPELIN-FLINK-2_0",
  "ApplicationMode": "INTERACTIVE",
  "ServiceExecutionRole": "arn:aws:iam::AccountID:role/ZeppelinRole",
  "ApplicationConfiguration": {
    "ApplicationSnapshotConfiguration": {
      "SnapshotsEnabled": false
    },
    "VpcConfigurations": [
      {
        "SubnetIds": [
          "SubnetID 1",
```



```
        "SubnetID 2",
        "SubnetID 3"
    ],
    "SecurityGroupIds": [
        "VPC Security Group ID"
    ]
},
],
"ZeppelinApplicationConfiguration": {
    "CatalogConfiguration": {
        "GlueDataCatalogConfiguration": {
            "DatabaseARN": "arn:aws:glue:us-east-1:AccountID:database/default"
        }
    }
}
}
```

3. Run the following command to create your application:

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

4. When the command completes, you should see output similar to the following, showing the details for your new Studio notebook:

```
{
  "ApplicationDetail": {
    "ApplicationARN": "arn:aws:kinesisanalytics:us-east-1:012345678901:application/MyNotebook",
    "ApplicationName": "MyNotebook",
    "RuntimeEnvironment": "ZEPPELIN-FLINK-2_0",
    "ApplicationMode": "INTERACTIVE",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/ZepplinRole",
    ...
  }
}
```

5. Run the following command to start your application. Replace the sample value with your account ID.

```
aws kinesisanalyticsv2 start-application --application-arn arn:aws:kinesisanalytics:us-east-1:012345678901:application/MyNotebook\
```

## Send data to your Amazon MSK cluster

In this section, you run a Python script in your Amazon EC2 client to send data to your Amazon MSK data source.

1. Connect to your Amazon EC2 client.
2. Run the following commands to install Python version 3, Pip, and the Kafka for Python package, and confirm the actions:

```
sudo yum install python37
curl -O https://bootstrap.pypa.io/get-pip.py
python3 get-pip.py --user
pip install kafka-python
```

3. Configure the AWS CLI on your client machine by entering the following command:

```
aws configure
```

Provide your account credentials, and **us-east-1** for the region.

4. Create a file called `stock.py` with the following contents. Replace the sample value with your Amazon MSK cluster's Bootstrap Brokers string, and update the topic name if your topic is not **AWSKafkaTutorialTopic**:

```
from kafka import KafkaProducer
import json
import random
from datetime import datetime

BROKERS = "<<Bootstrap Broker List>>"
producer = KafkaProducer(
    bootstrap_servers=BROKERS,
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    retry_backoff_ms=500,
    request_timeout_ms=20000,
    security_protocol='PLAINTEXT')

def getStock():
    data = {}
    now = datetime.now()
    str_now = now.strftime("%Y-%m-%d %H:%M:%S")
    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 = getStock()
    # print(data)
    try:
        future = producer.send("AWSKafkaTutorialTopic", value=data)
        producer.flush()
        record_metadata = future.get(timeout=10)
        print("sent event to Kafka! topic {} partition {} offset {}".format(record_metadata.topic, record_metadata.partition, record_metadata.offset))
    except Exception as e:
        print(e.with_traceback())
```

5. Run the script with the following command:

```
$ python3 stock.py
```

6. Leave the script running while you complete the following section.

## Test your Studio notebook

In this section, you use your Studio notebook to query data from your Amazon MSK cluster.

1. Open the Kinesis Data Analytics console at <https://console.aws.amazon.com/kinesisanalytics/home?region=us-east-1#/applications/dashboard>.
2. On the **Kinesis Data Analytics applications** page, choose the **Studio notebook** tab. Choose **MyNotebook**.
3. In the **MyNotebook** page, choose **Open in Apache Zeppelin**.

The Apache Zeppelin interface opens in a new tab.

4. In the **Welcome to Zeppelin!** page, choose **Zeppelin new note**.
5. In the **Zeppelin Note** page, enter the following query into a new note:

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

Choose the run icon.

The application displays data from the Amazon MSK cluster.

To open the Apache Flink Dashboard for your application to view operational aspects, choose **FLINK JOB**. For more information about the Flink Dashboard, see [Apache Flink Dashboard](#) in the [Kinesis Data Analytics Developer Guide](#).

For more examples of Flink Streaming SQL queries, see [Queries](#) in the [Apache Flink documentation](#).

## Cleaning up your application and dependent resources

### Delete your Studio notebook

1. Open the Kinesis Data Analytics console.
2. Choose **MyNotebook**.
3. Choose **Actions**, then **Delete**.

### Delete your AWS Glue Database and Connection

1. Open the AWS Glue console at <https://console.aws.amazon.com/glue/>.
2. Choose **Databases** from the left navigation bar. Check the checkbox next to **Default** to select it. Choose **Action**, **Delete Database**. Confirm your selection.
3. Choose **Connections** from the left navigation bar. Check the checkbox next to **ZeppelinConnection** to select it. Choose **Action**, **Delete Connection**. Confirm your selection.

### Delete your IAM role and policy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Roles** from the left navigation bar.
3. Use the search bar to search for the **ZeppelinRole** role.
4. Choose the **ZeppelinRole** role. Choose **Delete Role**. Confirm the deletion.

### Delete your CloudWatch log group

The console creates a CloudWatch Logs group and log stream for you when you create your application using the console. You do not have a log group and stream if you created your application using the AWS CLI.

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Log groups** from the left navigation bar.
3. Choose the **/aws/kinesis-analytics/MyNotebook** log group.
4. Choose **Actions**, **Delete log group(s)**. Confirm the deletion.

## Clean up Kinesis Data Streams Resources

To delete your Kinesis stream, open the Kinesis Data Streams console, select your Kinesis stream, and choose **Actions, Delete**.

## Clean up MSK resources

Follow the steps in this section if you created an Amazon MSK cluster for this tutorial. This section has directions for cleaning up your Amazon EC2 client instance, Amazon VPC, and Amazon MSK cluster.

### Delete your Amazon MSK Cluster

Follow these steps if you created an Amazon MSK cluster for this tutorial.

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose **AWSKafkaTutorialCluster**. Choose **Delete**. Enter **delete** in the window that appears, and confirm your selection.

### Terminate your client instance

Follow these steps if you created an Amazon EC2 client instance for this tutorial.

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Instances** from the left navigation bar.
3. Choose the checkbox next to **ZeppelinClient** to select it.
4. Choose **Instance State, Terminate Instance**.

### Delete your Amazon VPC

Follow these steps if you created an Amazon VPC for this tutorial.

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Network Interfaces** from the left navigation bar.
3. Enter your VPC ID in the search bar and press enter to search.
4. Select the checkbox in the table header to select all the displayed network interfaces.
5. Choose **Actions, Detach**. In the window that appears, choose **Enable** under **Force detachment**. Choose **Detach**, and wait for all of the network interfaces to reach the **Available** status.
6. Select the checkbox in the table header to select all the displayed network interfaces again.
7. Choose **Actions, Delete**. Confirm the action.
8. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
9. Select **AWSKafkaTutorialVPC**. Choose **Actions, Delete VPC**. Enter **delete** and confirm the deletion.

## Tutorial: Deploying as an application with durable state

The following tutorial demonstrates how to deploy a Studio notebook as a Kinesis Data Analytics application with durable state.

**This tutorial contains the following sections:**

- [Setup \(p. 61\)](#)

- [Deploy an application with durable state using the AWS Management Console \(p. 61\)](#)
- [Deploy an Application with Durable State Using the AWS CLI \(p. 62\)](#)

## Setup

Create a new Studio notebook by following the [Creating a Studio notebook Tutorial \(p. 46\)](#), using either Kinesis Data Streams or Amazon MSK. Name the Studio notebook `ExampleTestDeploy`.

## Deploy an application with durable state using the AWS Management Console

1. Add an S3 bucket location where you want the packaged code to be stored under **Application code location - optional** in the console. This enables the steps to deploy and run your application directly from the notebook.
2. Add required permissions to the application role to enable the role you are using to read and write to an Amazon S3 bucket, and to launch a Kinesis Data Analytics application:
  - `AmazonS3FullAccess`
  - `AmazonKinesisAnalyticsFullAccess`
  - Access to your sources, destinations, and VPCs as applicable. For more information, see [IAM permissions for Studio notebooks \(p. 41\)](#).
3. Use the following sample code:

```
%flink.ssql(type=update)
CREATE TABLE exampleoutput (
  'ticket' VARCHAR,
  'price' DOUBLE
)
WITH (
  'connector' = 'kinesis',
  'stream' = 'ExampleOutputStream',
  'aws.region' = 'us-east-1',
  'scan.stream.initpos' = 'LATEST',
  'format' = 'json'
);

INSERT INTO exampleoutput SELECT ticker, price FROM exampleinputstream
```

4. With this feature launch, you will see a new dropdown on the right top corner of each note in your notebook with the name of the notebook. You can do the following:
  - View the Studio notebook settings in the AWS Management Console.
  - Build your Zeppelin Note and export it to Amazon S3. At this point, provide a name for your application and choose **Build and Export**. You will get a notification when the export completes.
  - If you need to, you can view and run any additional tests on the executable in Amazon S3.
  - Once the build is complete, you will be able to deploy your code as a Kinesis streaming application with durable state and autoscaling.
  - Use the dropdown and choose **Deploy Zeppelin Note as Kinesis streaming application**. Review the application name and choose **Deploy via AWS Console**.
  - This will lead you to the AWS Management Console page for creating a Kinesis Data Analytics application. Note that application name, parallelism, code location, default Glue DB, VPC (if applicable) and IAM roles have been pre-populated. Validate that the IAM roles have the required permissions to your sources and destinations. Snapshots are enabled by default for durable application state management.

- Choose **create application**.
- You can choose **configure** and modify any settings, and choose **Run** to start your streaming application.

## Deploy an Application with Durable State Using the AWS CLI

To deploy an application using the AWS CLI, you must update your AWS CLI to use the service model provided with your Beta 2 information. For information about how to use the updated service model, see [Setup \(p. 47\)](#).

The following example code creates a new Studio notebook:

```
aws kinesisanalyticsv2 create-application \
  --application-name <app-name> \
  --runtime-environment ZEPPELIN-FLINK-2_0 \
  --application-mode INTERACTIVE \
  --service-execution-role <iam-role> \
  --application-configuration '{
    "ZeppelinApplicationConfiguration": {
      "CatalogConfiguration": {
        "GlueDataCatalogConfiguration": {
          "DatabaseARN": "arn:aws:glue:us-east-1:<account>:database/<glue-database-
name>"
        }
      }
    },
    "FlinkApplicationConfiguration": {
      "ParallelismConfiguration": {
        "ConfigurationType": "CUSTOM",
        "Parallelism": 4,
        "ParallelismPerKPU": 4
      }
    },
    "DeployAsApplicationConfiguration": {
      "S3ContentLocation": {
        "BucketARN": "arn:aws:s3:::<s3bucket>",
        "BasePath": "/something/"
      }
    },
    "VpcConfigurations": [
      {
        "SecurityGroupIds": [
          "<security-group>"
        ],
        "SubnetIds": [
          "<subnet-1>",
          "<subnet-2>"
        ]
      }
    ]
  }' \
  --region us-east-1
```

The following code example starts a Studio notebook:

```
aws kinesisanalyticsv2 start-application \
  --application-name <app-name> \
  --region us-east-1 \
  --no-verify-ssl
```

The following code returns the URL for an application's Apache Zeppelin notebook page:

```
aws kinesisanalyticsv2 create-application-presigned-url \
  --application-name <app-name> \
  --url-type ZEPPELIN_UI_URL \

  --region us-east-1 \
  --no-verify-ssl
```

## Examples

The following example queries demonstrate how to analyze data using window queries in a Studio notebook.

- [Tumbling window \(p. 63\)](#)
- [Sliding window \(p. 63\)](#)
- [Interactive SQL \(p. 64\)](#)
- [BlackHole SQL connector \(p. 64\)](#)
- [Data generator \(p. 65\)](#)
- [Interactive Scala \(p. 65\)](#)
- [Interactive Python \(p. 66\)](#)
- [Interactive Python, SQL, and Scala \(p. 67\)](#)
- [Cross-account Kinesis data stream \(p. 69\)](#)

For information about Apache Flink SQL query settings, see [Flink on Zeppelin Notebooks for Interactive Data Analysis](#).

To view your application in the Apache Flink dashboard, choose **FLINK JOB** in your application's **Zeppelin Note** page.

For more information about window queries, see [Windows](#) in the [Apache Flink documentation](#).

For more examples of Apache Flink Streaming SQL queries, see [Queries](#) in the [Apache Flink documentation](#).

## Tumbling window

The following Flink Streaming SQL query selects the highest price in each five-second tumbling window from the `ZeppelinTopic` table:

```
%flink.ssql(type=update)
SELECT TUMBLE_END(event_time, INTERVAL '5' SECOND) as winend, MAX(price) as
  five_second_high, ticker
FROM ZeppelinTopic
GROUP BY ticker, TUMBLE(event_time, INTERVAL '5' SECOND)
```

## Sliding window

The following Apache Flink Streaming SQL query selects the highest price in each five-second sliding window from the `ZeppelinTopic` table:

```
%flink.ssql(type=update)
SELECT HOP_END(event_time, INTERVAL '3' SECOND, INTERVAL '5' SECOND) AS winend, MAX(price)
  AS sliding_five_second_max
FROM ZeppelinTopic//or your table name in AWS Glue
GROUP BY HOP(event_time, INTERVAL '3' SECOND, INTERVAL '5' SECOND)
```

## Interactive SQL

This example prints the max of event time and processing time and the sum of values from the key-values table. Ensure that you have the sample data generation script from the [the section called "Data generator" \(p. 65\)](#) running. To try other SQL queries such as filtering and joins in your Studio notebook, see the Apache Flink documentation: [Queries](#) in the Apache Flink documentation.

```
%flink.ssql(type=single, parallelism=4, refreshInterval=1000, template=<h1>{2}</h1> records
seen until <h1>Processing Time: {1}</h1> and <h1>Event Time: {0}</h1>)

-- An interactive query prints how many records from the `key-value-stream` we have seen so
far, along with the current processing and event time.
SELECT
    MAX(`et`) as `et`,
    MAX(`pt`) as `pt`,
    SUM(`value`) as `sum`
FROM
    `key-values`
```

```
%flink.ssql(type=update, parallelism=4, refreshInterval=1000)

-- An interactive tumbling window query that displays the number of records observed per
(event time) second.
-- Browse through the chart views to see different visualizations of the streaming result.
SELECT
    TUMBLE_START(`et`, INTERVAL '1' SECONDS) as `window`,
    `key`,
    SUM(`value`) as `sum`
FROM
    `key-values`
GROUP BY
    TUMBLE(`et`, INTERVAL '1' SECONDS),
    `key`;
```

## BlackHole SQL connector

The BlackHole SQL connector doesn't require that you create a Kinesis data stream or an Amazon MSK cluster to test your queries. For information about the BlackHole SQL connector, see [BlackHole SQL Connector](#) in the Apache Flink documentation. In this example, the default catalog is an in-memory catalog.

```
%flink.ssql

CREATE TABLE default_catalog.default_database.blackhole_table (
    `key` BIGINT,
    `value` BIGINT,
    `et` TIMESTAMP(3)
) WITH (
    'connector' = 'blackhole'
)
```

```
%flink.ssql(parallelism=1)

INSERT INTO `test-target`
SELECT
    `key`,
    `value`,
    `et`
FROM
    `test-source`
```



```
WHERE
  `key` > 3
```

```
%flink.ssql(parallelism=2)

INSERT INTO `default_catalog`.`default_database`.`blackhole_table`
SELECT
  `key`,
  `value`,
  `et`
FROM
  `test-target`
WHERE
  `key` > 7
```

## Data generator

This example uses Scala to generate sample data. You can use this sample data to test various queries. Use the create table statement to create the key-values table.

```
import org.apache.flink.streaming.api.functions.source.datagen.DataGeneratorSource
import org.apache.flink.streaming.api.functions.source.datagen.RandomGenerator
import org.apache.flink.streaming.api.scala.DataStream

import java.sql.Timestamp

// ad-hoc convenience methods to be defined on Table
implicit class TableOps[T](table: DataStream[T]) {
  def asView(name: String): DataStream[T] = {
    if (stenv.listTemporaryViews.contains(name)) {
      stenv.dropTemporaryView("`" + name + "`")
    }
    stenv.createTemporaryView("`" + name + "`", table)
    return table;
  }
}
```

```
%flink(parallelism=4)
val stream = senv
  .addSource(new DataGeneratorSource(RandomGenerator.intGenerator(1, 10), 1000))
  .map(key => (key, 1, new Timestamp(System.currentTimeMillis)))
  .asView("key-values-data-generator")
```

```
%flink.ssql(parallelism=4)
-- no need to define the paragraph type with explicit parallelism (such as
"%flink.ssql(parallelism=2)")
-- in this case the INSERT query will inherit the parallelism of the of the above paragraph
INSERT INTO `key-values`
SELECT
  `_1` as `key`,
  `_2` as `value`,
  `_3` as `et`
FROM
  `key-values-data-generator`
```

## Interactive Scala

This is the Scala translation of the [the section called “Interactive SQL” \(p. 64\)](#). For more Scala examples, see [Table API](#) in the Apache Flink documentation.

```
%flink
import org.apache.flink.api.scala._
import org.apache.flink.table.api._
import org.apache.flink.table.api.bridge.scala._

// ad-hoc convenience methods to be defined on Table
implicit class TableOps(table: Table) {
  def asView(name: String): Table = {
    if (stenv.listTemporaryViews.contains(name)) {
      stenv.dropTemporaryView(name)
    }
    stenv.createTemporaryView(name, table)
    return table;
  }
}
```

```
%flink(parallelism=4)

// A view that computes many records from the `key-values` we have seen so far, along with
// the current processing and event time.
val query01 = stenv
  .from("`key-values`")
  .select(
    $"et".max().as("et"),
    $"pt".max().as("pt"),
    $"value".sum().as("sum")
  ).asView("query01")
```

```
%flink.sql(type=single, parallelism=16, refreshInterval=1000, template=<h1>{2}</h1>
  records seen until <h1>Processing Time: {1}</h1> and <h1>Event Time: {0}</h1>)

-- An interactive query prints the query01 output.
SELECT * FROM query01
```

```
%flink(parallelism=4)

// An tumbling window view that displays the number of records observed per (event time)
// second.
val query02 = stenv
  .from("`key-values`")
  .window(Tumble over 1.seconds on $"et" as $"w")
  .groupBy($"w", $"key")
  .select(
    $"w".start.as("window"),
    $"key",
    $"value".sum().as("sum")
  ).asView("query02")
```

```
%flink.sql(type=update, parallelism=4, refreshInterval=1000)

-- An interactive query prints the query02 output.
-- Browse through the chart views to see different visualizations of the streaming result.
SELECT * FROM `query02`
```

## Interactive Python

This is the Python translation of the [the section called “Interactive SQL” \(p. 64\)](#). For more Python examples, see [Table API](#) in the Apache Flink documentation.

```
%flink.pyflink
from pyflink.table.table import Table

def as_view(table, name):
    if (name in st_env.list_temporary_views()):
        st_env.drop_temporary_view(name)
    st_env.create_temporary_view(name, table)
    return table

Table.as_view = as_view
```

```
%flink.pyflink(parallelism=16)

# A view that computes many records from the `key-values` we have seen so far, along with
the current processing and event time
st_env \
    .from_path("`keyvalues`") \
    .select(", ".join([
        "max(et) as et",
        "max(pt) as pt",
        "sum(value) as sum"
    ])) \
    .as_view("query01")
```

```
%flink.ssql(type=single, parallelism=16, refreshInterval=1000, template=<h1>{2}</h1>
records seen until <h1>Processing Time: {1}</h1> and <h1>Event Time: {0}</h1>)

-- An interactive query prints the query01 output.
SELECT * FROM query01
```

```
%flink.pyflink(parallelism=16)

# A view that computes many records from the `key-values` we have seen so far, along with
the current processing and event time
st_env \
    .from_path("`key-values`") \
    .window(Tumble.over("1.seconds").on("et").alias("w")) \
    .group_by("w, key") \
    .select(", ".join([
        "w.start as window",
        "key",
        "sum(value) as sum"
    ])) \
    .as_view("query02")
```

```
%flink.ssql(type=update, parallelism=16, refreshInterval=1000)

-- An interactive query prints the query02 output.
-- Browse through the chart views to see different visualizations of the streaming result.
SELECT * FROM `query02`
```

## Interactive Python, SQL, and Scala

You can use any combination of SQL, Python, and Scala in your notebook for interactive analysis. In a Studio notebook that you plan to deploy as an application with durable state, you can use a combination of SQL and Scala. This example shows you the sections that are ignored and those that get deployed in the application with durable state.

```
%flink.ssql
```

```
CREATE TABLE `default_catalog`.`default_database`.`my-test-source` (  
  `key` BIGINT NOT NULL,  
  `value` BIGINT NOT NULL,  
  `et` TIMESTAMP(3) NOT NULL,  
  `pt` AS PROCTIME(),  
  WATERMARK FOR `et` AS `et` - INTERVAL '5' SECOND  
)  
WITH (  
  'connector' = 'kinesis',  
  'stream' = 'kda-notebook-example-test-source-stream',  
  'aws.region' = 'eu-west-1',  
  'scan.stream.initpos' = 'LATEST',  
  'format' = 'json',  
  'json.timestamp-format.standard' = 'ISO-8601'  
)
```

```
%flink.sql  
CREATE TABLE `default_catalog`.`default_database`.`my-test-target` (  
  `key` BIGINT NOT NULL,  
  `value` BIGINT NOT NULL,  
  `et` TIMESTAMP(3) NOT NULL,  
  `pt` AS PROCTIME(),  
  WATERMARK FOR `et` AS `et` - INTERVAL '5' SECOND  
)  
WITH (  
  'connector' = 'kinesis',  
  'stream' = 'kda-notebook-example-test-target-stream',  
  'aws.region' = 'eu-west-1',  
  'scan.stream.initpos' = 'LATEST',  
  'format' = 'json',  
  'json.timestamp-format.standard' = 'ISO-8601'  
)
```

```
%flink()  
  
// ad-hoc convenience methods to be defined on Table  
implicit class TableOps(table: Table) {  
  def asView(name: String): Table = {  
    if (stenv.listTemporaryViews.contains(name)) {  
      stenv.dropTemporaryView(name)  
    }  
    stenv.createTemporaryView(name, table)  
    return table;  
  }  
}
```

```
%flink(parallelism=1)  
val table = stenv  
  .from("`default_catalog`.`default_database`.`my-test-source`")  
  .select($"key", $"value", $"et")  
  .filter($"key" > 10)  
  .asView("query01")
```

```
%flink.sql(parallelism=1)  
  
-- forward data  
INSERT INTO `default_catalog`.`default_database`.`my-test-target`  
SELECT * FROM `query01`
```

```
%flink.sql(type=update, parallelism=1, refreshInterval=1000)
```

```
-- forward data to local stream (ignored when deployed as application)
SELECT * FROM `query01`
```

```
%flink
```

```
// tell me the meaning of life (ignored when deployed as application!)
print("42!")
```

## Cross-account Kinesis data stream

To use a Kinesis data stream that's in an account other than the account that has your Studio notebook, create a service execution role in the account where your Studio notebook is running and a role trust policy in the account that has the data stream. Use `aws.credentials.provider`, `aws.credentials.role.arn`, and `aws.credentials.role.sessionName` in the Kinesis connector in your create table DDL statement to create a table against the data stream.

Use the following service execution role for the Studio notebook account.

```
{
  "Sid": "AllowNotebookToAssumeRole",
  "Effect": "Allow",
  "Action": "sts:AssumeRole"
  "Resource": "*"
}
```

Use the `AmazonKinesisFullAccess` policy and the following role trust policy for the data stream account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<accountID>:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

Use the following paragraph for the create table statement.

```
%flink.ssql
CREATE TABLE test1 (
  name VARCHAR,
  age BIGINT
) WITH (
  'connector' = 'kinesis',
  'stream' = 'stream-assume-role-test',
  'aws.region' = 'us-east-1',
  'aws.credentials.provider' = 'ASSUME_ROLE',
  'aws.credentials.role.arn' = 'arn:aws:iam::<accountID>:role/stream-assume-role-test-role',
  'aws.credentials.role.sessionName' = 'stream-assume-role-test-session',
  'scan.stream.initpos' = 'TRIM_HORIZON',
  'format' = 'json'
)
```

# Comparing Studio notebooks and Kinesis Data Analytics for SQL Applications

Kinesis Data Analytics offers SQL support with [Kinesis Data Analytics for SQL Applications](#). The following features are available for Studio notebooks, but not in Kinesis Data Analytics for SQL Applications:

- Joining stream data between multiple Kinesis data streams, or between a Kinesis data stream and an Amazon MSK topic
- Real-time visualization of transformed data in a data stream
- Using Python scripts or Scala programs within the same application

## Troubleshooting

This section contains troubleshooting information for Studio notebooks.

### Stopping a stuck application

To stop an application that is stuck in a transient state, call the [StopApplication](#) action with the `Force` parameter set to `true`. For more information, see [Running Applications](#) in the [Kinesis Data Analytics Developer Guide](#).

### Canceling jobs

This section shows you how to cancel Apache Flink jobs that you can't get to from Apache Zeppelin. If you want to cancel such a job, go to the Apache Flink dashboard, copy the job ID, then use it in one of the following examples.

To cancel a single job:

```
%flink.pyflink
import requests

requests.patch("https://zeppelin-flink:8082/jobs/[job_id]", verify=False)
```

To cancel all running jobs:

```
%flink.pyflink
import requests

r = requests.get("https://zeppelin-flink:8082/jobs", verify=False)
jobs = r.json()['jobs']

for job in jobs:
    if (job["status"] == "RUNNING"):
        print(requests.patch("https://zeppelin-flink:8082/jobs/{}".format(job["id"]),
            verify=False))
```

To cancel all jobs:

```
%flink.pyflink
import requests
```

```
r = requests.get("https://zeppelin-flink:8082/jobs", verify=False)
jobs = r.json()['jobs']

for job in jobs:
    requests.patch("https://zeppelin-flink:8082/jobs/{}".format(job["id"]), verify=False)
```

## Restarting the Apache Flink interpreter

To restart the Apache Flink interpreter within your Studio notebook

1. Choose **Configuration** near the top right corner of the screen.
2. Choose **Interpreter**.
3. Choose **restart** and then **OK**.

## Appendix: Creating custom IAM policies

You normally use managed IAM policies to allow your application to access dependent resources. If you need finer control over your application's permissions, you can use a custom IAM policy. This section contains examples of custom IAM policies.

### Note

In the following policy examples, replace the placeholder text with your application's values.

**This topic contains the following sections:**

- [AWS Glue \(p. 71\)](#)
- [CloudWatch Logs \(p. 72\)](#)
- [Kinesis streams \(p. 72\)](#)
- [Amazon MSK clusters \(p. 74\)](#)

## AWS Glue

The following example policy grants permissions to access a AWS Glue database.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GlueTable",
      "Effect": "Allow",
      "Action": [
        "glue:GetConnection",
        "glue:GetTable",
        "glue:GetTables",
        "glue:GetDatabase",
        "glue:CreateTable",
        "glue:UpdateTable"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<accountId>:connection/*",
        "arn:aws:glue:<region>:<accountId>:table/<database-name>/*",
        "arn:aws:glue:<region>:<accountId>:database/<database-name>",
        "arn:aws:glue:<region>:<accountId>:database/hive",
        "arn:aws:glue:<region>:<accountId>:catalog"
      ]
    }
  ],
}
```

```
{
  "Sid": "GlueDatabase",
  "Effect": "Allow",
  "Action": "glue:GetDatabases",
  "Resource": "*"
}
```

## CloudWatch Logs

The following policy grants permissions to access CloudWatch Logs:

```
{
  "Sid": "ListCloudwatchLogGroups",
  "Effect": "Allow",
  "Action": [
    "logs:DescribeLogGroups"
  ],
  "Resource": [
    "arn:aws:logs:<region>:<accountId>:log-group:*"
  ]
},
{
  "Sid": "ListCloudwatchLogStreams",
  "Effect": "Allow",
  "Action": [
    "logs:DescribeLogStreams"
  ],
  "Resource": [
    "<logGroupArn>:log-stream:*"
  ]
},
{
  "Sid": "PutCloudwatchLogs",
  "Effect": "Allow",
  "Action": [
    "logs:PutLogEvents"
  ],
  "Resource": [
    "<logStreamArn>"
  ]
}
```

### Note

If you create your application using the console, the console adds the necessary policies to access CloudWatch Logs to your application role.

## Kinesis streams

Your application can use a Kinesis Stream for a source or a destination. Your application needs read permissions to read from a source stream, and write permissions to write to a destination stream.

The following policy grants permissions to read from a Kinesis Stream used as a source:

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



```
    "Effect": "Allow",
    "Action": "kinesis:ListShards",
    "Resource": "*"
  },
  {
    "Sid": "KinesisShardConsumption",
    "Effect": "Allow",
    "Action": [
      "kinesis:GetShardIterator",
      "kinesis:GetRecords",
      "kinesis:DescribeStream",
      "kinesis:DescribeStreamSummary",
      "kinesis:RegisterStreamConsumer",
      "kinesis:DeregisterStreamConsumer"
    ],
    "Resource": "arn:aws:kinesis:<region>:<accountId>:stream/<stream-name>"
  },
  {
    "Sid": "KinesisEfoConsumer",
    "Effect": "Allow",
    "Action": [
      "kinesis:DescribeStreamConsumer",
      "kinesis:SubscribeToShard"
    ],
    "Resource": "arn:aws:kinesis:<region>:<account>:stream/<stream-name>/consumer/*"
  }
]
```

The following policy grants permissions to write to a Kinesis Stream used as a destination:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "KinesisStreamSink",
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:DescribeStreamSummary",
        "kinesis:DescribeStream"
      ],
      "Resource": "arn:aws:kinesis:<region>:<accountId>:stream/<stream-name>"
    }
  ]
}
```

If your application accesses an encrypted Kinesis stream, you must grant additional permissions to access the stream and the stream's encryption key.

The following policy grants permissions to access an encrypted source stream and the stream's encryption key:

```
{
  "Sid": "ReadEncryptedKinesisStreamSource",
  "Effect": "Allow",
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": [
    "<inputStreamKeyArn>"
  ]
}
```

```
}  
,
```

The following policy grants permissions to access an encrypted destination stream and the stream's encryption key:

```
{  
  "Sid": "WriteEncryptedKinesisStreamSink",  
  "Effect": "Allow",  
  "Action": [  
    "kms:GenerateDataKey"  
  ],  
  "Resource": [  
    "<outputStreamKeyArn>"  
  ]  
}
```

## Amazon MSK clusters

To grant access to an Amazon MSK cluster, you grant access to the cluster's VPC. For policy examples for accessing an Amazon VPC, see [VPC Application Permissions](#).

# Getting Started with Amazon Kinesis Data Analytics for Apache Flink (DataStream API)

This section introduces you to the fundamental concepts of Kinesis Data Analytics for Apache Flink and the DataStream API. 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 Flink Application \(p. 75\)](#)
- [Prerequisites for Completing the Exercises \(p. 75\)](#)
- [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 76\)](#)
- [Step 2: Set Up the AWS Command Line Interface \(AWS CLI\) \(p. 78\)](#)
- [Step 3: Create and Run a Kinesis Data Analytics for Apache Flink Application \(p. 79\)](#)
- [Step 4: Clean Up AWS Resources \(p. 90\)](#)
- [Step 5: Next Steps \(p. 91\)](#)

## Components of a Kinesis Data Analytics for Flink Application

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

A Kinesis Data Analytics 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 \(p. 10\)](#).
- **Operators:** The application processes data by using one or more *operators*. An operator can transform, enrich, or aggregate data. For more information, see [DataStream API Operators \(p. 14\)](#).
- **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 \(p. 11\)](#).

After you create, compile, and package your application code, 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 11](#). 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
```

To get started, go to [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 76\)](#).

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

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

1. [Sign Up for AWS \(p. 76\)](#)
2. [Create an IAM User \(p. 76\)](#)

### Sign Up for AWS

When you sign up for AWS, your account is automatically signed up for all Amazon services, including 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

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

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

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

### Create an IAM User

Services in AWS, such as 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 account ID without any hyphens. For example, if your 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 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 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 \(AWS CLI\) \(p. 78\)](#)

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

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

### 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 [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 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 Apache Flink Application \(p. 79\)](#)

# Step 3: Create and Run a Kinesis Data Analytics for Apache Flink Application

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

**This section contains the following steps:**

- [Create Two Amazon Kinesis Data Streams \(p. 79\)](#)
- [Write Sample Records to the Input Stream \(p. 80\)](#)
- [Download and Examine the Apache Flink Streaming Java Code \(p. 81\)](#)
- [Compile the Application Code \(p. 81\)](#)
- [Upload the Apache Flink Streaming Java Code \(p. 82\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 82\)](#)
- [Next Step \(p. 90\)](#)

## Create Two Amazon Kinesis Data Streams

Before you create a Kinesis Data Analytics 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:

```
import datetime  
import json  
import random  
import boto3  
  
STREAM_NAME = "ExampleInputStream"  
  
def get_data():  
    return {  
        'EVENT_TIME': datetime.datetime.now().isoformat(),  
        'TICKER': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),  
        'PRICE': round(random.random() * 100, 2)}  
  
def generate(stream_name, kinesis_client):  
    while True:  
        data = get_data()  
        print(data)  
        kinesis_client.put_record(  
            StreamName=stream_name,  
            Data=json.dumps(data),  
            PartitionKey="partitionkey")  
  
if __name__ == '__main__':  
    generate(STREAM_NAME, boto3.client('kinesis'))
```

2. Later in the tutorial, you run the `stock.py` script to send data to the application.



```
$ 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:

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

2. Navigate to the `amazon-kinesis-data-analytics-java-examples/GettingStarted` 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 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:

```
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 \(p. 21\)](#).

## 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 \(p. 75\)](#).

### 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 -Dflink.version=1.13.2
```

- Use your development environment. See your development environment documentation for details.

**Note**

The provided source code relies on libraries from Java 11.

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. 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 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 \(AWS CLI\) \(p. 85\)](#)

## 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 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 version 1.13.2 (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 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (**012345678901**) with your account ID.

```
{
  "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"
      ],
      "Resource": [
```

```

        "arn:aws:logs:us-west-2:012345678901:log-group:*"
    ],
    {
        "Sid": "DescribeLogStreams",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeLogStreams"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:*"
        ]
    },
    {
        "Sid": "PutLogEvents",
        "Effect": "Allow",
        "Action": [
            "logs:PutLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        ]
    },
    {
        "Sid": "ReadInputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream"
    },
    {
        "Sid": "WriteOutputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleOutputStream"
    }
]
}

```

## 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 **Properties**, for **Group ID**, enter **ProducerConfigProperties**.
5. Enter the following application properties and values:

Key	Value
<b>flink.inputstream.initpos</b>	<b>LATEST</b>
<b>aws.region</b>	<b>us-west-2</b>

Key	Value
<b>AggregationEnabled</b>	<b>false</b>

- Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
- For **CloudWatch logging**, select the **Enable** check box.
- 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

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

## 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 Apache Flink 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.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "kinesis:DescribeStream",  
      "Resource": "arn:aws:kinesis:us-east-1:012345678901:stream/SourceStream",  
    },  
    {  
      "Effect": "Allow",  
      "Action": "kinesis:PutRecord",  
      "Resource": "arn:aws:kinesis:us-east-1:012345678901:stream/SinkStream",  
    },  
  ],  
}
```

```
"Statement": [
  {
    "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:*",
    "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream"
  },
  {
    "Sid": "WriteOutputStream",
    "Effect": "Allow",
    "Action": "kinesis:*",
    "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleOutputStream"
  }
]
```

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 Amazon 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 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" \(p. 85\)](#).

- 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 **KAReadSourceStreamWriteSinkStream** 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.

```
{
  "ApplicationName": "test",
  "ApplicationDescription": "my java test app",
  "RuntimeEnvironment": "FLINK-1_13",
  "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:

```
aws kinesisanalyticstv2 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`.

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

2. Execute the [StartApplication](#) action with the preceding request to start the application:

```
aws kinesisanalyticstv2 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`.

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

2. Execute the [StopApplication](#) action with the following request to stop the application:

```
aws kinesisanalyticstv2 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" \(p. 206\)](#).



## 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 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`.

```
{
  "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 file name, 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 [the section called "Create Two Amazon Kinesis Data Streams" \(p. 79\)](#) section.

```
{
  "ApplicationName": "test",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationCodeConfigurationUpdate": {
      "CodeContentUpdate": {
        "S3ContentLocationUpdate": {
          "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
          "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
          "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhyvpvDU"
        }
      }
    }
  }
}
```

## Next Step

[Step 4: Clean Up AWS Resources \(p. 90\)](#)

# 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 \(p. 90\)](#)
- [Delete Your Kinesis Data Streams \(p. 90\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 90\)](#)
- [Delete Your IAM Resources \(p. 91\)](#)
- [Delete Your CloudWatch Resources \(p. 91\)](#)
- [Next Step \(p. 91\)](#)

## Delete Your Kinesis Data Analytics Application

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Kinesis Data Analytics panel, choose **MyApplication**.
3. 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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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.

## Next Step

[Step 5: Next Steps \(p. 91\)](#)

## Step 5: Next Steps

Now that you've created and run a basic Kinesis Data Analytics application, see the following resources for more advanced Kinesis Data Analytics solutions.

- **The AWS Streaming Data Solution for Amazon Kinesis:** The AWS Streaming Data Solution for Amazon Kinesis automatically configures the AWS services necessary to easily capture, store, process, and deliver streaming data. The solution provides multiple options for solving streaming data use cases. The Kinesis Data Analytics option provides an end-to-end streaming ETL example demonstrating a real-world application that runs analytical operations on simulated New York taxi data. The solution sets up all necessary AWS resources such as IAM roles and policies, a CloudWatch dashboard, and CloudWatch alarms.
- **AWS Streaming Data Solution for Amazon MSK:** The AWS Streaming Data Solution for Amazon MSK provides AWS CloudFormation templates where data flows through producers, streaming storage, consumers, and destinations.
- **Clickstream Lab with Apache Flink and Apache Kafka:** An end to end lab for clickstream use cases using Amazon Managed Streaming for Apache Kafka for streaming storage and Amazon Kinesis Data Analytics for Apache Flink applications for stream processing.
- **Streaming Analytics Workshop:** In this workshop, you build an end-to-end streaming architecture to ingest, analyze, and visualize streaming data in near real-time. You set out to improve the operations of a taxi company in New York City. You analyze the telemetry data of a taxi fleet in New York City in near real-time to optimize their fleet operations.

- **Kinesis Data Analytics for Apache Flink: Examples (p. 113):** This section of this Developer Guide provides examples of creating and working with applications in Kinesis Data Analytics. They include example code and step-by-step instructions to help you create Kinesis Data Analytics applications and test your results.
- **Learn Flink: Hands On Training:** Official introductory Apache Flink training that gets you started writing scalable streaming ETL, analytics, and event-driven applications.

**Note**

Be aware that Kinesis Data Analytics does not support the Apache Flink version (1.12) used in this training. You can use Flink 1.13 in Flink Kinesis Data Analytics.

# Getting Started with Amazon Kinesis Data Analytics for Apache Flink (Table API)

This section introduces you to the fundamental concepts of Kinesis Data Analytics for Apache Flink and the Table API. 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 Flink Application \(p. 93\)](#)
- [Prerequisites \(p. 93\)](#)
- [Create and Run a Kinesis Data Analytics for Apache Flink Application \(p. 94\)](#)
- [Clean Up AWS Resources \(p. 101\)](#)
- [Next Steps \(p. 102\)](#)

## Components of a Kinesis Data Analytics for Flink Application

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

A Kinesis Data Analytics application has the following components:

- **Runtime properties:** You can use *runtime properties* to configure your application without recompiling your application code.
- **Table Source:** The application consumes data by using a source. A *source* connector reads data from a Kinesis data stream, an Amazon MSK topic, or similar. For more information, see [Table API Sources \(p. 15\)](#).
- **Functions:** The application processes data by using one or more functions. A *function* can transform, enrich, or aggregate data.
- **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 MSK topic, an Amazon S3 bucket, and so on. For more information, see [Table API Sinks \(p. 16\)](#).

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

## Prerequisites

Before starting this tutorial, complete the first two steps of the [Getting Started with Amazon Kinesis Data Analytics for Apache Flink \(DataStream API\) \(p. 75\)](#):

- [Step 1: Set Up an AWS Account and Create an Administrator User](#) (p. 76)
- [Step 2: Set Up the AWS Command Line Interface \(AWS CLI\)](#) (p. 78)

To get started, see [Create an Application](#) (p. 94).

## Create and Run a Kinesis Data Analytics for Apache Flink Application

In this exercise, you create a Kinesis Data Analytics application with an Amazon MSK topic as a source and an Amazon S3 bucket as a sink.

**This section contains the following steps.**

- [Create Dependent Resources](#) (p. 94)
- [Write Sample Records to the Input Stream](#) (p. 95)
- [Download and Examine the Apache Flink Streaming Java Code](#) (p. 96)
- [Compile the Application Code](#) (p. 97)
- [Upload the Apache Flink Streaming Java Code](#) (p. 97)
- [Create and Run the Kinesis Data Analytics Application](#) (p. 98)
- [Next Step](#) (p. 101)

### Create Dependent Resources

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

- A virtual private cloud (VPC) based on Amazon VPC and an Amazon MSK cluster
- An Amazon S3 bucket to store the application's code and output (`ka-app-<username>`)

### Create a VPC and an Amazon MSK Cluster

To create a VPC and Amazon MSK cluster to access from your Kinesis Data Analytics application, follow the [Getting Started Using Amazon MSK](#) tutorial.

When completing the tutorial, note the following:

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

```
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
{...
  "BootstrapBrokerStringTls": "b-2.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-1.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-3.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094"
}
```

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

## Create an Amazon S3 Bucket

You can create the Amazon S3 bucket using the console. For instructions for creating this resource, see the following topics:

- [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service User Guide*. Give the Amazon S3 bucket a globally unique name by appending your login name, such as **ka-app-*<username>***.

## Other Resources

When you create your application, Kinesis Data Analytics creates the following Amazon 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 Amazon MSK topic for the application to process.

1. Connect to the client instance you created in [Step 4: Create a Client Machine](#) of the [Getting Started Using Amazon MSK](#) tutorial.
2. Install Python3, Pip, and the Kafka Python library:

```
$ sudo yum install python37
$ curl -O https://bootstrap.pypa.io/get-pip.py
$ python3 get-pip.py --user
$ pip install kafka-python
```

3. Create a file named `stock.py` with the following contents. Replace the `BROKERS` value with your bootstrap broker list you recorded previously.

```
from kafka import KafkaProducer
import json
import random
from datetime import datetime

BROKERS = "<Bootstrap Brokers List>"
producer = KafkaProducer(
    bootstrap_servers=BROKERS,
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    retry_backoff_ms=500,
    request_timeout_ms=20000,
    security_protocol='PLAINTEXT')

def getStock():
    data = {}
    now = datetime.now()
    str_now = now.strftime("%Y-%m-%d %H:%M:%S")
    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 =getStock()
    # print(data)
    try:
        future = producer.send("AWSKafkaTutorialTopic", value=data)
        producer.flush()
        record_metadata = future.get(timeout=10)
        print("sent event to Kafka! topic {} partition {} offset
        {}".format(record_metadata.topic, record_metadata.partition, record_metadata.offset))
    except Exception as e:
        print(e.with_traceback())
```

4. Later in the tutorial, you run the `stock.py` script to send data to the application.

```
$ python3 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 Java application code

1. Clone the remote repository using the following command:

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

2. Navigate to the `amazon-kinesis-data-analytics-java-examples/GettingStartedTable` 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 libraries.
- The `StreamingJob.java` file contains the main method that defines the application's functionality.
- The application uses a `FlinkKafkaConsumer` to read from the Amazon MSK topic. The following snippet creates a `FlinkKafkaConsumer` object:

```
final FlinkKafkaConsumer<StockRecord> consumer = new
    FlinkKafkaConsumer<StockRecord>(kafkaTopic, new KafkaEventDeserializationSchema(),
    kafkaProps);
```

- Your application creates source and sink connectors to access external resources using `StreamExecutionEnvironment` and `TableEnvironment` objects.
- The application creates source and sink connectors using dynamic application properties, so you can specify your application parameters (such as your S3 bucket) without recompiling the code.

```
//read the parameters from the Kinesis Analytics environment
Map<String, Properties> applicationProperties =
    KinesisAnalyticsRuntime.getApplicationProperties();
Properties flinkProperties = null;

String kafkaTopic = parameter.get("kafka-topic", "AWSKafkaTutorialTopic");
String brokers = parameter.get("brokers", "");
String s3Path = parameter.get("s3Path", "");
```



```
if (applicationProperties != null) {
    flinkProperties = applicationProperties.get("FlinkApplicationProperties");
}

if (flinkProperties != null) {
    kafkaTopic = flinkProperties.get("kafka-topic").toString();
    brokers = flinkProperties.get("brokers").toString();
    s3Path = flinkProperties.get("s3Path").toString();
}
```

For more information about runtime properties, see [Runtime Properties \(p. 21\)](#).

## 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 \(p. 75\)](#).

### 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 -Dflink.version=1.13.2
```

- Use your development environment. See your development environment documentation for details.

#### Note

The provided source code relies on libraries from Java 11.

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 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

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 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 as **Apache Flink version 1.13.2 (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 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 Amazon S3 bucket.

#### To edit the IAM policy to add S3 bucket permissions

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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy.

```
{
  "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"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:*"
        ]
    },
    {
        "Sid": "DescribeLogStreams",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeLogStreams"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:*"
        ]
    },
    {
        "Sid": "PutLogEvents",
        "Effect": "Allow",
        "Action": [
            "logs:PutLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:kinesis-analytics-log-stream"
        ]
    },
    {
        "Sid": "WriteObjects",
        "Effect": "Allow",
        "Action": [
            "s3:Abort*",
            "s3:DeleteObject*",
            "s3:GetObject*",
            "s3:GetBucket*",
            "s3:List*",
            "s3:ListBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::ka-app-<username>",
            "arn:aws:s3:::ka-app-<username>/*"
        ]
    }
]
```

## Configure the Application

Use the following procedure to configure the application.

### To 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 **Properties**, choose **Create group**. For **Group ID**, enter **FlinkApplicationProperties**.
5. Enter the following application properties and values:

Key	Value
kafka-topic	AWSKafkaTutorialTopic
brokers	<i>Your Amazon MSK cluster's Bootstrap Brokers list</i>
s3Path	ka-app- <i>&lt;username&gt;</i>
security.protocol	SSL
ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
ssl.truststore.password	changeit

6. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
7. For **CloudWatch logging**, select the **Enable** check box.
8. In the **Virtual Private Cloud (VPC)** section, choose **VPC configuration based on Amazon MSK cluster**. Choose **AWSKafkaTutorialCluster**.
9. 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

Use the following procedure to run the application.

### To 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**.

3. From your Amazon EC2 client, run the Python script you created previously to write records to the Amazon MSK cluster for your application to process:

```
$ python3 stock.py
```

## Stop the Application

To stop the application, on the **MyApplication** page, choose **Stop**. Confirm the action.

## Next Step

[Clean Up AWS Resources \(p. 101\)](#)

# Clean Up AWS Resources

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

**This topic contains the following sections.**

- [Delete Your Kinesis Data Analytics Application \(p. 101\)](#)
- [Delete Your Amazon MSK Cluster \(p. 101\)](#)
- [Delete Your VPC \(p. 101\)](#)
- [Delete Your Amazon S3 Objects and Bucket \(p. 102\)](#)
- [Delete Your IAM Resources \(p. 102\)](#)
- [Delete Your CloudWatch Resources \(p. 102\)](#)
- [Next Step \(p. 102\)](#)

## Delete Your Kinesis Data Analytics Application

Use the following procedure to delete the application.

**To delete the application**

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Kinesis Data Analytics panel, choose **MyApplication**.
3. On the application page, choose **Delete** and then confirm the deletion.

## Delete Your Amazon MSK Cluster

To delete your Amazon MSK cluster, follow [Step 8: Delete the Amazon MSK Cluster](#) in the [Amazon Managed Streaming for Apache Kafka Developer Guide](#).

## Delete Your VPC

To delete your Amazon VPC, do the following:

- Open the Amazon VPC console.
- Choose your VPC.

- For **Actions**, choose **Delete VPC**.

## Delete Your Amazon S3 Objects and Bucket

Use the following procedure to delete your S3 objects and bucket.

### To delete your S3 objects 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

Use the following procedure to delete your IAM resources.

### To 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

Use the following procedure to delete your CloudWatch resources.

### To delete your CloudWatch resources

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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.

## Next Step

[Next Steps \(p. 102\)](#)

## Next Steps

Now that you've created and run a Kinesis Data Analytics application that uses the Table API, see [Step 5: Next Steps \(p. 91\)](#) in the [Getting Started with Amazon Kinesis Data Analytics for Apache Flink \(DataStream API\) \(p. 75\)](#).

# Getting Started with Amazon Kinesis Data Analytics for Apache Flink for Python

This section introduces you to the fundamental concepts of Kinesis Data Analytics for Apache Flink using Python and the Table API. 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 Flink Application \(p. 103\)](#)
- [Prerequisites \(p. 103\)](#)
- [Create and Run a Kinesis Data Analytics for Python Application \(p. 104\)](#)
- [Clean Up AWS Resources \(p. 111\)](#)

## Components of a Kinesis Data Analytics for Flink Application

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

A Kinesis Data Analytics application has the following components:

- **Runtime properties:** You can use *runtime properties* to configure your application without recompiling your application code.
- **Table Source:** The application consumes data by using a source. A *source* connector reads data from a Kinesis data stream, an Amazon MSK topic, or similar. For more information, see [Table API Sources \(p. 15\)](#).
- **Functions:** The application processes data by using one or more functions. A *function* can transform, enrich, or aggregate data.
- **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 MSK topic, an Amazon S3 bucket, and so on. For more information, see [Table API Sinks \(p. 16\)](#).

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

## Prerequisites

Before starting this tutorial, complete the first two steps of the [Getting Started with Amazon Kinesis Data Analytics for Apache Flink \(DataStream API\) \(p. 75\)](#):

- [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 76\)](#)

- [Step 2: Set Up the AWS Command Line Interface \(AWS CLI\) \(p. 78\)](#)

To get started, see [Create an Application \(p. 104\)](#).

## Create and Run a Kinesis Data Analytics for Python Application

In this exercise, you create a Kinesis Data Analytics application for Python application with an Kinesis stream as a source and a sink.

**This section contains the following steps.**

- [Create Dependent Resources \(p. 104\)](#)
- [Write Sample Records to the Input Stream \(p. 105\)](#)
- [Create and Examine the Apache Flink Streaming Python Code \(p. 106\)](#)
- [Upload the Apache Flink Streaming Python Code \(p. 107\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 108\)](#)
- [Next Step \(p. 111\)](#)

### Create Dependent Resources

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

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

### Create Two Kinesis Streams

Before you create a Kinesis Data Analytics 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
```

## Create an Amazon S3 Bucket

You can create the Amazon S3 bucket using the console. For instructions for creating this resource, see the following topics:

- [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service User Guide*. Give the Amazon S3 bucket a globally unique name by appending your login name, such as **ka-app-*<username>***.

## Other Resources

When you create your application, Kinesis Data Analytics creates the following Amazon 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 \(Boto\)](#).

### Note

The Python script in this section uses the AWS CLI. You must configure your AWS CLI to use your account credentials and default region. To configure your AWS CLI, enter the following:

```
aws configure
```

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

```
import datetime  
import json  
import random  
import boto3  
import time  
  
STREAM_NAME = "ExampleInputStream"  
  
def get_data():  
    return {  
        'event_time': datetime.datetime.now().isoformat(),  
        'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),  
        'price': round(random.random() * 100, 2)}  
  
def generate(stream_name, kinesis_client):  
    while True:  
        data = get_data()
```

```
print(data)
kinesis_client.put_record(
    StreamName=stream_name,
    Data=json.dumps(data),
    PartitionKey="partitionkey")
time.sleep(2)

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

2. Run the stock.py script:

```
$ python stock.py
```

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

## Create and Examine the Apache Flink Streaming Python Code

The Python 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
```

3. Navigate to the amazon-kinesis-data-analytics-java-examples/python/GettingStarted directory.

The application code is located in the getting-started.py file. Note the following about the application code:

- The application uses a Kinesis table source to read from the source stream. The following snippet calls the create\_table function to create the Kinesis table source:

```
table_env.execute_sql(
    create_table(input_table_name, input_stream, input_region, stream_initpos)
)
```

The create\_table function uses a SQL command to create a table that is backed by the streaming source:

```
def create_table(table_name, stream_name, region, stream_initpos):
    return """ CREATE TABLE {0} (
        ticker VARCHAR(6),
        price DOUBLE,
        event_time TIMESTAMP(3),
        WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
    )
    PARTITIONED BY (ticker)
    WITH (
        'connector' = 'kinesis',
        'stream' = '{1}',
```

```
'aws.region' = '{2}',  
'scan.stream.initpos' = '{3}',  
'sink.partition-field-delimiter' = ';',  
'sink.producer.collection-max-count' = '100',  
'format' = 'json',  
'json.timestamp-format.standard' = 'ISO-8601'  
) """.format(  
    table_name, stream_name, region, stream_initpos  
)
```

- The application creates two tables, then writes the contents of one table to the other.

```
# 2. Creates a source table from a Kinesis Data Stream  
table_env.execute_sql(  
    create_table(input_table_name, input_stream, input_region, stream_initpos)  
)  
  
# 3. Creates a sink table writing to a Kinesis Data Stream  
table_env.execute_sql(  
    create_table(output_table_name, output_stream, output_region, stream_initpos)  
)  
  
# 4. Inserts the source table data into the sink table  
table_result = table_env.execute_sql("INSERT INTO {0} SELECT * FROM {1}"  
    .format(output_table_name, input_table_name))
```

- The application uses the Flink connector, from the [flink-sql-connector-kinesis\\_2.12/1.13.2](#) file.

## Upload the Apache Flink Streaming Python Code

In this section, you create an Amazon S3 bucket and upload your application code.

### To upload the application code using the console:

1. Use your preferred compression application to compress the `streaming-file-sink.py` and [https://mvnrepository.com/artifact/org.apache.flink/flink-sql-connector-kinesis\\_2.12/1.13.2](https://mvnrepository.com/artifact/org.apache.flink/flink-sql-connector-kinesis_2.12/1.13.2) files. Name the archive `myapp.zip`. If you include the outer folder in your archive, you must include this in the path with the code in your configuration file(s): `GettingStarted/getting-started.py`.
2. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
3. Choose **Create bucket**.
4. 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**.
5. In the **Configure options** step, keep the settings as they are, and choose **Next**.
6. In the **Set permissions** step, keep the settings as they are, and choose **Next**.
7. Choose **Create bucket**.
8. In the Amazon S3 console, choose the `ka-app-code-<username>` bucket, and choose **Upload**.
9. In the **Select files** step, choose **Add files**. Navigate to the `myapp.zip` file that you created in the previous step. Choose **Next**.
10. You don't need to change any of the settings for the object, so choose **Upload**.

### To upload the application code using the AWS CLI:

1. Use your preferred compression application to compress the `streaming-file-sink.py` and [https://mvnrepository.com/artifact/org.apache.flink/flink-sql-connector-kinesis\\_2.12/1.13.2](https://mvnrepository.com/artifact/org.apache.flink/flink-sql-connector-kinesis_2.12/1.13.2) files.

Name the archive `myapp.zip`. If you include the outer folder in your archive, you must include this in the path with the code in your configuration file(s): `GettingStarted/getting-started.py`.

2. Run the following command:

```
$ aws s3 --region aws region cp myapp.zip s3://ka-app-code-<username>
```

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 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 as **Apache Flink version 1.13.2 (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 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

Use the following procedure to configure the application.

#### To 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 `myapp.zip`.
3. Under **Access to application resources**, for **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
4. Under **Properties**, choose **Add group**. For **Group ID**, enter `consumer.config.0`.
5. Enter the following application properties and values:

Key	Value
<code>input.stream.name</code>	<code>ExampleInputStream</code>
<code>aws.region</code>	<code>us-west-2</code>
<code>flink.stream.initpos</code>	<code>LATEST</code>

Choose **Save**.

- Under **Properties**, choose **Add group** again. For **Group ID**, enter `producer.config.0`.
- Enter the following application properties and values:

Key	Value
<code>output.stream.name</code>	<code>ExampleOutputStream</code>
<code>aws.region</code>	<code>us-west-2</code>
<code>shard.count</code>	<code>1</code>

- Under **Properties**, choose **Add group** again. For **Group ID**, enter `kinesis.analytics.flink.run.options`. This special property group tells your application where to find its code resources. For more information, see [Specifying your Code Files \(p. 19\)](#).
- Enter the following application properties and values:

Key	Value
<code>python</code>	<code>getting-started.py</code>
<code>jarfile</code>	<code>flink-sql-connector-kinesis_2.12-1.13.2.jar</code>

- Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
- For **CloudWatch logging**, choose the **Enable** check box.
- 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`

## Edit the IAM Policy

Edit the IAM policy to add permissions to access the Amazon S3 bucket.

### To edit the IAM policy to add S3 bucket permissions

- Open the IAM console at <https://console.aws.amazon.com/iam/>.
- Choose **Policies**. Choose the `kinesis-analytics-service-MyApplication-us-west-2` policy that the console created for you in the previous section.

3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (**012345678901**) with your account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::ka-app-code-username/myapp.zip"
      ]
    },
    {
      "Sid": "DescribeLogGroups",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
      ]
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:*"
      ]
    },
    {
      "Sid": "PutLogEvents",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:kinesis-analytics-log-stream"
      ]
    },
    {
      "Sid": "ReadInputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    },
    {
      "Sid": "WriteOutputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
    }
  ]
}
```

```
} ]
```

## Run the Application

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

## Stop the Application

To stop the application, on the **MyApplication** page, choose **Stop**. Confirm the action.

## Next Step

[Clean Up AWS Resources \(p. 111\)](#)

# Clean Up AWS Resources

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

**This topic contains the following sections.**

- [Delete Your Kinesis Data Analytics Application \(p. 111\)](#)
- [Delete Your Kinesis Data Streams \(p. 111\)](#)
- [Delete Your Amazon S3 Objects and Bucket \(p. 112\)](#)
- [Delete Your IAM Resources \(p. 112\)](#)
- [Delete Your CloudWatch Resources \(p. 112\)](#)

## Delete Your Kinesis Data Analytics Application

Use the following procedure to delete the application.

**To delete the application**

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Kinesis Data Analytics panel, choose **MyApplication**.
3. On the application 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 Objects and Bucket

Use the following procedure to delete your S3 objects and bucket.

### To delete your S3 objects 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

Use the following procedure to delete your IAM resources.

### To 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

Use the following procedure to delete your CloudWatch resources.

### To delete your CloudWatch resources

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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 Apache Flink: 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 \(DataStream API\)](#) (p. 75)

## 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

- [DataStream API Examples](#) (p. 113)
- [Python Examples](#) (p. 174)
- [Kinesis Data Analytics Solutions](#) (p. 195)

## DataStream API Examples

The following examples demonstrate how to create applications using the Apache Flink DataStream API.

## Topics

- [Example: Tumbling Window](#) (p. 113)
- [Example: Sliding Window](#) (p. 120)
- [Example: Writing to an Amazon S3 Bucket](#) (p. 126)
- [Tutorial: Using a Kinesis Data Analytics application to Replicate Data from One MSK Cluster to Another in a VPC](#) (p. 135)
- [Example: Use an EFO Consumer with a Kinesis Data Stream](#) (p. 139)
- [Example: Writing to Kinesis Data Firehose](#) (p. 146)
- [Example: Read From a Kinesis Stream in a Different Account](#) (p. 157)
- [Tutorial: Using a Custom Truststore with Amazon MSK](#) (p. 163)
- [Example: Using Apache Beam](#) (p. 168)

## Example: Tumbling Window

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

### Note

To set up required prerequisites for this exercise, first complete the [Getting Started \(DataStream API\) \(p. 75\)](#) exercise.

### This topic contains the following sections:

- [Create Dependent Resources \(p. 114\)](#)
- [Write Sample Records to the Input Stream \(p. 114\)](#)
- [Download and Examine the Application Code \(p. 115\)](#)
- [Compile the Application Code \(p. 116\)](#)
- [Upload the Apache Flink Streaming Java Code \(p. 116\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 116\)](#)
- [Clean Up AWS Resources \(p. 118\)](#)

## Create Dependent Resources

Before you create a Kinesis Data Analytics 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 User 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:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

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

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

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

2. Run the `stock.py` script:

```
$ 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
```

3. Navigate to the `amazon-kinesis-data-analytics-java-examples/TumblingWindow` directory.

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));
```

- Add the following import statement:

```
import
    org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows; //
flink 1.13
```

- 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 (Flink 1.11)
    .window(TumblingProcessingTimeWindows.of(Time.seconds(5))) //Flink 1.13
    .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 \(p. 75\)](#) in the [Getting Started \(DataStream API\) \(p. 75\)](#) tutorial.
2. Compile the application with the following command:

```
mvn package -Dflink.version=1.13.2
```

### Note

The provided source code relies on libraries from Java 11. If you are using a development environment,

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 \(p. 114\)](#) 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 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 uses Apache Flink version 1.13.2.

- Leave the version pulldown as **Apache Flink version 1.13.2 (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 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (`012345678901`) with your account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "logs:DescribeLogGroups",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*",
        "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-apps-1.0.jar"
      ]
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": "logs:DescribeLogStreams",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:*"
    },
    {
      "Sid": "PutLogEvents",
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
    },
    {
      "Sid": "ListCloudwatchLogGroups",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
      ]
    }
  ],
}
```

```
{
  "Sid": "ReadInputStream",
  "Effect": "Allow",
  "Action": "kinesis:*",
  "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
},
{
  "Sid": "WriteOutputStream",
  "Effect": "Allow",
  "Action": "kinesis:*",
  "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
}
]
```

## 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 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

1. On the **MyApplication** page, choose **Run**. Leave the **Run without snapshot** option selected, and confirm the action.
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 Tumbling Window tutorial.

**This topic contains the following sections:**

- [Delete Your Kinesis Data Analytics Application \(p. 119\)](#)
- [Delete Your Kinesis Data Streams \(p. 119\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 119\)](#)
- [Delete Your IAM Resources \(p. 119\)](#)
- [Delete Your CloudWatch Resources \(p. 119\)](#)

## 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. In the application's page, choose **Delete** and then confirm the deletion.

## Delete Your Kinesis Data Streams

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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 application that aggregates data using a sliding window.

### Note

To set up required prerequisites for this exercise, first complete the [Getting Started \(DataStream API\) \(p. 75\)](#) exercise.

**This topic contains the following sections:**

- [Create Dependent Resources \(p. 120\)](#)
- [Write Sample Records to the Input Stream \(p. 120\)](#)
- [Download and Examine the Application Code \(p. 121\)](#)
- [Compile the Application Code \(p. 122\)](#)
- [Upload the Apache Flink Streaming Java Code \(p. 122\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 122\)](#)
- [Clean Up AWS Resources \(p. 125\)](#)

## Create Dependent Resources

Before you create a Kinesis Data Analytics 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 streams `ExampleInputStream` and `ExampleOutputStream`.
- [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service User 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:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"
```



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

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

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

2. Run the stock.py script:

```
$ 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
```

3. Navigate to the amazon-kinesis-data-analytics-java-examples/SlidingWindow directory.

The application code is located in the `SlidingWindowStreamingJobWithParallelism.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 minimum value for each stock symbol over a 10-second window that slides by 5 seconds. The following code creates the operator and sends the aggregated data to a new Kinesis Data Streams sink:
- Add the following import statement:

```
import
    org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows; //
flink 1.13
```

- 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 (Flink 1.11)
      .window(TumblingProcessingTimeWindows.of(Time.seconds(5))) //Flink 1.13
      .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 \(p. 75\)](#) in the [Getting Started \(DataStream API\) \(p. 75\)](#) tutorial.
2. Compile the application with the following command:

```
mvn package -Dflink.version=1.13.2
```

### Note

The provided source code relies on libraries from Java 11. If you are using a development environment,

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 \(p. 120\)](#) 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

1. Open the Kinesis Data Analytics console at <https://console.aws.amazon.com/kinesisanalytics>.
2. On the 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 version 1.13.2 (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 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (**012345678901**) with your account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "logs:DescribeLogGroups",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*",
        "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-apps-1.0.jar"
      ]
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": "logs:DescribeLogStreams",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:*"
    },
    {
      "Sid": "PutLogEvents",
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
    },
    {
```

```
        "Sid": "ListCloudwatchLogGroups",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeLogGroups"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:*"
        ]
    },
    {
        "Sid": "ReadInputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    },
    {
        "Sid": "WriteOutputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
    }
]
```

## 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
  --current-application-version-id <VersionId>
  --application-configuration-update "{\"FlinkApplicationConfigurationUpdate\":
  { \"ParallelismConfigurationUpdate\": {\"ParallelismUpdate\": 5, \"ConfigurationTypeUpdate
\": \"CUSTOM\" } } }
```

You can retrieve the current application version ID using the [DescribeApplication](#) or [ListApplications](#) actions.

## Run the Application

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

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 \(p. 125\)](#)
- [Delete Your Kinesis Data Streams \(p. 125\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 125\)](#)
- [Delete Your IAM Resources \(p. 126\)](#)
- [Delete Your CloudWatch Resources \(p. 126\)](#)

### 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. In the application's page, choose **Delete** and then confirm the deletion.

### Delete Your Kinesis Data Streams

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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 an Amazon S3 Bucket

In this exercise, you create a Kinesis Data Analytics for Apache Flink 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 \(DataStream API\) \(p. 75\)](#) exercise.

### This topic contains the following sections:

- [Create Dependent Resources \(p. 126\)](#)
- [Write Sample Records to the Input Stream \(p. 127\)](#)
- [Download and Examine the Application Code \(p. 128\)](#)
- [Modify the Application Code \(p. 129\)](#)
- [Compile the Application Code \(p. 129\)](#)
- [Upload the Apache Flink Streaming Java Code \(p. 129\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 129\)](#)
- [Verify the Application Output \(p. 132\)](#)
- [Optional: Customize the Source and Sink \(p. 132\)](#)
- [Clean Up AWS Resources \(p. 134\)](#)

## Create Dependent Resources

Before you create a Kinesis Data Analytics for Apache Flink 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>**)

### Note

Kinesis Data Analytics for Apache Flink cannot write data to Amazon S3 with server-side encryption enabled on Kinesis Data Analytics.

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 User 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:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

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

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

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

2. Run the `stock.py` script:

```
$ 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
```

3. Navigate to the `amazon-kinesis-data-analytics-java-examples/S3Sink` directory.

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:

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

- You need to add the following import statement:

```
import org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows;
```

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

The sink reads messages in a tumbling window, encodes messages into S3 bucket objects, and sends the encoded objects to the S3 sink. The following code encodes objects for sending to Amazon S3:

```
input.map(value -> { // Parse the JSON  
    JsonNode jsonNode = jsonParser.readValue(value, JsonNode.class);  
    return new Tuple2<>(jsonNode.get("TICKER").toString(), 1);  
}).returns(Types.TUPLE(Types.STRING, Types.INT))  
    .keyBy(0) // Logically partition the stream for each word  
    // .timeWindow(Time.minutes(1)) // Tumbling window definition // Flink  
1.11  
    .window(TumblingProcessingTimeWindows.of(Time.minutes(1))) // Flink 1.13  
    .sum(1) // Count the appearances by ticker per partition  
    .map(value -> value.f0 + " count: " + value.f1.toString() + "\n")  
    .addSink(createS3SinkFromStaticConfig());  
  
env.execute("Flink S3 Streaming Sink Job");
```

### Note

The application uses a Flink `StreamingFileSink` object to write to Amazon S3. For more information about the `StreamingFileSink`, see [StreamingFileSink](#) in the [Apache Flink documentation](#).



## 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 user name to specify the application's output location:

```
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 \(p. 75\)](#) in the [Getting Started \(DataStream API\) \(p. 75\)](#) tutorial.
2. Compile the application with the following command:

```
mvn package -Dflink.version=1.13.2
```

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 11. If you are using a development environment,

## 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 \(p. 126\)](#) 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

1. Open the Kinesis Data Analytics console at <https://console.aws.amazon.com/kinesisanalytics>.
2. On the 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 version 1.13.2 (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 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:

- For **Application name**, enter **MyApplication**.
- For **Runtime**, choose **Apache Flink**.
- Leave the version as **Apache Flink version 1.13.2 (Recommended version)**.

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

7. Choose **Create application**.

**Note**

When you create a Kinesis Data Analytics for Apache Flink 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
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 user name.

```
{
    "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": [
    "arn:aws:logs:region:account-id:log-group:%LOG_GROUP_PLACEHOLDER%:log-
stream:*"
  ]
},
{
  "Sid": "PutCloudwatchLogs",
  "Effect": "Allow",
  "Action": [
    "logs:PutLogEvents"
  ],
  "Resource": [
    "arn:aws:logs:region:account-id:log-group:%LOG_GROUP_PLACEHOLDER%:log-
stream:%LOG_STREAM_PLACEHOLDER%"
  ]
}
,
{
  "Sid": "ReadInputStream",
  "Effect": "Allow",
  "Action": "kinesis:*",
  "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
},
{
  "Sid": "WriteObjects",
  "Effect": "Allow",
  "Action": [
    "s3:Abort*",
    "s3:DeleteObject*",
    "s3:GetObject*",
    "s3:GetBucket*",
    "s3:List*",
    "s3:ListBucket",
    "s3:PutObject"
  ],
  "Resource": [
    "arn:aws:s3:::ka-app-<username>",
    "arn:aws:s3:::ka-app-<username>/*"
  ]
}
]
```

## 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**.
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`

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

1. On the **MyApplication** page, choose **Run**. Leave the **Run without snapshot** option selected, and confirm the action.
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.

## Optional: Customize the Source and Sink

In this section, you customize settings on the source and sink objects.

**Note**

After changing the code sections described in the sections following, do the following to reload the application code:

- Repeat the steps in the [the section called “Compile the Application Code” \(p. 129\)](#) section to compile the updated application code.
- Repeat the steps in the [the section called “Upload the Apache Flink Streaming Java Code” \(p. 129\)](#) section to upload the updated application code.
- On the application's page in the console, choose **Configure** and then choose **Update** to reload the updated application code into your application.

**This section contains the following sections:**

- [Configure Data Partitioning \(p. 132\)](#)
- [Configure Read Frequency \(p. 133\)](#)
- [Configure Write Buffering \(p. 133\)](#)

## Configure Data Partitioning

In this section, you configure the names of the folders that the streaming file sink creates in the S3 bucket. You do this by adding a bucket assigner to the streaming file sink.

To customize the folder names created in the S3 bucket, do the following:

1. Add the following import statements to the beginning of the `S3StreamingSinkJob.java` file:

```
import
    org.apache.flink.streaming.api.functions.sink.filesystem.rollingpolicies.DefaultRollingPolicy;
import
    org.apache.flink.streaming.api.functions.sink.filesystem.bucketassigners.DateTimeBucketAssigner;
```

2. Update the `createS3SinkFromStaticConfig()` method in the code to look like the following:

```
private static StreamingFileSink<String> createS3SinkFromStaticConfig() {

    final StreamingFileSink<String> sink = StreamingFileSink
        .forRowFormat(new Path(s3SinkPath), new SimpleStringEncoder<String>("UTF-8"))
        .withBucketAssigner(new DateTimeBucketAssigner("yyyy-MM-dd--HH"))
        .withRollingPolicy(DefaultRollingPolicy.create().build())
        .build();
    return sink;
}
```

The preceding code example uses the `DateTimeBucketAssigner` with a custom date format to create folders in the S3 bucket. The `DateTimeBucketAssigner` uses the current system time to create bucket names. If you want to create a custom bucket assigner to further customize the created folder names, you can create a class that implements [BucketAssigner](#). You implement your custom logic by using the `getBucketId` method.

A custom implementation of `BucketAssigner` can use the [Context](#) parameter to obtain more information about a record in order to determine its destination folder.

## Configure Read Frequency

In this section, you configure the frequency of reads on the source stream.

The Kinesis Streams consumer reads from the source stream five times per second by default. This frequency will cause issues if there is more than one client reading from the stream, or if the application needs to retry reading a record. You can avoid these issues by setting the read frequency of the consumer.

To set the read frequency of the Kinesis consumer, you set the `SHARD_GETRECORDS_INTERVAL_MILLIS` setting.

The following code example sets the `SHARD_GETRECORDS_INTERVAL_MILLIS` setting to one second:

```
kinesisConsumerConfig.setProperty(ConsumerConfigConstants.SHARD_GETRECORDS_INTERVAL_MILLIS,
    "1000");
```

## Configure Write Buffering

In this section, you configure the write frequency and other settings of the sink.

By default, the application writes to the destination bucket every minute. You can change this interval and other settings by configuring the `DefaultRollingPolicy` object.

### Note

The Apache Flink streaming file sink writes to its output bucket every time the application creates a checkpoint. The application creates a checkpoint every minute by default. To increase the write interval of the S3 sink, you must also increase the checkpoint interval.

To configure the `DefaultRollingPolicy` object, do the following:

1. Increase the application's CheckpointInterval setting. The following input for the [UpdateApplication](#) action sets the checkpoint interval to 10 minutes:

```
{
  "ApplicationConfigurationUpdate": {
    "FlinkApplicationConfigurationUpdate": {
      "CheckpointConfigurationUpdate": {
        "ConfigurationTypeUpdate" : "CUSTOM",
        "CheckpointIntervalUpdate": 600000
      }
    }
  },
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 5
}
```

To use the preceding code, specify the current application version. You can retrieve the application version by using the [ListApplications](#) action.

2. Add the following import statement to the beginning of the `S3StreamingSinkJob.java` file:

```
import java.util.concurrent.TimeUnit;
```

3. Update the `createS3SinkFromStaticConfig` method in the `S3StreamingSinkJob.java` file to look like the following:

```
private static StreamingFileSink<String> createS3SinkFromStaticConfig() {

    final StreamingFileSink<String> sink = StreamingFileSink
        .forRowFormat(new Path(s3SinkPath), new
SimpleStringEncoder<String>("UTF-8"))
        .withBucketAssigner(new DateTimeBucketAssigner("yyyy-MM-dd--HH"))
        .withRollingPolicy(
            DefaultRollingPolicy.create()
                .withRolloverInterval(TimeUnit.MINUTES.toMillis(8))
                .withInactivityInterval(TimeUnit.MINUTES.toMillis(5))
                .withMaxPartSize(1024 * 1024 * 1024)
                .build())
        .build();
    return sink;
}
```

The preceding code example sets the frequency of writes to the Amazon S3 bucket to 8 minutes.

For more information about configuring the Apache Flink streaming file sink, see [Row-encoded Formats](#) in the [Apache Flink documentation](#).

## Clean Up AWS Resources

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

**This topic contains the following sections:**

- [Delete Your Kinesis Data Analytics Application \(p. 135\)](#)
- [Delete Your Kinesis Data Stream \(p. 135\)](#)
- [Delete Your Amazon S3 Objects and Bucket \(p. 135\)](#)
- [Delete Your IAM Resources \(p. 135\)](#)
- [Delete Your CloudWatch Resources \(p. 135\)](#)

## 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. On the application's page, choose **Delete** and then confirm the deletion.

## Delete Your Kinesis Data Stream

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Kinesis Data Streams panel, choose **ExampleInputStream**.
3. On 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. On 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. On 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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. On 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 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 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 \(DataStream API\) \(p. 75\)](#) exercise.

**This tutorial contains the following sections:**

- [Create an Amazon VPC with an Amazon MSK cluster \(p. 136\)](#)
- [Create the Application Code \(p. 136\)](#)
- [Upload the Apache Flink Streaming Java Code \(p. 137\)](#)
- [Create the Application \(p. 137\)](#)
- [Configure the Application \(p. 137\)](#)
- [Run the Application \(p. 139\)](#)
- [Test the Application \(p. 139\)](#)

## 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`:

```
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):

```
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn  
{...  
  "BootstrapBrokerStringTls": "b-2.awskafkatutorialcluste.t79r6y.c4.kafka.us-  
west-2.amazonaws.com:9094,b-1.awskafkatutorialcluste.t79r6y.c4.kafka.us-  
west-2.amazonaws.com:9094,b-3.awskafkatutorialcluste.t79r6y.c4.kafka.us-  
west-2.amazonaws.com:9094"  
}
```

- 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. We recommend using Java 11.

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
```

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 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:



```
mvn package -Dflink.version=1.13.2
```

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

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

#### Note

The provided source code relies on libraries from Java 11. If you are using a development environment,

## 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 \(DataStream API\) \(p. 75\)](#) tutorial.

#### Note

If you deleted the Amazon S3 bucket from the Getting Started tutorial, follow the [the section called "Upload the Apache Flink Streaming Java Code" \(p. 82\)](#) step again.

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 `KafkaGettingStartedJob-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 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 version 1.13.2**.
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 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**.

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

**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:

Key	Value
topic	AWSKafkaTutorialTopic
bootstrap.servers	<i>The bootstrap server list you saved previously</i>
security.protocol	SSL
ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
ssl.truststore.password	changeit

**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:

Key	Value
topic	AWSKafkaTutorialTopicDestination
bootstrap.servers	<i>The bootstrap server list you saved previously</i>
security.protocol	SSL
ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
ssl.truststore.password	changeit
transaction.timeout.ms	1000

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 \(p. 21\)](#).

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

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

## 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 \(p. 312\)](#) section in the [Troubleshooting \(p. 309\)](#) topic.

## Example: Use an EFO Consumer with a Kinesis Data Stream

In this exercise, you create a Kinesis Data Analytics application that reads from a Kinesis Data Stream using an [Enhanced Fan-Out \(EFO\)](#) consumer. If a Kinesis consumer uses EFO, the Kinesis Data Streams service gives it its own dedicated bandwidth, rather than having the consumer share the fixed bandwidth of the stream with the other consumers reading from the stream.

For more information about using EFO with the Kinesis consumer, see [FLIP-128: Enhanced Fan Out for Kinesis Consumers](#).

The application you create in this example uses AWS Kinesis Connector (flink-connector-kinesis) 1.13.2.

#### Note

To set up required prerequisites for this exercise, first complete the [Getting Started \(DataStream API\) \(p. 75\)](#) exercise.

**This topic contains the following sections:**

- [Create Dependent Resources](#) (p. 140)
- [Write Sample Records to the Input Stream](#) (p. 140)
- [Download and Examine the Application Code](#) (p. 141)
- [Compile the Application Code](#) (p. 141)
- [Upload the Apache Flink Streaming Java Code](#) (p. 142)
- [Create and Run the Kinesis Data Analytics Application](#) (p. 142)
- [Clean Up AWS Resources](#) (p. 145)

## Create Dependent Resources

Before you create a Kinesis Data Analytics 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 User 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:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

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

def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
```

```
print(data)
kinesis_client.put_record(
    StreamName=stream_name,
    Data=json.dumps(data),
    PartitionKey="partitionkey")

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

2. Run the `stock.py` script:

```
$ 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
```

3. Navigate to the `amazon-kinesis-data-analytics-java-examples/EfoConsumer` directory.

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

- You enable the EFO consumer by setting the following parameters on the Kinesis consumer:
  - **RECORD\_PUBLISHER\_TYPE:** Set this parameter to **EFO** for your application to use an EFO consumer to access the Kinesis Data Stream data.
  - **EFO\_CONSUMER\_NAME:** Set this parameter to a string value that is unique among the consumers of this stream. Re-using a consumer name in the same Kinesis Data Stream will cause the previous consumer using that name to be terminated.
- The following code example demonstrates how to assign values to the consumer configuration properties to use an EFO consumer to read from the source stream:

```
consumerConfig.putIfAbsent(RECORD_PUBLISHER_TYPE, "EFO");
consumerConfig.putIfAbsent(EFO_CONSUMER_NAME, "basic-efo-flink-app");
```

## 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 \(p. 75\)](#) in the [Getting Started \(DataStream API\) \(p. 75\)](#) tutorial.
2. Compile the application with the following command:

```
mvn package -Dflink.version=1.13.2
```

### Note

The provided source code relies on libraries from Java 11. If you are using a development environment,

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. 140) 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 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**.
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 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (**012345678901**) with your account ID.

#### Note

These permissions grant the application the ability to access the EFO consumer.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "logs:DescribeLogGroups",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*",
        "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-apps-1.0.jar"
      ]
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": "logs:DescribeLogStreams",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:*"
    },
    {
      "Sid": "PutLogEvents",
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
    },
    {
      "Sid": "ListCloudwatchLogGroups",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
      ]
    },
    {
      "Sid": "AllStreams",
      "Effect": "Allow",
      "Action": [
        "kinesis:ListShards",
        "kinesis:ListStreamConsumers",
        "kinesis:DescribeStreamSummary"
      ],
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/*"
    },
    {
      "Sid": "Stream",
      "Effect": "Allow",
      "Action": [
```

```

        "kinesis:DescribeStream",
        "kinesis:RegisterStreamConsumer",
        "kinesis:DeregisterStreamConsumer"
    ],
    "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    },
    {
        "Sid": "WriteOutputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
    },
    {
        "Sid": "Consumer",
        "Effect": "Allow",
        "Action": [
            "kinesis:DescribeStreamConsumer",
            "kinesis:SubscribeToShard"
        ],
        "Resource": [
            "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream/
consumer/my-flink-efo-consumer",
            "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream/
consumer/my-flink-efo-consumer:*"
        ]
    }
]
}

```

## 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 **Properties**, choose **Create Group**. For **Group ID**, enter **ConsumerConfigProperties**.
5. Enter the following application properties and values:

Key	Value
<b>flink.stream.recordpublisher</b>	<b>EFO</b>
<b>flink.stream.efo.consumername</b>	<b>my-flink-efo-consumer</b>
<b>INPUT_STREAM</b>	<b>ExampleInputStream</b>
<b>flink.inputstream.initpos</b>	<b>LATEST</b>
<b>AWS_REGION</b>	<b>us-west-2</b>

6. Under **Properties**, choose **Create Group**. For **Group ID**, enter **ProducerConfigProperties**.
7. Enter the following application properties and values:



Key	Value
<b>OUTPUT_STREAM</b>	<b>ExampleOutputStream</b>
<b>AWS_REGION</b>	<b>us-west-2</b>
<b>AggregationEnabled</b>	<b>false</b>

- Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
- For **CloudWatch logging**, select the **Enable** check box.
- 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

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

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

You can also check the Kinesis Data Streams console, in the data stream's **Enhanced fan-out** tab, for the name of your consumer (*my-flink-efo-consumer*).

## Clean Up AWS Resources

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

#### This topic contains the following sections:

- [Delete Your Kinesis Data Analytics Application \(p. 145\)](#)
- [Delete Your Kinesis Data Streams \(p. 146\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 146\)](#)
- [Delete Your IAM Resources \(p. 146\)](#)
- [Delete Your CloudWatch Resources \(p. 146\)](#)

### Delete Your Kinesis Data Analytics Application

- Open the Kinesis Data Analytics console at <https://console.aws.amazon.com/kinesisanalytics>.
- in the Kinesis Data Analytics panel, choose **MyApplication**.
- In the application's page, choose **Delete** and then confirm the deletion.

## Delete Your Kinesis Data Streams

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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 Kinesis Data Firehose

In this exercise, you create a Kinesis Data Analytics 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 \(DataStream API\) \(p. 75\)](#) exercise.

### This section contains the following steps:

- [Create Dependent Resources \(p. 147\)](#)
- [Write Sample Records to the Input Stream \(p. 147\)](#)
- [Download and Examine the Apache Flink Streaming Java Code \(p. 148\)](#)
- [Compile the Application Code \(p. 148\)](#)

- [Upload the Apache Flink Streaming Java Code \(p. 149\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 149\)](#)
- [Clean Up AWS Resources \(p. 155\)](#)

## Create Dependent Resources

Before you create a Kinesis Data Analytics for Apache Flink 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 User 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:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

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

def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
```

```
StreamName=stream_name,  
Data=json.dumps(data),  
PartitionKey="partitionkey")  
  
if __name__ == '__main__':  
    generate(STREAM_NAME, boto3.client('kinesis'))
```

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
```

2. Navigate to the `amazon-kinesis-data-analytics-java-examples/FirehoseSink` directory.

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 \(p. 75\)](#) in the [Getting Started \(DataStream API\) \(p. 75\)](#) 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 with previous Apache Flink versions” (p. 249).**
3. Compile the application with the following command:

```
mvn package -Dflink.version=1.13.2
```

**Note**

The provided source code relies on libraries from Java 11. If you are using a development environment,

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 \(p. 147\)](#) section.

### To upload the application code

1. Open the Amazon S3 console at <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 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 \(Console\) \(p. 149\)](#)
- [Create and Run the Application \(AWS CLI\) \(p. 152\)](#)

### 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 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 Apache Flink uses Apache Flink version 1.13.2.

- Leave the version pulldown as **Apache Flink version 1.13.2 (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 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
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.

```
{
  "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": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
      ]
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:*"
      ]
    }
  ]
}
```

```
    },
    {
      "Sid": "PutLogEvents",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
      ]
    },
    {
      "Sid": "ReadInputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream"
    },
    {
      "Sid": "WriteDeliveryStream",
      "Effect": "Allow",
      "Action": "firehose:*",
      "Resource": "arn:aws:firehose:us-west-2:012345678901:deliverystream/ExampleDeliveryStream"
    }
  ]
}
```

## 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

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

## 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.

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 [the section called "Update the Application Code" \(p. 155\)](#) 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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "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:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream"
    },
    {
      "Sid": "WriteDeliveryStream",
      "Effect": "Allow",
      "Action": "firehose:*",
      "Resource": "arn:aws:firehose:us-west-2:012345678901:deliverystream/ExampleDeliveryStream"
    }
  ]
}
```

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 Amazon 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 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" \(p. 152\)](#).

- 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 **KAReadSourceStreamWriteSinkStream** 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" \(p. 147\)](#) section (ka-

app-code-*<username>*.) Replace the sample account ID (*012345678901*) in the service execution role with your account ID.

```
{
  "ApplicationName": "test",
  "ApplicationDescription": "my java test app",
  "RuntimeEnvironment": "FLINK-1_13",
  "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"
    }
  }
}
```

2. Execute the [CreateApplication](#) action with the preceding request to create the application:

```
aws kinesisanalyticstv2 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`.

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

2. Execute the [StartApplication](#) action with the preceding request to start the application:

```
aws kinesisanalyticstv2 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`.

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

2. Execute the [StopApplication](#) action with the following request to stop the application:

```
aws kinesisanalyticstv2 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" \(p. 206\)](#).

## 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" \(p. 147\)](#) section.

```
{
  "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 \(p. 156\)](#)
- [Delete Your Kinesis Data Stream \(p. 156\)](#)

- [Delete Your Kinesis Data Firehose Delivery Stream \(p. 156\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 156\)](#)
- [Delete Your IAM Resources \(p. 156\)](#)
- [Delete Your CloudWatch Resources \(p. 157\)](#)

## 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

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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: 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 \(p. 157\)](#)
- [Setup \(p. 157\)](#)
- [Create Source Kinesis Stream \(p. 158\)](#)
- [Create and Update IAM Roles and Policies \(p. 158\)](#)
- [Update the Python Script \(p. 160\)](#)
- [Update the Java Application \(p. 161\)](#)
- [Build, Upload, and Run the Application \(p. 162\)](#)

## 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 \(DataStream API\) \(p. 75\)](#) 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/bPxrFcYEXAMPLEKEY

[ka-sink-stream-account-profile]
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/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.

### Note

When you use the console to create your application, the console creates a policy called `kinesis-analytics-service-<application name>-<application region>`, and a role called `kinesis-analytics-<application name>-<application region>`.

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.

```
{
  "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:::ka-app-code-username/aws-kinesis-analytics-java-
apps-1.0.jar"
        ]
    },
    {
        "Sid": "ListCloudwatchLogGroups",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeLogGroups"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:SINK012345678:log-group:*"
        ]
    },
    {
        "Sid": "ListCloudwatchLogStreams",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeLogStreams"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:SINK012345678:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:*"
        ]
    },
    {
        "Sid": "PutCloudwatchLogs",
        "Effect": "Allow",
        "Action": [
            "logs:PutLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:SINK012345678:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:kinesis-analytics-log-stream"
        ]
    }
]
```

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":
      "arn:aws:kinesis:us-west-2:SOURCE123456784:stream/
SourceAccountExampleInputStream"
  }
]
```

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.

```
{
  "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.

```
import json
import boto3
import random
import datetime
import os

os.environ['AWS_PROFILE'] = 'ka-source-stream-account-profile'
os.environ['AWS_DEFAULT_REGION'] = 'us-west-2'

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="SourceAccountExampleInputStream",
        Data=data,
        PartitionKey="partitionkey")
```

## 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.

```
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 org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;
import org.apache.flink.streaming.connectors.kinesis.config.AWSConfigConstants;

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");
    inputProperties.setProperty(AWSConfigConstants.AWS_ROLE_ARN, roleArn);
    inputProperties.setProperty(AWSConfigConstants.AWS_ROLE_SESSION_NAME,
roleSessionName);
    inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
    inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
"LATEST");

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

private static FlinkKinesisProducer<String> createSinkFromStaticConfig() {
    Properties outputProperties = new Properties();
    outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
    outputProperties.setProperty("AggregationEnabled", "false");

    FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
SimpleStringSchema(), outputProperties);
    sink.setDefaultStream(outputStreamName);
    sink.setDefaultPartition("0");
    return sink;
}

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

    DataStream<String> input = createSourceFromStaticConfig(env);

    input.addSink(createSinkFromStaticConfig());

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

## 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.

```
mvn package -Dflink.version=1.13.2
```

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.

```
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.

## Tutorial: Using a Custom Truststore with Amazon MSK

The following tutorial demonstrates how to securely connect (encryption in transit) to a Kafka Cluster that uses server certificates issued by a custom, private or even self-hosted Certificate Authority (CA).

For connecting any Kafka Client securely over TLS to a Kafka Cluster, the Kafka Client (like the example Flink Application) must trust the complete chain of trust presented by the Kafka Cluster's server certificates (from the Issuing CA up to the Root-Level CA). As an example for a custom Truststore, we will use an Amazon MSK cluster with Mutual TLS (MTLS) Authentication enabled. This implies that the MSK cluster nodes use server certificates that are issued by an AWS Certificate Manager Private Certificate Authority (ACM Private CA) that is private to your account and Region and therefore not trusted by the default Truststore of the Java Virtual Machine (JVM) executing the Flink Application.

### Note

- A **Keystore** is used to store private key and identity certificates an application should present to both server or client for verification.
- A **Truststore** is used to store certificates from Certified Authorities (CA) that verify the certificate presented by the server in an SSL connection.

You can also use the technique in this tutorial for interactions between a Kinesis Data Analytics application and other Apache Kafka sources, such as:

- A custom Apache Kafka cluster hosted in AWS ([Amazon EC2](#) or [Amazon EKS](#))
- A [Confluent Kafka](#) cluster hosted in AWS
- An on-premises Kafka cluster accessed through [AWS Direct Connect](#) or VPN

Your application will use a custom consumer (`CustomFlinkKafkaConsumer`) that overrides the `open` method to load the custom truststore. This makes the truststore available to the application after the application restarts or replaces threads.

The custom truststore is retrieved and stored using the following code, from the `CustomFlinkKafkaConsumer.java` file:

```
@Override
public void open(Configuration configuration) throws Exception {
    // write truststore to /tmp
    // NOTE: make sure that truststore is in JKS format for KDA/Flink. See README for
    details
    dropFile("/tmp");

    super.open(configuration);
}

private void dropFile(String destFolder) throws Exception
{
    InputStream input = null;
    OutputStream outputStream = null;

    try {
```

```
ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
input = classLoader.getResourceAsStream("kafka.client.truststore.jks");
byte[] buffer = new byte[input.available()];
input.read(buffer);

File destDir = new File(destFolder);
File targetFile = new File(destDir, "kafka.client.truststore.jks");
OutputStream outputStream = new FileOutputStream(targetFile);
outputStream.write(buffer);
outputStream.flush();
}
```

#### Note

Apache Flink requires the truststore to be in [JKS format](#).

#### Note

To set up the required prerequisites for this exercise, first complete the [Getting Started \(DataStream API\)](#) (p. 75) exercise.

#### This tutorial contains the following sections:

- [Create a VPC with an Amazon MSK Cluster](#) (p. 164)
- [Create a Custom Truststore and Apply It to Your Cluster](#) (p. 165)
- [Create the Application Code](#) (p. 165)
- [Upload the Apache Flink Streaming Java Code](#) (p. 165)
- [Create the Application](#) (p. 166)
- [Configure the Application](#) (p. 166)
- [Run the Application](#) (p. 167)
- [Test the Application](#) (p. 168)

## Create a 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, also do the following:

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

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

#### Note

If the `kafka-topics.sh` command returns a `ZooKeeperClientTimeoutException`, verify that the Kafka cluster's security group has an inbound rule to allow all traffic from the client instance's private IP address.

- 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):

```
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
{...
  "BootstrapBrokerStringTls": "b-2.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-1.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-3.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094"
}
```

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

## Create a Custom Truststore and Apply It to Your Cluster

In this section, you create a custom certificate authority (CA), use it to generate a custom truststore, and apply it to your MSK cluster.

To create and apply your custom truststore, follow the [Client Authentication](#) tutorial in the *Amazon Managed Streaming for Apache Kafka Developer Guide*.

## Create the Application Code

In this section, you 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:

```
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/CustomKeystore/KDAFlinkStreamingJob.java` and `CustomFlinkKafkaConsumer.java` files. You can examine the code to familiarize yourself with the structure of Kinesis Data Analytics for Apache Flink 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:

```
mvn package -Dflink.version=1.13.2
```

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

```
target/flink-app-1.0-SNAPSHOT.jar
```

### Note

The provided source code relies on libraries from Java 11. If you're using a development environment,

## 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 [Getting Started \(DataStream API\) \(p. 75\)](#) tutorial.

### Note

If you deleted the Amazon S3 bucket from the Getting Started tutorial, follow the [the section called "Upload the Apache Flink Streaming Java Code" \(p. 82\)](#) step again.

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 `KafkaGettingStartedJob-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 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 version 1.13.2**.
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 Apache Flink 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**.
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 **flink-app-1.0-SNAPSHOT.jar**.
3. Under **Access to application resources**, for **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.

### Note

When you specify application resources using the console (such as logs or a 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:

Key	Value
topic	AWSKafkaTutorialTopic
bootstrap.servers	<i>The bootstrap server list you saved previously</i>
security.protocol	SSL
ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
ssl.truststore.password	changeit

**Note**

The `ssl.truststore.password` for the default certificate is "changeit"—you don't need to change this value if you're using the default certificate.

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

Key	Value
topic	AWSKafkaTutorialTopicDestination
bootstrap.servers	<i>The bootstrap server list you saved previously</i>
security.protocol	SSL
ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
ssl.truststore.password	changeit
transaction.timeout.ms	1000

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 \(p. 21\)](#).

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

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

## 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 that 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 \(p. 312\)](#) section in the [Troubleshooting \(p. 309\)](#) topic.

## Example: Using Apache Beam

In this exercise, you create a Kinesis Data Analytics application that transforms data using [Apache Beam](#). Apache Beam is a programming model for processing streaming data. For information about using Apache Beam with Kinesis Data Analytics, see [Using Apache Beam \(p. 6\)](#).

### Note

To set up required prerequisites for this exercise, first complete the [Getting Started \(DataStream API\) \(p. 75\)](#) exercise.

**This topic contains the following sections:**

- [Create Dependent Resources \(p. 168\)](#)
- [Write Sample Records to the Input Stream \(p. 169\)](#)
- [Download and Examine the Application Code \(p. 169\)](#)
- [Compile the Application Code \(p. 170\)](#)
- [Upload the Apache Flink Streaming Java Code \(p. 170\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 171\)](#)
- [Clean Up AWS Resources \(p. 173\)](#)
- [Next Steps \(p. 174\)](#)

## Create Dependent Resources

Before you create a Kinesis Data Analytics 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 streams `ExampleInputStream` and `ExampleOutputStream`.



- [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service User 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 random strings to the stream for the application to process.

### Note

This section requires the [AWS SDK for Python \(Boto\)](#).

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

```
import json
import boto3
import random

kinesis = boto3.client('kinesis')

while True:
    data = random.choice(['ping', 'telnet', 'ftp', 'tracert', 'netstat'])
    print(data)
    kinesis.put_record(
        StreamName="ExampleInputStream",
        Data=data,
        PartitionKey="partitionkey")
```

2. Run the `ping.py` script:

```
$ python ping.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
```

3. Navigate to the `amazon-kinesis-data-analytics-java-examples/Beam` directory.

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

- The application uses the Apache Beam [ParDo](#) to process incoming records by invoking a custom transform function called `PingPongFn`.

The code to invoke the `PingPongFn` function is as follows:

```
.apply("Pong transform",
    ParDo.of(new PingPongFn()))
```

- Kinesis Data Analytics applications that use Apache Beam require the following components. If you don't include these components and versions in your `pom.xml`, your application loads the incorrect versions from the environment dependencies, and since the versions do not match, your application crashes at runtime.

```
<jackson.version>2.10.2</jackson.version>
...
<dependency>
  <groupId>com.fasterxml.jackson.module</groupId>
  <artifactId>jackson-module-jaxb-annotations</artifactId>
  <version>2.10.2</version>
</dependency>
```

- The `PingPongFn` transform function passes the input data into the output stream, unless the input data is **ping**, in which case it emits the string **pong** to the output stream.

The code of the transform function is as follows:

```
private static class PingPongFn extends DoFn<KinesisRecord, byte[]> {
    private static final Logger LOG = LoggerFactory.getLogger(PingPongFn.class);

    @ProcessElement
    public void processElement(ProcessContext c) {
        String content = new String(c.element().getDataAsBytes(),
            StandardCharsets.UTF_8);
        if (content.trim().equalsIgnoreCase("ping")) {
            LOG.info("Ponged!");
            c.output("pong\n".getBytes(StandardCharsets.UTF_8));
        } else {
            LOG.info("No action for: " + content);
            c.output(c.element().getDataAsBytes());
        }
    }
}
```

## 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 \(p. 75\)](#) in the [Getting Started \(DataStream API\) \(p. 75\)](#) tutorial.
2. Compile the application with the following command:

```
mvn package -Dflink.version=1.13.2 -Dflink.version.minor=1.8
```

### Note

The provided source code relies on libraries from Java 11. If you are using a development environment,

Compiling the application creates the application JAR file (`target/basic-beam-app-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 \(p. 168\)](#) 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 `basic-beam-app-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 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 uses Apache Flink version 1.13.2.
- Leave the version pulldown as **Apache Flink version 1.13.2 (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 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (`012345678901`) with your account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
```

```

        "s3:GetObject",
        "logs:DescribeLogGroups",
        "s3:GetObjectVersion"
    ],
    "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*",
        "arn:aws:s3:::ka-app-code-<username>/basic-beam-app-1.0.jar"
    ]
},
{
    "Sid": "DescribeLogStreams",
    "Effect": "Allow",
    "Action": "logs:DescribeLogStreams",
    "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
},
{
    "Sid": "PutLogEvents",
    "Effect": "Allow",
    "Action": "logs:PutLogEvents",
    "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
},
{
    "Sid": "ListCloudwatchLogGroups",
    "Effect": "Allow",
    "Action": [
        "logs:DescribeLogGroups"
    ],
    "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
    ]
},
{
    "Sid": "ReadInputStream",
    "Effect": "Allow",
    "Action": "kinesis:*",
    "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
},
{
    "Sid": "WriteOutputStream",
    "Effect": "Allow",
    "Action": "kinesis:*",
    "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
}
]
}

```

## 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 **basic-beam-app-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 **BeamApplicationProperties**.
5. Enter the following application properties and values:

Key	Value
<code>InputStreamName</code>	<code>ExampleInputStream</code>
<code>OutputStreamName</code>	<code>ExampleOutputStream</code>
<code>AwsRegion</code>	<code>us-west-2</code>

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 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

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

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 \(p. 173\)](#)
- [Delete Your Kinesis Data Streams \(p. 173\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 174\)](#)
- [Delete Your IAM Resources \(p. 174\)](#)
- [Delete Your CloudWatch Resources \(p. 174\)](#)

## 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. In the application's page, choose **Delete** and then confirm the deletion.

## Delete Your Kinesis Data Streams

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.

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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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.

## Next Steps

Now that you've created and run a basic Kinesis Data Analytics application that transforms data using Apache Beam, see the following application for an example of a more advanced Kinesis Data Analytics solution.

- **Beam on Kinesis Data Analytics Streaming Workshop:** In this workshop, we explore an end to end example that combines batch and streaming aspects in one uniform Apache Beam pipeline.

# Python Examples

The following examples demonstrate how to create applications using Python with the Apache Flink Table API.

### Topics

- [Example: Creating a Tumbling Window in Python \(p. 175\)](#)
- [Example: Creating a Sliding Window in Python \(p. 181\)](#)

- [Example: Send Streaming Data to Amazon S3 in Python](#) (p. 188)

## Example: Creating a Tumbling Window in Python

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

### Note

To set up required prerequisites for this exercise, first complete the [Getting Started \(Python\)](#) (p. 103) exercise.

**This topic contains the following sections:**

- [Create Dependent Resources](#) (p. 175)
- [Write Sample Records to the Input Stream](#) (p. 175)
- [Download and Examine the Application Code](#) (p. 176)
- [Compress and Upload the Apache Flink Streaming Python Code](#) (p. 177)
- [Create and Run the Kinesis Data Analytics Application](#) (p. 177)
- [Clean Up AWS Resources](#) (p. 180)

## Create Dependent Resources

Before you create a Kinesis Data Analytics 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 streams `ExampleInputStream` and `ExampleOutputStream`.
- [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service User 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 \(Boto\)](#).

### Note

The Python script in this section uses the AWS CLI. You must configure your AWS CLI to use your account credentials and default region. To configure your AWS CLI, enter the following:

```
aws configure
```

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

```
import datetime
```

```
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

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

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

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

2. Run the stock.py script:

```
$ python stock.py
```

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

## Download and Examine the Application Code

The Python 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
```

3. Navigate to the amazon-kinesis-data-analytics-java-examples/python/TumblingWindow directory.

The application code is located in the tumbling-windows.py file. Note the following about the application code:

- The application uses a Kinesis table source to read from the source stream. The following snippet calls the create\_table function to create the Kinesis table source:

```
table_env.execute_sql(
    create_table(input_table_name, input_stream, input_region, stream_initpos)
)
```

The create\_table function uses a SQL command to create a table that is backed by the streaming source:



```
def create_table(table_name, stream_name, region, stream_initpos):
    return """ CREATE TABLE {0} (
        ticker VARCHAR(6),
        price DOUBLE,
        event_time TIMESTAMP(3),
        WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND

    )
    PARTITIONED BY (ticker)
    WITH (
        'connector' = 'kinesis',
        'stream' = '{1}',
        'aws.region' = '{2}',
        'scan.stream.initpos' = '{3}',
        'sink.partitioner-field-delimiter' = ';',
        'sink.producer.collection-max-count' = '100',
        'format' = 'json',
        'json.timestamp-format.standard' = 'ISO-8601'
    ) """ .format(
        table_name, stream_name, region, stream_initpos
    )
```

- The application uses the Tumble operator to aggregate records within a specified tumbling window, and return the aggregated records as a table object:

```
tumbling_window_table = (
    input_table.window(
        Tumble.over("10.seconds").on("event_time").alias("ten_second_window")
    )
    .group_by("ticker, ten_second_window")
    .select("ticker, price.sum as price, ten_second_window.end as event_time")
)
```

- The application uses the Kinesis Flink connector, from the [amazon-kinesis-sql-connector-flink-2.12/1.13.2.jar](#).

## Compress and Upload the Apache Flink Streaming Python Code

In this section, you upload your application code to the Amazon S3 bucket you created in the [Create Dependent Resources \(p. 175\)](#) section.

1. Use your preferred compression application to compress the `streaming-file-sink.py` and `flink-sql-connector-kinesis_2.12-1.13.2.jar` files. Name the archive `myapp.zip`.
2. In the Amazon S3 console, choose the **ka-app-code-*<username>*** bucket, and choose **Upload**.
3. In the **Select files** step, choose **Add files**. Navigate to the `myapp.zip` 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

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 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 uses Apache Flink version 1.13.2.

- Leave the version pulldown as **Apache Flink version 1.13.2 (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 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**.
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 `myapp.zip`.
3. Under **Access to application resources**, for **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
4. Under **Properties**, choose **Add group**. For **Group ID**, enter `consumer.config.0`.
5. Enter the following application properties and values:

Key	Value
<code>input.stream.name</code>	<code>ExampleInputStream</code>
<code>aws.region</code>	<code>us-west-2</code>
<code>flink.stream.initpos</code>	<code>LATEST</code>

Choose **Save**.

6. Under **Properties**, choose **Add group** again. For **Group ID**, enter `producer.config.0`.
7. Enter the following application properties and values:

Key	Value
<code>output.stream.name</code>	<code>ExampleOutputStream</code>
<code>aws.region</code>	<code>us-west-2</code>

Key	Value
<b>shard.count</b>	<b>1</b>

8. Under **Properties**, choose **Add group** again. For **Group ID**, enter **kinesis.analytics.flink.run.options**. This special property group tells your application where to find its code resources. For more information, see [Specifying your Code Files \(p. 19\)](#).
9. Enter the following application properties and values:

Key	Value
<b>python</b>	<b>tumbling-windows.py</b>
<b>jarfile</b>	<b>flink-sql-connector-kinesis_2.12-1.13.2.jar</b>

10. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
11. For **CloudWatch logging**, select the **Enable** check box.
12. 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.

## 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (**012345678901**) with your account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "logs:DescribeLogGroups",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*",
        "arn:aws:s3:::ka-app-code-<username>/myapp.zip"
      ]
    }
  ]
}
```

```
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": "logs:DescribeLogStreams",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
    },
    {
      "Sid": "PutLogEvents",
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
    },
    {
      "Sid": "ListCloudwatchLogGroups",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
      ]
    },
    {
      "Sid": "ReadInputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    },
    {
      "Sid": "WriteOutputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
    }
  ]
}
```

## Run the Application

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

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 \(p. 181\)](#)
- [Delete Your Kinesis Data Streams \(p. 181\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 181\)](#)
- [Delete Your IAM Resources \(p. 181\)](#)
- [Delete Your CloudWatch Resources \(p. 181\)](#)

## 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. In the application's page, choose **Delete** and then confirm the deletion.

## Delete Your Kinesis Data Streams

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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: Creating a Sliding Window in Python

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

### Note

To set up required prerequisites for this exercise, first complete the [Getting Started \(Python\) \(p. 103\)](#) exercise.

**This topic contains the following sections:**

- [Create Dependent Resources \(p. 182\)](#)
- [Write Sample Records to the Input Stream \(p. 182\)](#)
- [Download and Examine the Application Code \(p. 183\)](#)
- [Compress and Upload the Apache Flink Streaming Python Code \(p. 184\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 184\)](#)
- [Clean Up AWS Resources \(p. 187\)](#)

## Create Dependent Resources

Before you create a Kinesis Data Analytics 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 streams `ExampleInputStream` and `ExampleOutputStream`.
- [How Do I Create an S3 Bucket?](#) in the *Amazon Simple Storage Service User 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 \(Boto\)](#).

### Note

The Python script in this section uses the AWS CLI. You must configure your AWS CLI to use your account credentials and default region. To configure your AWS CLI, enter the following:

```
aws configure
```

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

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

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

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

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

2. Run the `stock.py` script:

```
$ python stock.py
```

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

## Download and Examine the Application Code

The Python 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
```

3. Navigate to the `amazon-kinesis-data-analytics-java-examples/python/SlidingWindow` directory.

The application code is located in the `sliding-windows.py` file. Note the following about the application code:

- The application uses a Kinesis table source to read from the source stream. The following snippet calls the `create_table` function to create the Kinesis table source:

```
table_env.execute_sql(
    create_table(input_table_name, input_stream, input_region, stream_initpos)
)
```

The `create_table` function uses a SQL command to create a table that is backed by the streaming source:

```
def create_table(table_name, stream_name, region, stream_initpos):
    return """ CREATE TABLE {0} (
        ticker VARCHAR(6),
        price DOUBLE,
        event_time TIMESTAMP(3),
        WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
    )
    PARTITIONED BY (ticker)
    WITH (
        'connector' = 'kinesis',
        'stream' = '{1}',
```

```
'aws.region' = '{2}',
'scan.stream.initpos' = '{3}',
'sink.partition-field-delimiter' = ';',
'sink.producer.collection-max-count' = '100',
'format' = 'json',
'json.timestamp-format.standard' = 'ISO-8601'
) """.format(
    table_name, stream_name, region, stream_initpos
)
```

- The application uses the `Slide` operator to aggregate records within a specified sliding window, and return the aggregated records as a table object:

```
sliding_window_table = (
    input_table.window(
        Slide.over("10.seconds")
        .every("5.seconds")
        .on("event_time")
        .alias("ten_second_window")
    )
    .group_by("ticker, ten_second_window")
    .select("ticker, price.min as price, ten_second_window.end as event_time")
)
```

- The application uses the Kinesis Flink connector, from the `amazon-kinesis-connector-flink-2.0.0.jar` file.

## Compress and Upload the Apache Flink Streaming Python Code

In this section, you upload your application code to the Amazon S3 bucket you created in the [Create Dependent Resources \(p. 182\)](#) section.

1. Use your preferred compression application to compress the `streaming-file-sink.py` and `flink-sql-connector-kinesis_2.12-1.13.2.jar` files. Name the archive `myapp.zip`.
2. In the Amazon S3 console, choose the **ka-app-code-*<username>*** bucket, and choose **Upload**.
3. In the **Select files** step, choose **Add files**. Navigate to the `myapp.zip` 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

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 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 uses Apache Flink version 1.13.2.



- Leave the version pulldown as **Apache Flink version 1.13.2 (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 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**.
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 **myapp.zip**.
3. Under **Access to application resources**, for **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
4. Under **Properties**, choose **Add group**. For **Group ID**, enter **consumer.config.0**.
5. Enter the following application properties and values:

Key	Value
<b>input.stream.name</b>	<b>ExampleInputStream</b>
<b>aws.region</b>	<b>us-west-2</b>
<b>flink.stream.initpos</b>	<b>LATEST</b>

Choose **Save**.

6. Under **Properties**, choose **Add group** again. For **Group ID**, enter **producer.config.0**.
7. Enter the following application properties and values:

Key	Value
<b>output.stream.name</b>	<b>ExampleOutputStream</b>
<b>aws.region</b>	<b>us-west-2</b>
<b>shard.count</b>	<b>1</b>

8. Under **Properties**, choose **Add group** again. For **Group ID**, enter **kinesis.analytics.flink.run.options**. This special property group tells your application where to find its code resources. For more information, see [Specifying your Code Files \(p. 19\)](#).
9. Enter the following application properties and values:

Key	Value
python	sliding-windows.py
jarfile	amazon-kinesis-connector-flink-2.0.0.jar

10. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
11. For **CloudWatch logging**, select the **Enable** check box.
12. 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.

## 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (**012345678901**) with your account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "logs:DescribeLogGroups",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*",
        "arn:aws:s3:::ka-app-code-<username>/myapp.zip"
      ]
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": "logs:DescribeLogStreams",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
    }
  ]
}
```

```
        "Sid": "PutLogEvents",
        "Effect": "Allow",
        "Action": "logs:PutLogEvents",
        "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
    },
    {
        "Sid": "ListCloudwatchLogGroups",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeLogGroups"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:*"
        ]
    },
    {
        "Sid": "ReadInputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    },
    {
        "Sid": "WriteOutputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
    }
]
```

## Run the Application

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

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 \(p. 187\)](#)
- [Delete Your Kinesis Data Streams \(p. 188\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 188\)](#)
- [Delete Your IAM Resources \(p. 188\)](#)
- [Delete Your CloudWatch Resources \(p. 188\)](#)

## 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. In the application's page, choose **Delete** and then confirm the deletion.

## Delete Your Kinesis Data Streams

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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: Send Streaming Data to Amazon S3 in Python

In this exercise, you create a Python Kinesis Data Analytics application that streams data to an Amazon Simple Storage Service sink.

### Note

To set up required prerequisites for this exercise, first complete the [Getting Started \(Python\) \(p. 103\)](#) exercise.

### This topic contains the following sections:

- [Create Dependent Resources \(p. 189\)](#)
- [Write Sample Records to the Input Stream \(p. 189\)](#)
- [Download and Examine the Application Code \(p. 190\)](#)
- [Compress and Upload the Apache Flink Streaming Python Code \(p. 191\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 191\)](#)

- [Clean Up AWS Resources \(p. 194\)](#)

## Create Dependent Resources

Before you create a Kinesis Data Analytics 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-code-<username>`)

### Note

Kinesis Data Analytics for Apache Flink cannot write data to Amazon S3 with server-side encryption enabled on Kinesis Data Analytics.

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 User 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 \(Boto\)](#).

### Note

The Python script in this section uses the AWS CLI. You must configure your AWS CLI to use your account credentials and default region. To configure your AWS CLI, enter the following:

```
aws configure
```

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

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

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

def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
```

```
print(data)
kinesis_client.put_record(
    StreamName=stream_name,
    Data=json.dumps(data),
    PartitionKey="partitionkey")

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

2. Run the `stock.py` script:

```
$ python stock.py
```

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

## Download and Examine the Application Code

The Python 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
```

3. Navigate to the `amazon-kinesis-data-analytics-java-examples/python/S3Sink` directory.

The application code is located in the `streaming-file-sink.py` file. Note the following about the application code:

- The application uses a Kinesis table source to read from the source stream. The following snippet calls the `create_table` function to create the Kinesis table source:

```
table_env.execute_sql(
    create_table(input_table_name, input_stream, input_region, stream_initpos)
)
```

The `create_table` function uses a SQL command to create a table that is backed by the streaming source:

```
def create_table(table_name, stream_name, region, stream_initpos):
    return """ CREATE TABLE {0} (
        ticker VARCHAR(6),
        price DOUBLE,
        event_time TIMESTAMP(3),
        WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
    )
    PARTITIONED BY (ticker)
    WITH (
        'connector' = 'kinesis',
        'stream' = '{1}',
        'aws.region' = '{2}',
        'scan.stream.initpos' = '{3}',
        'sink.partitionner-field-delimiter' = ';'
    )
```

```
        'sink.producer.collection-max-count' = '100',  
        'format' = 'json',  
        'json.timestamp-format.standard' = 'ISO-8601'  
    ) ""$.format(  
        table_name, stream_name, region, stream_initpos  
    )  
}
```

- The application uses the filesystem connector to send records to an Amazon S3 bucket:

```
def create_sink_table(table_name, bucket_name):  
    return """ CREATE TABLE {0} (  
        ticker VARCHAR(6),  
        price DOUBLE,  
        event_time TIMESTAMP(3),  
        WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND  
    )  
    PARTITIONED BY (ticker)  
    WITH (  
        'connector'='filesystem',  
        'path'='s3a://{1}/',  
        'format'='csv',  
        'sink.partition-commit.policy.kind'='success-file',  
        'sink.partition-commit.delay' = '1 min'  
    ) ""$.format(  
        table_name, bucket_name)
```

- The application uses the Kinesis Flink connector, from the amazon-kinesis-connector-flink-2.0.0.jar file.

## Compress and Upload the Apache Flink Streaming Python Code

In this section, you upload your application code to the Amazon S3 bucket you created in the [Create Dependent Resources \(p. 189\)](#) section.

1. Use your preferred compression application to compress the streaming-file-sink.py and flink-sql-connector-kinesis\_2.12-1.13.2.jar files. Name the archive myapp.zip.
2. In the Amazon S3 console, choose the **ka-app-code-*<username>*** bucket, and choose **Upload**.
3. In the **Select files** step, choose **Add files**. Navigate to the myapp.zip 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

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 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 uses Apache Flink version 1.13.2.

- Leave the version pulldown as **Apache Flink version 1.13.2 (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 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**.
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 **myapp.zip**.
3. Under **Access to application resources**, for **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
4. Under **Properties**, choose **Add group**. For **Group ID**, enter **consumer.config.0**.
5. Enter the following application properties and values:

Key	Value
<b>input.stream.name</b>	<b>ExampleInputStream</b>
<b>aws.region</b>	<b>us-west-2</b>
<b>flink.stream.initpos</b>	<b>LATEST</b>

Choose **Save**.

6. Under **Properties**, choose **Add group** again. For **Group ID**, enter **kinesis.analytics.flink.run.options**. This special property group tells your application where to find its code resources. For more information, see [Specifying your Code Files \(p. 19\)](#).
7. Enter the following application properties and values:

Key	Value
<b>python</b>	<b>streaming-file-sink.py</b>
<b>jarfile</b>	<b>S3Sink/lib/flink-sql-connector-kinesis_2.12-1.13.2.jar</b>



- Under **Properties**, choose **Add group** again. For **Group ID**, enter **sink.config.0**. This special property group tells your application where to find its code resources. For more information, see [Specifying your Code Files \(p. 19\)](#).
- Enter the following application properties and values: (replace *bucket-name* with the actual name of your Amazon S3 bucket.)

Key	Value
<code>output.bucket.name</code>	<i>bucket-name</i>

- Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
- For **CloudWatch logging**, select the **Enable** check box.
- 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.

## Edit the IAM Policy

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

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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "logs:DescribeLogGroups",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*",
        "arn:aws:s3:::ka-app-code-<username>/myapp.zip"
      ]
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": "logs:DescribeLogStreams",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
```

```
    },
    {
      "Sid": "PutLogEvents",
      "Effect": "Allow",
      "Action": "logs:PutLogEvents",
      "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
    },
    {
      "Sid": "ListCloudwatchLogGroups",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
      ]
    },
    {
      "Sid": "ReadInputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    },
    {
      "Sid": "WriteObjects",
      "Effect": "Allow",
      "Action": [
        "s3:Abort*",
        "s3:DeleteObject*",
        "s3:GetObject*",
        "s3:GetBucket*",
        "s3:List*",
        "s3:ListBucket",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::ka-app-<username>",
        "arn:aws:s3:::ka-app-<username>/*"
      ]
    }
  ]
}
```

## Run the Application

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

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 \(p. 195\)](#)
- [Delete Your Kinesis Data Stream \(p. 195\)](#)
- [Delete Your Amazon S3 Objects and Bucket \(p. 195\)](#)

- [Delete Your IAM Resources \(p. 195\)](#)
- [Delete Your CloudWatch Resources \(p. 195\)](#)

## 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. In the application's page, choose **Delete** and then confirm the deletion.

## Delete Your Kinesis Data Stream

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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-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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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 Solutions

The following end-to-end examples demonstrate advanced Kinesis Data Analytics solutions.

### Topics

- [AWS Streaming Data Solution for Amazon Kinesis \(p. 196\)](#)
- [Clickstream Lab with Apache Flink and Apache Kafka \(p. 196\)](#)

- [Custom Scaling using Application Auto Scaling \(p. 196\)](#)
- [Amazon CloudWatch Dashboard \(p. 196\)](#)
- [AWS Streaming Data Solution for Amazon MSK \(p. 196\)](#)
- [More Kinesis Data Analytics Solutions on GitHub \(p. 197\)](#)

## AWS Streaming Data Solution for Amazon Kinesis

The AWS Streaming Data Solution for Amazon Kinesis automatically configures the AWS services necessary to easily capture, store, process, and deliver streaming data. The solution provides multiple options for solving streaming data use cases. The Kinesis Data Analytics option provides an end-to-end streaming ETL example demonstrating a real-world application that runs analytical operations on simulated New York taxi data.

Each solution includes the following components:

- A AWS CloudFormation package to deploy the complete example.
- A CloudWatch dashboard for displaying application metrics.
- CloudWatch alarms on the most relevant application metrics.
- All necessary IAM roles and policies.

The solution can be found here: [Streaming Data Solution for Amazon Kinesis](#)

## Clickstream Lab with Apache Flink and Apache Kafka

An end to end lab for clickstream use cases using Amazon Managed Streaming for Apache Kafka for streaming storage and Amazon Kinesis Data Analytics for Apache Flink applications for stream processing.

The solution can be found here: [Clickstream Lab](#)

## Custom Scaling using Application Auto Scaling

A sample that helps users automatically scale their Kinesis Data Analytics for Apache Flink applications using Application Auto Scaling. This enables users to set up custom scaling policies and custom scaling attributes.

The solution can be found here: [Kinesis Data Analytics Flink App Autoscaling](#)

## Amazon CloudWatch Dashboard

A sample CloudWatch dashboard for monitoring Amazon Kinesis Data Analytics applications. The sample dashboard also includes a [demo application](#) to help with demonstrating the functionality of the dashboard.

The solution can be found here: [Kinesis Data Analytics Metrics Dashboard](#)

## AWS Streaming Data Solution for Amazon MSK

The AWS Streaming Data Solution for Amazon MSK provides AWS CloudFormation templates where data flows through producers, streaming storage, consumers, and destinations.

The solution can be found here: [AWS Streaming Data Solution for Amazon MSK](#)

## More Kinesis Data Analytics Solutions on GitHub

The following end-to-end examples demonstrate advanced Kinesis Data Analytics solutions and are available on GitHub:

- [Amazon Kinesis Data Analytics Flink – Benchmarking Utility](#)
- [Snapshot Manager – Amazon Kinesis Data Analytics for Apache Flink](#)
- [Streaming ETL with Apache Flink and Amazon Kinesis Data Analytics](#)
- [Real-time sentiment analysis on customer feedback](#)

# Security in Amazon Kinesis Data Analytics

Cloud security at AWS is the highest priority. As an AWS customer, you will benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. The effectiveness of our security is regularly tested and verified by third-party auditors as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Kinesis Data Analytics, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Kinesis Data Analytics. The following topics show you how to configure Kinesis Data Analytics to meet your security and compliance objectives. You'll also learn how to use other Amazon services that can help you to monitor and secure your Kinesis Data Analytics resources.

## Topics

- [Data Protection in Amazon Kinesis Data Analytics for Apache Flink \(p. 198\)](#)
- [Identity and Access Management in Amazon Kinesis Data Analytics for Apache Flink \(p. 199\)](#)
- [Monitoring Amazon Kinesis Data Analytics \(p. 202\)](#)
- [Compliance Validation for Amazon Kinesis Data Analytics for Apache Flink \(p. 202\)](#)
- [Resilience in Amazon Kinesis Data Analytics for Apache Flink \(p. 202\)](#)
- [Infrastructure Security in Kinesis Data Analytics for Apache Flink \(p. 203\)](#)
- [Security Best Practices for Kinesis Data Analytics for Apache Flink \(p. 204\)](#)

## Data Protection in Amazon Kinesis Data Analytics for Apache Flink

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 Apache Flink

#### Encryption at Rest

Note the following about encrypting data at rest with Kinesis Data Analytics for Apache Flink:

- 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 KMS-Managed Keys \(SSE-KMS\)](#).
- A Kinesis Data Analytics for Apache Flink 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.
- Durable application storage is encrypted at rest.
- Running application storage 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 Apache Flink

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 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.

```
kinesis-analytics-ApplicationName
```

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 \(p. 200\)](#)
- [Permissions Policy \(p. 200\)](#)

## 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.

```
{
  "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.

```
{
  "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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "WriteOutputKinesis",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:aws-region:aws-account-id:stream/output-stream-name"
      ]
    }
  ]
}
```

## Permissions Policy for Writing to a Kinesis Data Firehose Delivery Stream

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

```
{
  "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

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

```
    ],  
    "Resource": "*"    
  }  
]  
}
```

## Permissions Policy for Writing to a CloudWatch Log Stream

For information about adding permissions to write to a CloudWatch log stream, see [Adding Permissions to Write to the CloudWatch Log Stream](#) (p. 211).

# Monitoring Amazon Kinesis Data Analytics

Kinesis Data Analytics provides monitoring functionality for your applications. For more information, see [Logging and Monitoring](#) (p. 206).

## Compliance Validation for Amazon Kinesis Data Analytics for Apache Flink

Third-party auditors assess the security and compliance of Amazon Kinesis Data Analytics for Apache Flink as part of multiple AWS compliance programs. These include SOC, PCI, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see [Amazon Web 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 Apache Flink 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 Apache Flink 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 Apache Flink

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 Apache Flink 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 Apache Flink 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 \(p. 24\)](#).

## 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 Apache Flink

As a managed service, Amazon Kinesis Data Analytics is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access Kinesis Data Analytics through the network. 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 Apache Flink

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 Amazon 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. 198).

### Use CloudTrail to Monitor API Calls

Kinesis Data Analytics is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an Amazon service in Kinesis Data Analytics.

Using the information collected by CloudTrail, you can determine the request that was made to Kinesis Data Analytics, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information, see [the section called "Using AWS CloudTrail" \(p. 235\)](#).

# Logging and Monitoring in Amazon Kinesis Data Analytics for Apache Flink

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Kinesis Data Analytics and your Kinesis Data Analytics 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 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 \(p. 206\)](#)
- [Analyzing Logs with CloudWatch Logs Insights \(p. 213\)](#)
- [Viewing Kinesis Data Analytics Metrics and Dimensions \(p. 215\)](#)
- [Writing Custom Messages to CloudWatch Logs \(p. 233\)](#)
- [Logging Kinesis Data Analytics API Calls with AWS CloudTrail \(p. 235\)](#)

## Setting Up Application Logging

By adding an Amazon CloudWatch logging option to your Kinesis Data Analytics 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.

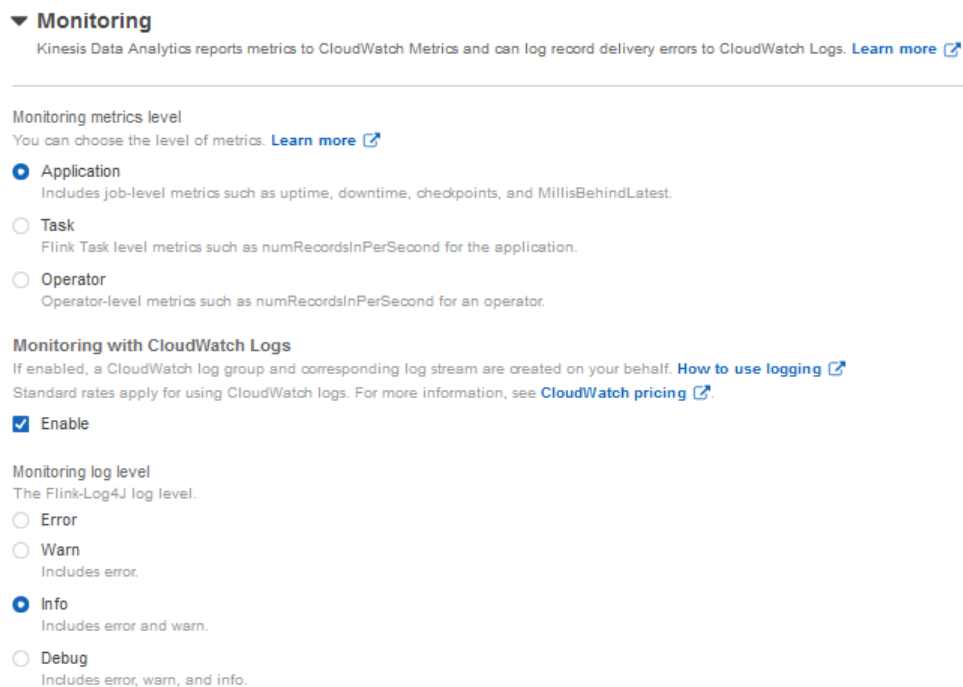
**This topic contains the following sections:**

- [Setting Up CloudWatch Logging Using the Console \(p. 207\)](#)
- [Setting Up CloudWatch Logging Using the CLI \(p. 208\)](#)
- [Application Monitoring Levels \(p. 211\)](#)
- [Logging Best Practices \(p. 212\)](#)
- [Logging Troubleshooting \(p. 212\)](#)
- [Next Step \(p. 213\)](#)

## 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.



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" \(p. 211\)](#).

## 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 \(p. 208\)](#)
- [Working with Application CloudWatch Logging Options \(p. 208\)](#)
- [Adding Permissions to Write to the CloudWatch Log Stream \(p. 211\)](#)

## 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 \(p. 321\)](#).

### 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. For more information about the action, see [CreateApplication](#).

```
{
  "ApplicationName": "test",
  "ApplicationDescription": "test-application-description",
  "RuntimeEnvironment": "FLINK-1_13",
  "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
  "ApplicationConfiguration": {
```



```
"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](#).

```
{
  "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](#).

```
{
  "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](#).

```
{
```

```
"ApplicationName": "<Name of application to delete log option from>",
"CloudWatchLoggingOptionId": "<ID of the application log option to delete>",
"CurrentApplicationVersionId": <Version of the application to delete the log option
from>
}
```

## 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" \(p. 211\)](#).

### Set the Application Logging Level when Creating an Application

The following example request for the [CreateApplication](#) action sets the application log level to INFO.

```
{
  "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_13",
    "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.

```
{
  "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 Apache Flink \(p. 199\)](#).

### 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.

```
{
  "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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt0123456789000",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:123456789012:log-group:my-log-group:log-stream:my-log-stream*",
        "arn:aws:logs:us-east-1:123456789012:log-group:my-log-group:*",
        "arn:aws:logs:us-east-1:123456789012:log-group:*"
      ]
    }
  ]
}
```

## 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” \(p. 30\)](#).
- **Operator:** Metrics are scoped to each operator. For information about operators, see [the section called “DataStream API Operators” \(p. 14\)](#).
- **Parallelism:** Metrics are scoped to application parallelism. You can only set this metrics level using the `MonitoringConfigurationUpdate` parameter of the `UpdateApplication` API. You cannot set this metrics level using the console. For information about parallelism, see [the section called “Scaling” \(p. 30\)](#).

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” \(p. 211\)](#).

- 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 `downtime` to diagnose other application issues. For information about reading CloudWatch metrics, see [Metrics and Dimensions \(p. 215\)](#).

## 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" \(p. 213\)](#).

# 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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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 \(p. 214\)](#)
- [Analyze Operations: Change in Parallelism \(p. 214\)](#)
- [Analyze Errors: Access Denied \(p. 215\)](#)
- [Analyze Errors: Source or Sink Not Found \(p. 215\)](#)
- [Analyze Errors: Application Task-Related Failures \(p. 215\)](#)

## 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 \(p. 30\)](#).

```
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](#).

```
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" \(p. 32\)](#).

```
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 ResourceNotFoundException logs. ResourceNotFoundException 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 status switches from RUNNING to RESTARTING.

```
fields @timestamp, @message
| filter applicationARN like /arn:aws:kinesisanalytics:us-west-2:012345678901:application
  \YourApplication/
| filter @message like /switched from RUNNING to RESTARTING/
| sort @timestamp desc
```

For applications using Apache Flink version 1.8.2 and prior, task-related failures will result in the application status switching from RUNNING to FAILED instead. When using Apache Flink 1.8.2 and prior, use the following query to search for application task-related failures:

```
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 Kinesis Data Analytics Metrics and Dimensions

This topic contains the following sections:

- [Application Metrics \(p. 216\)](#)
- [Kinesis Data Streams Connector Metrics \(p. 222\)](#)

- [Amazon MSK Connector Metrics \(p. 223\)](#)
- [Apache Zeppelin Metrics \(p. 224\)](#)
- [Viewing CloudWatch Metrics \(p. 224\)](#)
- [Setting CloudWatch Metrics Reporting Levels \(p. 225\)](#)
- [Using Custom Metrics with Amazon Kinesis Data Analytics for Apache Flink \(p. 226\)](#)
- [Using CloudWatch Alarms with Amazon Kinesis Data Analytics for Apache Flink \(p. 229\)](#)

When your Kinesis Data Analytics for Apache Flink application processes a data source, Kinesis Data Analytics reports the following metrics and dimensions to Amazon CloudWatch.

## Application Metrics

Metric	Unit	Description	Level	Usage Notes	
downtime	Milliseconds	For jobs currently in a failing/ recovering situation, the time elapsed during this outage.	Application	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.	
uptime	Milliseconds	The time that the job has been running without interruption.	Application	You can use this metric to determine if a job is running successfully. This metric returns -1 for completed jobs.	
fullRestarts	Count	The total number of times this job has fully restarted since it was submitted. This metric does not measure fine-grained restarts.	Application	You can use this metric to evaluate general application health. Restarts can occur during internal maintenance by Kinesis Data Analytics. Restarts higher than normal can indicate a	



Metric	Unit	Description	Level	Usage Notes	
				problem with the application.	
numberOfFailedCheckpoints	Count	The number of times checkpointing has failed.	Application	You can use this metric to monitor application health and progress. Checkpoints may fail due to application problems, such as throughput or permissions issues.	
lastCheckpointDuration	Milliseconds	The time it took to complete the last checkpoint	Application	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.	

Metric	Unit	Description	Level	Usage Notes	
lastCheckpointSize	Bytes	The total size of the last checkpoint	Application	<p>You can use this metric to determine running application storage utilization.</p> <p>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.</p>	
cpuUtilization	Percentage	Overall percentage of CPU utilization across task managers. For example, if there are five task managers, Kinesis Data Analytics publishes five samples of this metric per reporting interval.	Application	You can use this metric to monitor minimum, average, and maximum CPU utilization in your application.	
heapMemoryUtilization	Percentage	Overall heap memory utilization across task managers. For example, if there are five task managers, Kinesis Data Analytics publishes five samples of this metric per reporting interval.	Application	<p>You can use this metric to monitor minimum, average, and maximum heap memory utilization in your application. This value is calculated for all task managers using the following formula:</p> <div> <math display="block">\left( \frac{\text{Heap.Used}}{\text{Heap.Committed}} \right)</math> </div>	

Metric	Unit	Description	Level	Usage Notes	
oldGenerationGCTime	Milliseconds	The total time spent performing old garbage collection operations.	Application	You can use this metric to monitor sum, average, and maximum garbage collection time.	
oldGenerationGCCount	Count	The total number of old garbage collection operations that have occurred across all task managers.	Application		
threadCount	Count	The total number of live threads used by the application.	Application	This metric measures the number of threads used by the application code. This is not the same as application parallelism.	
numRecordsIn	Count	The total number of records this application, operator, or task has received.	Application, Operator, Task, Parallelism	The metric's Level specifies whether this metric measures the total number of records the entire application, a specific operator, or a specific task has received.	
numRecordsInPerSecond	Count/Second	The total number of records this application, operator or task has received per second.	Application, Operator, Task, Parallelism	The metric's Level specifies whether this metric measures the total number of records the entire application, a specific operator, or a specific task has received per second.	

Metric	Unit	Description	Level	Usage Notes	
numRecordsOut	Count	The total number of records this application, operator or task has emitted.	Application, Operator, Task, Parallelism	The metric's Level specifies whether this metric measures the total number of records the entire application, a specific operator, or a specific task has emitted.	
numRecordsOutPerSecond	Count/Second	The total number of records this application, operator or task has emitted per second.	Application, Operator, Task, Parallelism	The metric's Level specifies whether this metric measures the total number of records the entire application, a specific operator, or a specific task has emitted per second.	
numLateRecordsDropped	Count	The number of records this operator or task has dropped due to arriving late.	Application, Operator, Task, Parallelism		
currentInputWatermark	Milliseconds	The last watermark this application/operator/task/thread has received	Application, Operator, Task, Parallelism	This record is only emitted for dimensions with two inputs. This is the minimum value of the last received watermarks.	
currentOutputWatermark	Milliseconds	The last watermark this application/operator/task/thread has emitted	Application, Operator, Task, Parallelism		

Metric	Unit	Description	Level	Usage Notes	
managedMemoryUsed	Bytes	The amount of managed memory currently used.	Application, Operator, Task, Parallelism	<p>*Available for KDA applications running Flink version 1.13.</p> <p>This relates to memory managed by Flink outside the Java heap. It is used for the RocksDB state backend, and is also available to applications.</p>	
managedMemoryTotal	Bytes	The total amount of managed memory.	Application, Operator, Task, Parallelism	<p>*Available for KDA applications running Flink version 1.13.</p> <p>This relates to memory managed by Flink outside the Java heap. It is used for the RocksDB state backend, and is also available to applications.</p>	
managedMemoryUsedPercentage	Percentage*	Derived by $\frac{\text{managedMemoryUsed}}{\text{managedMemoryTotal}}$	Application, Operator, Task, Parallelism	<p>*Available for KDA applications running Flink version 1.13.</p> <p>This relates to memory managed by Flink outside the Java heap. It is used for the RocksDB state backend, and is also available to applications.</p>	

Metric	Unit	Description	Level	Usage Notes	
idleTimeMsPerSecond	Milliseconds	The time (in milliseconds) this task or operator is idle (has no data to process) per second. Idle time excludes back pressured time, so if the task is back pressured it is not idle.	Task, Operator, Parallelism	*Available for KDA applications running Flink version 1.13.  These metrics can be useful in identifying bottlenecks in an application.	
backPressuredTimeMsPerSecond	Milliseconds	*The time (in milliseconds) this task or operator is back pressured per second.	Task, Operator, Parallelism	*Available for KDA applications running Flink version 1.13.  These metrics can be useful in identifying bottlenecks in an application.	
busyTimeMsPerSecond	Milliseconds	The time (in milliseconds) this task or operator is busy (neither idle nor back pressured) per second. Can be NaN, if the value could not be calculated.	Task, Operator, Parallelism	*Available for KDA applications running Flink version 1.13.  These metrics can be useful in identifying bottlenecks in an application.	

## Kinesis Data Streams Connector Metrics

AWS emits all records for Kinesis Data Streams in addition to the following:

Metric	Unit	Description	Level	Usage Notes
millisBehindLatest	Milliseconds	The number of milliseconds the consumer is behind the head of the stream, indicating how far behind current time the consumer is.	Application (for Stream), Parallelism (for ShardId)	<ul style="list-style-type: none"> <li>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</li> </ul>

Metric	Unit	Description	Level	Usage Notes
				by stream name and shard id. <ul style="list-style-type: none"> <li>A value of -1 indicates that the service has not yet reported a value for the metric.</li> </ul>
bytesRequestedPerBatch	Bytes	The bytes requested in a single call to <code>getRecords</code> .	Application (for Stream), Parallelism (for ShardId)	

## Amazon MSK Connector Metrics

AWS emits all records for Amazon MSK in addition to the following:

Metric	Unit	Description	Level	Usage Notes
currentoffsets	N/A	The consumer's current read offset, for each partition. A particular partition's metric can be specified by topic name and partition id.	Application (for Topic), Parallelism (for PartitionId)	
commitsFailed	N/A	The total number of offset commit failures to Kafka, if offset committing and checkpointing are enabled.	Application, Operator, Task, Parallelism	Committing offsets back to Kafka is only a means to expose consumer progress, so a commit failure does not affect the integrity of Flink's checkpointed partition offsets.
commitsSucceeded	N/A	The total number of successful offset commits to Kafka, if offset committing and checkpointing are enabled.	Application, Operator, Task, Parallelism	
committedoffsets	N/A	The last successfully committed offsets to Kafka, for	Application (for Topic), Parallelism (for PartitionId)	

Metric	Unit	Description	Level	Usage Notes
		each partition. A particular partition's metric can be specified by topic name and partition id.		
records_lag_max	Count	The maximum lag in terms of number of records for any partition in this window	Application, Operator, Task, Parallelism	
bytes_consumed_r	Bytes	The average number of bytes consumed per second for a topic	Application, Operator, Task, Parallelism	

## Apache Zeppelin Metrics

For Studio notebooks, AWS emits the following metrics at the application level: `KPUs`, `cpuUtilization`, `heapMemoryUtilization`, `oldGenerationGCtime`, `oldGenerationGCCount`, and `threadCount`. In addition, it emits the metrics shown in the following table, also at the application level.

Metric	Unit	Description	Prometheus name
zeppelinCpuUtilization	Percentage	Overall percentage of CPU utilization in the Apache Zeppelin server.	process_cpu_usage
zeppelinHeapMemoryUtilization	Percentage	Overall percentage of heap memory utilization for the Apache Zeppelin server.	jvm_memory_used_bytes
zeppelinThreadCount	Count	The total number of live threads used by the Apache Zeppelin server.	jvm_threads_live_threads
zeppelinWaitingJobs	Count	The number of queued Apache Zeppelin jobs waiting for a thread.	jetty_threads_jobs
zeppelinServerUptime	Seconds	The total time that the server has been up and running.	process_uptime_seconds

## 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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the **CloudWatch Metrics by Category** pane for 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.

```
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 Apache Flink supports the following metrics levels:

- **Application:** The application only reports the highest level of metrics for each application. Kinesis Data Analytics metrics are published at the Application level by default.
- **Task:** The application reports task-specific metric dimensions for metrics defined with the Task metric reporting level, such as number of records in and out of the application per second.
- **Operator:** The application reports operator-specific metric dimensions for metrics defined with the Operator metric reporting level, 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.

#### Note

You should only use this metric level for troubleshooting because of the amount of metric data that the service generates. You can only set this metric level using the CLI. This metric level is not available in the console.

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**:

```
{
  "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 [Custom Log Levels](#) in the [Apache Log4j](#) documentation.

## Using Custom Metrics with Amazon Kinesis Data Analytics for Apache Flink

Kinesis Data Analytics for Apache Flink exposes 19 metrics to CloudWatch, including metrics for resource usage and throughput. In addition, you can create your own metrics to track application-specific data, such as processing events or accessing external resources.

**This topic contains the following sections:**

- [How It Works](#) (p. 226)
- [Examples](#) (p. 227)
- [Viewing Custom Metrics](#) (p. 228)

### How It Works

Custom metrics in Kinesis Data Analytics use the Apache Flink metric system. Apache Flink metrics have the following attributes:

- **Type:** A metric's type describes how it measures and reports data. Available Apache Flink metric types include Count, Gauge, Histogram, and Meter. For more information about Apache Flink metric types, see [Metric Types](#).

#### Note

AWS CloudWatch Metrics does not support the Histogram Apache Flink metric type. CloudWatch can only display Apache Flink metrics of the Count, Gauge, and Meter types.

- **Scope:** A metric's scope consists of its identifier and a set of key-value pairs that indicate how the metric will be reported to CloudWatch. A metric's identifier consists of the following:
  - A system scope, which indicates the level at which the metric is reported (e.g. Operator).
  - A user scope, that defines attributes such as user variables or the metric group names. These attributes are defined using `MetricGroup.addGroup(key, value)` or `MetricGroup.addGroup(name)`.

For more information about metric scope, see [Scope](#).

For more information about Apache Flink metrics, see [Metrics](#) in the [Apache Flink documentation](#).

To create a custom metric in your Kinesis Data Analytics for Apache Flink application, you can access the Apache Flink metric system from any user function that extends `RichFunction` by calling

[GetMetricGroup](#). This method returns a [MetricGroup](#) object you can use to create and register custom metrics. Kinesis Data Analytics reports all metrics created with the group key `KinesisAnalytics` to CloudWatch. Custom metrics that you define have the following characteristics:

- Your custom metric has a metric name and a group name. These names must consist of alphanumeric characters.
- Attributes that you define in user scope (except for the `KinesisAnalytics` metric group) are published as CloudWatch dimensions.
- Custom metrics are published at the `Application` level by default.
- Dimensions (Task/ Operator/ Parallelism) are added to the metric based on the application's monitoring level. You set the application's monitoring level using the [MonitoringConfiguration](#) parameter of the [CreateApplication](#) action, or the [MonitoringConfigurationUpdate](#) parameter of the [UpdateApplication](#) action.

## Examples

The following code examples demonstrate how to create a mapping class the creates and increments a custom metric, and how to implement the mapping class in your application by adding it to a `DataStream` object.

### Record Count Custom Metric

The following code example demonstrates how to create a mapping class that creates a metric that counts records in a data stream (the same functionality as the `numRecordsIn` metric):

```
private static class NoOpMapperFunction extends RichMapFunction<String, String> {
    private transient int valueToExpose = 0;
    private final String customMetricName;

    public NoOpMapperFunction(final String customMetricName) {
        this.customMetricName = customMetricName;
    }

    @Override
    public void open(Configuration config) {
        getRuntimeContext().getMetricGroup()
            .addGroup("kinesisanalytics")
            .addGroup("Program", "RecordCountApplication")
            .addGroup("NoOpMapperFunction")
            .gauge(customMetricName, (Gauge<Integer>) () -> valueToExpose);
    }

    @Override
    public String map(String value) throws Exception {
        valueToExpose++;
        return value;
    }
}
```

In the preceding example, the `valueToExpose` variable is incremented for each record that the application processes.

After defining your mapping class, you then create an in-application stream that implements the map:

```
DataStream<String> noopMapperFunctionAfterFilter =
    kinesisProcessed.map(new NoOpMapperFunction("FilteredRecords"));
```

For the complete code for this application, see [Record Count Custom Metric Application](#).

## Word Count Custom Metric

The following code example demonstrates how to create a mapping class that creates a metric that counts words in a data stream:

```
private static final class Tokenizer extends RichFlatMapFunction<String, Tuple2<String, Integer>> {

    private transient Counter counter;

    @Override
    public void open(Configuration config) {
        this.counter = getRuntimeContext().getMetricGroup()
            .addGroup("kinesisanalytics")
            .addGroup("Service", "WordCountApplication")
            .addGroup("Tokenizer")
            .counter("TotalWords");
    }

    @Override
    public void flatMap(String value, Collector<Tuple2<String, Integer>>out) {
        // normalize and split the line
        String[] tokens = value.toLowerCase().split("\\W+");

        // emit the pairs
        for (String token : tokens) {
            if (token.length() > 0) {
                counter.inc();
                out.collect(new Tuple2<>(token, 1));
            }
        }
    }
}
```

In the preceding example, the counter variable is incremented for each word that the application processes.

After defining your mapping class, you then create an in-application stream that implements the map:

```
// Split up the lines in pairs (2-tuples) containing: (word,1), and
// group by the tuple field "0" and sum up tuple field "1"
DataStream<Tuple2<String, Integer>> wordCountStream = input.flatMap(new
    Tokenizer()).keyBy(0).sum(1);

// Serialize the tuple to string format, and publish the output to kinesis sink
wordCountStream.map(tuple -> tuple.toString()).addSink(createSinkFromStaticConfig());
```

For the complete code for this application, see [Word Count Custom Metric Application](#).

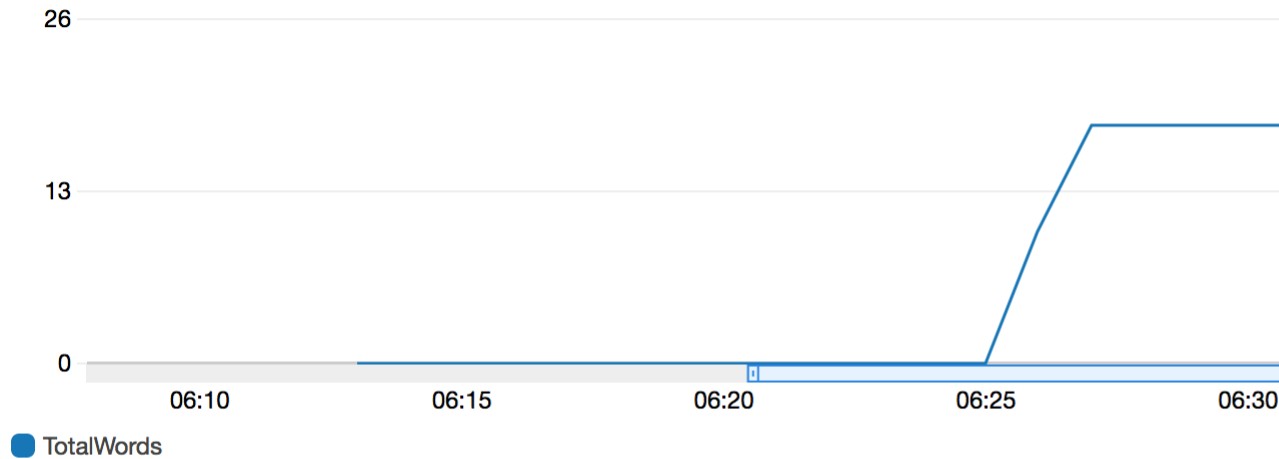
## Viewing Custom Metrics

Custom metrics for your application appear in the CloudWatch Metrics console in the **AWS/KinesisAnalytics** dashboard, under the **Application** metric group.

TotalWords 

1h 3h 12h 1d 3d 1w custom ▾

No unit



All metrics	Graphed metrics (1)	Graph options	Source
Ohio ▾	All > AWS/KinesisAnalytics > Application, Service, Tokenizer <input type="text" value="Search for any"/>		
<input checked="" type="checkbox"/>	Application (1)	Service	Tokenizer
<input checked="" type="checkbox"/>	flink-wordcount-application	WordCountApplication	Tokenizer

## Using CloudWatch Alarms with Amazon Kinesis Data Analytics for Apache Flink

Using Amazon CloudWatch metric alarms, you watch a CloudWatch metric over a time period that you specify. The alarm performs one or more actions based on the value of the metric or expression relative to a threshold over a number of time periods. An example of an action is sending a notification to an Amazon Simple Notification Service (Amazon SNS) topic.

For more information about CloudWatch alarms, see [Using Amazon CloudWatch Alarms](#).

### Recommended Alarms

This section contains the recommended alarms for monitoring Kinesis Data Analytics applications.

The table describes the recommended alarms and has the following columns:

- **Metric Expression:** The metric or metric expression to test against the threshold.

- **Statistic:** The statistic used to check the metric—for example, **Average**.
- **Threshold:** Using this alarm requires you to determine a threshold that defines the limit of expected application performance. You need to determine this threshold by monitoring your application under normal conditions.
- **Description:** Causes that might trigger this alarm, and possible solutions for the condition.

Metric Expression	Statistic	Threshold	Description
downtime > 0	Average	0	Recommended for all applications. The <code>Downtime</code> metric measures the duration of an outage. A downtime greater than zero indicates that the application has failed. For troubleshooting, see <a href="#">Application is Restarting (p. 313)</a> .
RATE (numberOfFailedCheckpoints) > 0	Average	0	Recommended for all applications. Use this metric to monitor application health and checkpointing progress. The application saves state data to checkpoints when it's healthy. Checkpointing can fail due to timeouts if the application isn't making progress in processing the input data. For troubleshooting, see <a href="#">Checkpointing is timing out (p. 316)</a> .
Operator.numRecordsOverSecond < <b>threshold</b>	Average	The minimum number of records emitted from the application during normal conditions.	Recommended for all applications. Falling below this threshold can indicate that the application isn't making expected progress on the input data. For troubleshooting, see <a href="#">Throughput is Too Slow (p. 315)</a> .
records_lag_max  millsBehindLatest > <b>threshold</b>	Maximum	The maximum expected latency during normal conditions.	Recommended for all applications. Use the <code>records_lag_max</code> metric for a Kafka source, or the <code>millsBehindLatest</code> for a Kinesis stream

Metric Expression	Statistic	Threshold	Description
<code>lastCheckpointDuration</code> > <b>threshold</b>	Maximum	The maximum expected checkpoint duration during normal conditions.	source. Rising above this threshold can indicate that the application isn't making expected progress on the input data. For troubleshooting, see <a href="#">Throughput is Too Slow</a> (p. 315).
<code>lastCheckpointSize</code> > <b>threshold</b>	Maximum	The maximum expected checkpoint size during normal conditions.	If the <code>lastCheckpointSize</code> continuously increases, rising above this threshold can indicate that the application isn't making expected progress on the input data, or that there are problems with application health such as backpressure. For troubleshooting, see <a href="#">Application State Data is Accumulating</a> (p. 316).

Metric Expression	Statistic	Threshold	Description
<code>heapMemoryUtilization &gt; threshold</code>	Maximum	The maximum expected <code>heapMemoryUtilization</code> size during normal conditions, with a recommended value of 90 percent.	You can use this metric to monitor the maximum memory utilization of task managers across the application. If the application reaches this threshold, you need to provision more resources. You do this by enabling automatic scaling or increasing the application parallelism. For more information about increasing resources, see <a href="#">Scaling (p. 30)</a> .
<code>cpuUtilization &gt; threshold</code>	Maximum	The maximum expected <code>cpuUtilization</code> size during normal conditions, with a recommended value of 80 percent.	You can use this metric to monitor the maximum CPU utilization of task managers across the application. If the application reaches this threshold, you need to provision more resources. You do this by enabling automatic scaling or increasing the application parallelism. For more information about increasing resources, see <a href="#">Scaling (p. 30)</a> .
<code>threadsCount &gt; threshold</code>	Maximum	The maximum expected <code>threadsCount</code> size during normal conditions.	You can use this metric to watch for thread leaks in task managers across the application. If this metric reaches this threshold, check your application code for threads being created without being closed.



Metric Expression	Statistic	Threshold	Description
<code>(oldGarbageCollectionTime * 100)/60_000 over 1 min period') &gt; threshold</code>	Maximum	The maximum expected oldGarbageCollectionTime duration. We recommend setting a threshold such that typical garbage collection time is 60 percent of the specified threshold, but the correct threshold for your application will vary.	If this metric is continually increasing, this can indicate that there is a memory leak in task managers across the application.
<code>RATE(oldGarbageCollectionTime) &gt; threshold</code>	Maximum	The maximum expected oldGarbageCollectionTime under normal conditions. The correct threshold for your application will vary.	If this metric is continually increasing, this can indicate that there is a memory leak in task managers across the application.
<code>Operator.currentOutputWatermark - Operator.currentInputWatermark &gt; threshold</code>	Minimum	The minimum expected watermark increment under normal conditions. The correct threshold for your application will vary.	If this metric is continually increasing, this can indicate that either the application is processing increasingly older events, or that an upstream subtask has not sent a watermark in an increasingly long time.

## Writing Custom Messages to CloudWatch Logs

You can write custom messages to your 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 \(p. 233\)](#)
- [Write to CloudWatch Logs Using SLF4J \(p. 234\)](#)

## Write to CloudWatch Logs Using Log4J

1. Add the following dependencies to your application's `pom.xml` file:

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.6.1</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
```

```
<version>2.6.1</version>
</dependency>
```

2. Include the object from the library:

```
import org.apache.logging.log4j.Logger;
```

3. Instantiate the `Logger` object, passing in your application class:

```
private static final Logger log = Logger.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.

```
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",
  "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"
}
```

## Write to CloudWatch Logs Using SLF4J

1. Add the following dependency to your application's `pom.xml` file:

```
<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:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

3. Instantiate the `Logger` object, passing in your application class:

```
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.

```
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",
  "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"
}
```

## Logging Kinesis Data Analytics API Calls with AWS CloudTrail

Amazon Kinesis Data Analytics is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Kinesis Data Analytics. CloudTrail captures all API calls for Kinesis Data Analytics as events. The calls captured include calls from the Kinesis Data Analytics console and code calls to the Kinesis Data Analytics API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Kinesis Data Analytics. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Kinesis Data Analytics, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

### 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.

```
{
  "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": {
          "logStreamARN": "arn:aws:logs:us-east-1:012345678910:log-
group:cloudtrail-test:log-stream:flink-cloudwatch"
        }
      },
      "responseElements": {
        "cloudWatchLoggingOptionDescriptions": [
          {
            "cloudWatchLoggingOptionId": "2.1",
            "logStreamARN": "arn:aws:logs:us-east-1:012345678910:log-
group:cloudtrail-test:log-stream:flink-cloudwatch"
          }
        ],
        "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-e789c4e9a3ca",  
      "eventID": "90ffe8e4-9e47-48c9-84e1-4f2d427d98a5",  
      "eventType": "AwsApiCall",  
      "apiVersion": "2018-05-23",  
      "recipientAccountId": "012345678910"  
    }  
  ]  
}
```

# Tuning Performance in Amazon Kinesis Data Analytics for Apache Flink

This topic describes techniques to monitor and improve the performance of your Kinesis Data Analytics application.

## Topics

- [Troubleshooting Performance \(p. 238\)](#)
- [Performance Best Practices \(p. 240\)](#)
- [Monitoring Performance \(p. 242\)](#)

## Troubleshooting Performance

This section contains a list of symptoms that you can check to diagnose and fix performance issues.

If your data source is a Kinesis stream, performance issues typically present as a high or increasing `MillisBehindLatest` metric. For other sources, you can check a similar metric that represents lag in reading from the source.

## The Data Path

When investigating a performance issue with your application, consider the entire path that your data takes. The following application components may become performance bottlenecks and create backpressure if they are not properly designed or provisioned:

- **Data sources and destinations:** Ensure that the external resources your application interacts with are properly provisioned for the throughput your application will experience.
- **State data:** Ensure that your application doesn't interact with the state store too frequently.

You can optimize the serializer your application is using. The default Kryo serializer can handle any serializable type, but you can use a more performant serializer if your application only stores data in POJO types. For information about Apache Flink serializers, see [Data Types & Serialization](#) in the [Apache Flink documentation](#).

- **Operators:** Ensure that the business logic implemented by your operators isn't too complicated, or that you aren't creating or using resources with every record processed. Also ensure that your application isn't creating sliding or tumbling windows too frequently.

## Performance Troubleshooting Solutions

This section contains potential solutions to performance issues.

## Topics

- [CloudWatch Monitoring Levels \(p. 239\)](#)
- [Application CPU Metric \(p. 239\)](#)

- [Application Parallelism \(p. 239\)](#)
- [Application Logging \(p. 239\)](#)
- [Operator Parallelism \(p. 239\)](#)
- [Application Logic \(p. 240\)](#)
- [Application Memory \(p. 240\)](#)

## CloudWatch Monitoring Levels

Verify that the CloudWatch Monitoring Levels are not set to too verbose a setting.

The Debug Monitoring Log Level setting generates a large amount of traffic, which can create backpressure. You should only use it while actively investigating issues with the application.

If your application has a high `Parallelism` setting, using the `Parallelism Monitoring Metrics Level` will similarly generate a large amount of traffic that can lead to backpressure. Only use this metrics level when `Parallelism` for your application is low, or while investigating issues with the application.

For more information, see [Application Monitoring Levels \(p. 211\)](#).

## Application CPU Metric

Check the application's CPU metric. If this metric is above 75 percent, you can allow the application to allocate more resources for itself by enabling auto scaling.

If auto scaling is enabled, the application allocates more resources if CPU usage is over 75 percent for 15 minutes. For more information about scaling, see the [Manage scaling properly \(p. 240\)](#) section following, and the [Scaling \(p. 30\)](#).

### Note

An application will only scale automatically in response to CPU usage. The application will not auto scale in response to other system metrics, such as `heapMemoryUtilization`. If your application has a high level of usage for other metrics, increase your application's parallelism manually.

## Application Parallelism

Increase the application's parallelism. You update the application's parallelism using the `ParallelismConfigurationUpdate` parameter of the [UpdateApplication](#) action.

The maximum KPIUs for an application is 32 by default, and can be increased by requesting a limit increase.

It is important to also assign parallelism to each operator based on its workload, rather than just increasing application parallelism alone. See [Operator Parallelism \(p. 239\)](#) following.

## Application Logging

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 about reading application logs, see [the section called "Analyzing Logs" \(p. 213\)](#).

## Operator Parallelism

Verify that your application's workload is distributed evenly among worker processes.

For information about tuning the workload of your application's operators, see [Operator scaling](#) (p. 241).

## Application Logic

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. An external dependency can also hinder performance if it is not performant or not reliably accessible, which may lead to the external dependency returning HTTP 500 errors.

If your application uses 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](#).

## Application Memory

Check your application for resource leaks. If your application is not properly disposing of threads or memory, you might see the `MillisBehindLatest`, `CheckpointSize`, and `CheckpointDurationMetric` spiking or gradually increasing. This condition may also lead to task manager or job manager failures.

# Performance Best Practices

This section describes special considerations for designing an application for performance.

## Manage scaling properly

This section contains information about managing application-level and operator-level scaling.

**This section contains the following topics:**

- [Manage application scaling properly](#) (p. 240)
- [Manage operator scaling properly](#) (p. 241)

## Manage application scaling properly

You can use autoscaling to handle unexpected spikes in application activity. Your application's KPIs will increase automatically if the following criteria are met:

- Autoscaling is enabled for the application.
- CPU usage remains above 75 percent for 15 minutes.

If autoscaling is enabled, but CPU usage does not remain at this threshold, the application will not scale up KPIs. If you experience a spike in CPU usage that does not meet this threshold, or a spike in a different usage metric such as `heapMemoryUtilization`, increase scaling manually to allow your application to handle activity spikes.

### Note

If the application has automatically added more resources through auto scaling, the application will release the new resources after a period of inactivity. Downscaling resources will temporarily affect performance.

For more information about scaling, see [Scaling](#) (p. 30).



## Manage operator scaling properly

You can improve your application's performance by verifying that your application's workload is distributed evenly among worker processes, and that the operators in your application have the system resources they need to be stable and performant.

You can set the parallelism for each operator in your application's code using the `parallelism` setting. If you don't set the parallelism for an operator, it will use the application-level parallelism setting. Operators that use the application-level parallelism setting can potentially use all of the system resources available for the application, making the application unstable.

To best determine the parallelism for each operator, consider the operator's relative resource requirements compared to the other operators in the application. Set operators that are more resource-intensive to a higher operator parallelism setting than less resource-intensive operators.

The total operator parallelism for the application is the sum of the parallelism for all the operators in the application. You tune the total operator parallelism for your application by determining the best ratio between it and the total task slots available for your application. A typical stable ratio of total operator parallelism to task slots is 4:1, that is, the application has one task slot available for every four operator subtasks available. An application with more resource intensive operators may need a ratio of 3:1 or 2:1, while an application with less resource-intensive operators may be stable with a ratio of 10:1.

You can set the ratio for the operator using [Runtime Properties \(p. 21\)](#), so you can tune the operator's parallelism without compiling and uploading your application code.

The following code example demonstrates how to set operator parallelism as a tunable ratio of the current application parallelism:

```
Map<String, Properties> applicationProperties =
    KinesisAnalyticsRuntime.getApplicationProperties();
operatorParallelism =
    StreamExecutionEnvironment.getParallelism() /
    Integer.getInteger(

    applicationProperties.get("OperatorProperties").getProperty("MyOperatorParallelismRatio")
    );
```

For information about subtasks, task slots, and other application resources, see [Application Resources \(p. 8\)](#).

To control the distribution of workload across your application's worker processes, use the `Parallelism` setting and the `KeyBy` partition method. For more information, see the following topics in the [Apache Flink documentation](#):

- [Parallel Execution](#)
- [DataStream Transformations](#)

## Monitor external dependency resource usage

If there is a performance bottleneck in a destination (such as Kinesis Streams, Kinesis Data Firehose, DynamoDB or OpenSearch Service), your application will experience backpressure. Verify that your external dependencies are properly provisioned for your application throughput.

### Note

Failures in other services can cause failures in your application. If you are seeing failures in your application, check the CloudWatch logs for your destination services for failures.

## Run your Apache Flink application locally

To troubleshoot memory issues, you can run your application in a local Flink installation. This will give you access to debugging tools such as the stack trace and heap dumps that are not available when running your application in Kinesis Data Analytics.

For information about creating a local Flink installation, see [Local Setup Tutorial](#) in the [Apache Flink documentation](#).

## Monitoring Performance

This section describes tools for monitoring an application's performance.

### Performance Monitoring using CloudWatch Metrics

You monitor your application's resource usage, throughput, checkpointing, and downtime using CloudWatch metrics. For information about using CloudWatch metrics with your Kinesis Data Analytics application, see [Metrics and Dimensions \(p. 215\)](#).

### Performance Monitoring using CloudWatch Logs and Alarms

You monitor error conditions that could potentially cause performance issues using CloudWatch Logs.

Error conditions appear in log entries as Apache Flink job status changes from the `RUNNING` status to the `FAILED` status.

You use CloudWatch alarms to create notifications for performance issues, such as resource use or checkpoint metrics above a safe threshold, or unexpected application status changes.

For information about creating CloudWatch alarms for a Kinesis Data Analytics application, see [Alarms \(p. 229\)](#).

# Kinesis Data Analytics for Apache Flink and Studio Notebook Quota

When working with Amazon Kinesis Data Analytics for Apache Flink, note the following quota:

- 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 quota 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 quota, see **To request a quota increase** in [Service Quotas](#).

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 GiB of memory. For each KPU, the service also provisions 50 GiB of running application storage.

- You can create up to 1,000 Kinesis Data Analytics [Snapshots \(p. 26\)](#) per application.
- You can assign up to 50 tags per application.
- The maximum size for an application JAR file is 512 MiB. If you exceed this quota, your application will fail to start.

For Studio notebooks, the following quotas apply. To request higher quotas, [create a support case](#).

- `websocketMessageSize` = 5 MiB
- `noteSize` = 5 MiB
- `noteCount` = 1000

# Kinesis Data Analytics for Apache Flink Maintenance

Kinesis Data Analytics patches your applications periodically with operating-system and container-image security updates to maintain compliance and meet AWS security goals. The following table lists the default time window during which Kinesis Data Analytics performs this type of maintenance. Maintenance for your application might happen at any time during the time window that corresponds to your Region. Your application might experience a downtime of 10 to 30 seconds during this maintenance process. However, the actual downtime duration depends on the application state. For information on how to minimize the impact of this downtime, see [the section called “Fault tolerance: checkpoints and savepoints”](#) (p. 246).

To change the time window during which Kinesis Data Analytics performs maintenance on your application, use the [UpdateApplicationMaintenanceConfiguration](#) API.

Region	Maintenance time window
US East (N. Virginia)	03:00–11:00 UTC
AWS GovCloud (US-East)	03:00–11:00 UTC
US East (Ohio)	03:00–11:00 UTC
US West (N. California)	06:00–14:00 UTC
US West (Oregon)	06:00–14:00 UTC
AWS GovCloud (US-West)	06:00–14:00 UTC
Asia Pacific (Hong Kong)	13:00–21:00 UTC
Asia Pacific (Mumbai)	16:30–00:30 UTC
Asia Pacific (Seoul)	13:00–21:00 UTC
Asia Pacific (Singapore)	14:00–22:00 UTC
Asia Pacific (Sydney)	12:00–20:00 UTC
Asia Pacific (Tokyo)	13:00–21:00 UTC
Canada (Central)	03:00–11:00 UTC
China (Beijing)	13:00–21:00 UTC
China (Ningxia)	13:00–21:00 UTC
Europe (Frankfurt)	06:00–14:00 UTC
Europe (Ireland)	22:00–06:00 UTC
Europe (London)	22:00–06:00 UTC
Europe (Milan)	21:00–05:00 UTC

Region	Maintenance time window
Europe (Paris)	23:00–07:00 UTC
Europe (Stockholm)	23:00–07:00 UTC
Middle East (Bahrain)	13:00–21:00 UTC
South America (São Paulo)	19:00–03:00 UTC

# Best Practices for Kinesis Data Analytics for Apache Flink

This section contains information and recommendations for developing stable, performant Amazon Kinesis Data Analytics applications.

## Topics

- [Fault tolerance: checkpoints and savepoints \(p. 246\)](#)
- [Performance and parallelism \(p. 246\)](#)
- [Logging \(p. 247\)](#)
- [Coding \(p. 247\)](#)
- [Studio notebook refresh interval \(p. 247\)](#)
- [Studio notebook optimum performance \(p. 247\)](#)

## Fault tolerance: checkpoints and savepoints

Use checkpoints and savepoints to implement fault tolerance in your Kinesis Data Analytics for Apache Flink 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 during scheduled maintenance, as well as in case of unexpected failures due to service issues, application dependency failures, and other issues. For information about scheduled maintenance, see [Maintenance \(p. 244\)](#).
- 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.

### Note

We recommend that your application create a snapshot several times a day to restart properly with correct state data. The correct frequency for your snapshots depends on your application's business logic. Taking frequent snapshots allows you to recover more recent data, but increases cost and requires more system resources.

For information about monitoring application downtime, see [Metrics and Dimensions \(p. 215\)](#).

For more information about implementing fault tolerance, see [Fault Tolerance \(p. 24\)](#).

## 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 \(p. 30\)](#).

## 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.
- Create CloudWatch alarms to notify you when your application is not running properly. For more information, see [Alarms \(p. 229\)](#)

For more information about implementing logging, see [Logging and Monitoring \(p. 206\)](#).

## 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 `system.exit()` in your application code, in either your application's `main` method or in user-defined functions. If you want to shut down your application from within code, throw an exception derived from `Exception` or `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 `main` method, the service will wrap it in a `ProgramInvocationException` when the application transitions to the `RUNNING` status, 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 `java.util.concurrent.ExecutionException`. For more information about shading your application JAR file, see [Apache Maven Shade Plugin](#).

## Studio notebook refresh interval

If you change the paragraph result refresh interval, set it to a value that is at least 1000 milliseconds.

## Studio notebook optimum performance

We tested with the following statement and got the best performance when `events-per-second` multiplied by `number-of-keys` was under 25,000,000. This was for `events-per-second` under 150,000.

```
SELECT key, sum(value) FROM key-values GROUP BY key
```



# Earlier Version Information for Kinesis Data Analytics for Apache Flink

This topic contains information about using Kinesis Data Analytics with older versions of Apache Flink. The versions of Apache Flink that Kinesis Data Analytics supports are **1.13.2** (recommended), **1.11.3**, **1.11.1**, **1.8.2** and **1.6.2**.

We recommend that you use the latest supported version of Apache Flink with your Kinesis Data Analytics application. Apache Flink version 1.13.2 has the following features:

- Support for [Apache Flink Table API & SQL](#)
- Support for Python applications.
- Support for Java version 11 and Scala version 2.12
- An improved memory model
- RocksDB optimizations for increased application stability
- Support for task manager and stack traces in the Apache Flink Dashboard.

**This topic contains the following sections:**

- [Using the Apache Flink Kinesis Streams Connector with previous Apache Flink versions \(p. 249\)](#)
- [Building Applications with Apache Flink 1.8.2 \(p. 250\)](#)
- [Building Applications with Apache Flink 1.6.2 \(p. 251\)](#)
- [Upgrading Applications \(p. 251\)](#)
- [Available Connectors in Apache Flink 1.6.2 and 1.8.2 \(p. 252\)](#)
- [Getting Started: Flink 1.11.3 \(p. 252\)](#)
- [Getting Started: Flink 1.8.2 \(p. 269\)](#)
- [Getting Started: Flink 1.6.2 \(p. 285\)](#)

## Using the Apache Flink Kinesis Streams Connector with previous Apache Flink versions

The Apache Flink Kinesis Streams connector was not included in Apache Flink prior to version 1.11. In order for your application to use the Apache Flink Kinesis connector with previous versions of Apache Flink, you must download, compile, and install the version of Apache Flink that your application uses. 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.

**Note**

Ensure that you are building the connector with [KPL version 0.14.0](#) or higher.

To download and install the Apache Flink version 1.8.2 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
```

**Note**

If you are compiling Flink on Microsoft Windows, you need to add the `-Drat.skip=true` parameter.

## Building Applications with Apache Flink 1.8.2

This section contains information about components that you use for building Kinesis Data Analytics applications that work with Apache Flink 1.8.2.

Use the following component versions for Kinesis Data Analytics applications:

Component	Version
Java	1.8 (recommended)
Apache Flink	1.8.2
Kinesis Data Analytics for Flink Runtime (aws-kinesisanalytics-runtime)	1.0.1
Kinesis Data Analytics Flink Connectors (aws-kinesisanalytics-flink)	1.0.1
Apache Maven	3.1

To compile an application using Apache Flink 1.8.2, run Maven with the following parameter:

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

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 for Flink 1.8.2 Getting Started Application](#).

For information about how to build and use application code for a Kinesis Data Analytics application, see [Creating Applications \(p. 3\)](#).

## 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:

Component	Version
Java	1.8 (recommended)
AWS Java SDK	1.11.379
Apache Flink	1.6.2
Kinesis Data Analytics for Flink Runtime (aws-kinesisanalytics-runtime)	1.0.1
Kinesis Data Analytics Flink Connectors (aws-kinesisanalytics-flink)	1.0.1
Apache Maven	3.1
Apache Beam	Not supported with Apache Flink 1.6.2.

### 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 \(p. 3\)](#).

## Upgrading Applications

To upgrade the version of an Kinesis Data Analytics application, you must update your application code, delete the previous application, and create a new application with the updated code. To do this, 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:

```
mvn package -Dflink.version=1.13.2
```

- Delete your existing application. Create your application again, and choose **Apache Flink version 1.13.2 (Recommended version)** for the application's **Runtime**.

**Note**

You cannot use snapshots from your previous application versions.

## Available Connectors in Apache Flink 1.6.2 and 1.8.2

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 \(1.6.2\)](#) in the [Apache Flink documentation \(1.6.2\)](#).
- For information about connectors available in the Apache Flink 1.8.2 framework, see [Connectors \(1.8.2\)](#) in the [Apache Flink documentation \(1.8.2\)](#).

## Getting Started: Flink 1.11.3

This topic contains a version of the [Getting Started \(DataStream API\) \(p. 75\)](#) Tutorial that uses Apache Flink 1.11.3.

This section introduces you to the fundamental concepts of Kinesis Data Analytics for Apache Flink and the DataStream API. 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 Flink Application \(p. 252\)](#)
- [Prerequisites for Completing the Exercises \(p. 253\)](#)
- [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 253\)](#)
- [Step 2: Set Up the AWS Command Line Interface \(AWS CLI\) \(p. 255\)](#)
- [Step 3: Create and Run a Kinesis Data Analytics for Apache Flink Application \(p. 256\)](#)
- [Step 4: Clean Up AWS Resources \(p. 267\)](#)
- [Step 5: Next Steps \(p. 268\)](#)

## Components of a Kinesis Data Analytics for Flink Application

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

A Kinesis Data Analytics 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 \(p. 10\)](#).

- **Operators:** The application processes data by using one or more *operators*. An operator can transform, enrich, or aggregate data. For more information, see [DataStream API Operators \(p. 14\)](#).
- **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 \(p. 11\)](#).

After you create, compile, and package your application code, 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 11](#). 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
```

To get started, go to [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 76\)](#).

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

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

1. [Sign Up for AWS \(p. 253\)](#)
2. [Create an IAM User \(p. 254\)](#)

### Sign Up for AWS

When you sign up for AWS, your account is automatically signed up for all Amazon services, including 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

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

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

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

## Create an IAM User

Services in AWS, such as 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 account ID without any hyphens. For example, if your 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 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 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 \(AWS CLI\) \(p. 255\)](#)

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

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

#### 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 [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 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 Apache Flink Application \(p. 256\)](#)

## Step 3: Create and Run a Kinesis Data Analytics for Apache Flink Application

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

### This section contains the following steps:

- [Create Two Amazon Kinesis Data Streams \(p. 257\)](#)
- [Write Sample Records to the Input Stream \(p. 257\)](#)
- [Download and Examine the Apache Flink Streaming Java Code \(p. 258\)](#)
- [Compile the Application Code \(p. 258\)](#)
- [Upload the Apache Flink Streaming Java Code \(p. 259\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 259\)](#)



- [Next Step \(p. 267\)](#)

## Create Two Amazon Kinesis Data Streams

Before you create a Kinesis Data Analytics 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:

```
import datetime  
import json  
import random  
import boto3  
  
STREAM_NAME = "ExampleInputStream"  
  
def get_data():  
    return {  
        'EVENT_TIME': datetime.datetime.now().isoformat(),  
        'TICKER': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),  
        'PRICE': round(random.random() * 100, 2)}  

```

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

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

2. Later in the tutorial, you run the `stock.py` script to send data to the application.

```
$ 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:

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

2. Navigate to the `amazon-kinesis-data-analytics-java-examples/GettingStarted` 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 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:

```
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 \(p. 21\)](#).

## 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 \(p. 75\)](#).

### 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 -Dflink.version=1.11.3
```

- Use your development environment. See your development environment documentation for details.

#### Note

The provided source code relies on libraries from Java 11. Ensure that your project's Java version is 11.

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. 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 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 \(Console\) \(p. 260\)](#)
- [Create and Run the Application \(AWS CLI\) \(p. 262\)](#)

## 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 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 version 1.11 (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 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (`012345678901`) with your account ID.

```
{
  "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"  
        ],  
        "Resource": [  
            "arn:aws:logs:us-west-2:012345678901:log-group:*"  
        ]  
    },  
    {  
        "Sid": "DescribeLogStreams",  
        "Effect": "Allow",  
        "Action": [  
            "logs:DescribeLogStreams"  
        ],  
        "Resource": [  
            "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/  
MyApplication:log-stream:*"  
        ]  
    },  
    {  
        "Sid": "PutLogEvents",  
        "Effect": "Allow",  
        "Action": [  
            "logs:PutLogEvents"  
        ],  
        "Resource": [  
            "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/  
MyApplication:log-stream:kinesis-analytics-log-stream"  
        ]  
    },  
    {  
        "Sid": "ReadInputStream",  
        "Effect": "Allow",  
        "Action": "kinesis:*",  
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/  
ExampleInputStream"  
    },  
    {  
        "Sid": "WriteOutputStream",  
        "Effect": "Allow",  
        "Action": "kinesis:*",  
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/  
ExampleOutputStream"  
    }  
]  
}
```

## 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**.

- Under **Properties**, for **Group ID**, enter **ProducerConfigProperties**.
- Enter the following application properties and values:

Key	Value
<code>flink.inputstream.initpos</code>	<b>LATEST</b>
<code>aws.region</code>	<b>us-west-2</b>
<code>AggregationEnabled</code>	<b>false</b>
<code>group.id</code>	<b>ProducerConfigProperties</b>

- Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
- For **CloudWatch logging**, select the **Enable** check box.
- 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

The Flink job graph can be viewed by running the application, opening the Apache Flink dashboard, and choosing the desired Flink job.

## 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 Apache Flink 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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "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:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream"
    },
    {
      "Sid": "WriteOutputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleOutputStream"
    }
  ]
}
```

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 Amazon 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 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**.

- 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**.

- On the **Attach permissions policies** page, choose **Next: Review**. You attach permissions policies after you create the role.
- 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.

- 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" \(p. 262\)](#).

- On the **Summary** page, choose the **Permissions** tab.
- Choose **Attach Policies**.
- In the search box, enter **KAREadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- Choose the **KAREadSourceStreamWriteSinkStream** 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

- 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.

```
{
  "ApplicationName": "test",
  "ApplicationDescription": "my java test app",
  "RuntimeEnvironment": "FLINK-1_11",
  "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:

```
aws kinesisanalyticstv2 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`.

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

2. Execute the [StartApplication](#) action with the preceding request to start the application:

```
aws kinesisanalyticstv2 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`.

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

2. Execute the [StopApplication](#) action with the following request to stop the application:

```
aws kinesisanalyticstv2 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" \(p. 206\)](#).

### 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 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`.

```
{ "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 kinesisanalyticstv2 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 file name, 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 [the section called "Create Two Amazon Kinesis Data Streams" \(p. 257\)](#) section.

```
{
  "ApplicationName": "test",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationCodeConfigurationUpdate": {
      "CodeContentUpdate": {
        "S3ContentLocationUpdate": {
          "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
          "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
          "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
        }
      }
    }
  }
}
```

## Next Step

[Step 4: Clean Up AWS Resources \(p. 267\)](#)

## 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 \(p. 267\)](#)
- [Delete Your Kinesis Data Streams \(p. 267\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 268\)](#)
- [Delete Your IAM Resources \(p. 268\)](#)
- [Delete Your CloudWatch Resources \(p. 268\)](#)
- [Next Step \(p. 268\)](#)

## Delete Your Kinesis Data Analytics Application

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Kinesis Data Analytics panel, choose **MyApplication**.
3. 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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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.

## Next Step

[Step 5: Next Steps \(p. 268\)](#)

## Step 5: Next Steps

Now that you've created and run a basic Kinesis Data Analytics application, see the following resources for more advanced Kinesis Data Analytics solutions.

- **The AWS Streaming Data Solution for Amazon Kinesis:** The AWS Streaming Data Solution for Amazon Kinesis automatically configures the AWS services necessary to easily capture, store, process, and deliver streaming data. The solution provides multiple options for solving streaming data use cases. The Kinesis Data Analytics option provides an end-to-end streaming ETL example demonstrating a real-world application that runs analytical operations on simulated New York taxi data. The solution sets up all necessary AWS resources such as IAM roles and policies, a CloudWatch dashboard, and CloudWatch alarms.
- **AWS Streaming Data Solution for Amazon MSK:** The AWS Streaming Data Solution for Amazon MSK provides AWS CloudFormation templates where data flows through producers, streaming storage, consumers, and destinations.
- **Clickstream Lab with Apache Flink and Apache Kafka:** An end to end lab for clickstream use cases using Amazon Managed Streaming for Apache Kafka for streaming storage and Amazon Kinesis Data Analytics for Apache Flink applications for stream processing.

- **Streaming Analytics Workshop:** In this workshop, you build an end-to-end streaming architecture to ingest, analyze, and visualize streaming data in near real-time. You set out to improve the operations of a taxi company in New York City. You analyze the telemetry data of a taxi fleet in New York City in near real-time to optimize their fleet operations.
- **Kinesis Data Analytics for Apache Flink: Examples (p. 113):** This section of this Developer Guide provides examples of creating and working with applications in Kinesis Data Analytics. They include example code and step-by-step instructions to help you create Kinesis Data Analytics applications and test your results.
- **Learn Flink: Hands On Training:** Official introductory Apache Flink training that gets you started writing scalable streaming ETL, analytics, and event-driven applications.

**Note**

Be aware that Kinesis Data Analytics does not support the Apache Flink version (1.12) used in this training. You can use Flink 1.13 in Flink Kinesis Data Analytics.

- **Apache Flink Code Examples:** A GitHub repository of a wide variety of Apache Flink application examples.

## Getting Started: Flink 1.8.2

This topic contains a version of the [Getting Started \(DataStream API\) \(p. 75\)](#) Tutorial that uses Apache Flink 1.8.2.

**Topics**

- [Components of a Kinesis Data Analytics for Flink Application \(p. 75\)](#)
- [Prerequisites for Completing the Exercises \(p. 270\)](#)
- [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 270\)](#)
- [Step 2: Set Up the AWS Command Line Interface \(AWS CLI\) \(p. 272\)](#)
- [Step 3: Create and Run a Kinesis Data Analytics for Apache Flink Application \(p. 273\)](#)
- [Step 4: Clean Up AWS Resources \(p. 284\)](#)

## Components of a Kinesis Data Analytics for Flink Application

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

A Kinesis Data Analytics 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 \(p. 10\)](#).
- **Operators:** The application processes data by using one or more *operators*. An operator can transform, enrich, or aggregate data. For more information, see [DataStream API Operators \(p. 14\)](#).
- **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 \(p. 11\)](#).

After you create, compile, and package your application code, 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.
- To use the Apache Flink Kinesis connector in this tutorial, you must download and install Apache Flink. For details, see [Using the Apache Flink Kinesis Streams Connector with previous Apache Flink versions \(p. 249\)](#).
- 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
```

To get started, go to [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 270\)](#).

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

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

1. [Sign Up for AWS \(p. 270\)](#)
2. [Create an IAM User \(p. 271\)](#)

### Sign Up for AWS

When you sign up for AWS, your AWS account is automatically signed up for all services in AWS, including 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 account, follow these steps to create one.

#### To create an AWS account

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

Part of the sign-up procedure involves receiving a phone call 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 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 account ID without any hyphens. For example, if your 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 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 \(AWS CLI\) \(p. 272\)](#)

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

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

**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 Regions, see [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 Apache Flink Application \(p. 273\)](#)

## Step 3: Create and Run a Kinesis Data Analytics for Apache Flink Application

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

### This section contains the following steps:

- [Create Two Amazon Kinesis Data Streams \(p. 274\)](#)
- [Write Sample Records to the Input Stream \(p. 274\)](#)
- [Download and Examine the Apache Flink Streaming Java Code \(p. 275\)](#)
- [Compile the Application Code \(p. 275\)](#)
- [Upload the Apache Flink Streaming Java Code \(p. 276\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 276\)](#)
- [Next Step \(p. 284\)](#)

## Create Two Amazon Kinesis Data Streams

Before you create a Kinesis Data Analytics 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:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

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

def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
```

```
kinesis_client.put_record(  
    StreamName=stream_name,  
    Data=json.dumps(data),  
    PartitionKey="partitionkey")  
  
if __name__ == '__main__':  
    generate(STREAM_NAME, boto3.client('kinesis'))
```

2. Later in the tutorial, you run the `stock.py` script to send data to the application.

```
$ 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:

```
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_8` 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 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:

```
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 \(p. 21\)](#).

## 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 \(p. 270\)](#).

### Note

**In order to use the Kinesis connector with versions of Apache Flink prior to 1.11, you need to download, build, and install Apache Maven. For more information, see [the section](#)**

called “Using the Apache Flink Kinesis Streams Connector with previous Apache Flink versions” (p. 249).

### 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 -Dflink.version=1.8.2
```

- Use your development environment. See your development environment documentation for details.

#### Note

The provided source code relies on libraries from Java 1.8. Ensure that your project's Java version is 1.8.

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. 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 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. 277)
- [Create and Run the Application \(AWS CLI\)](#) (p. 279)

## 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 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)**.
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 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (`012345678901`) with your account ID.

```
{
  "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"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:*"
        ]
    },
    {
        "Sid": "DescribeLogStreams",
        "Effect": "Allow",
        "Action": [
            "logs:DescribeLogStreams"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:*"
        ]
    },
    {
        "Sid": "PutLogEvents",
        "Effect": "Allow",
        "Action": [
            "logs:PutLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:kinesis-analytics-log-stream"
        ]
    },
    {
        "Sid": "ReadInputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    },
    {
        "Sid": "WriteOutputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
    }
]
```

## 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 **Properties**, for **Group ID**, enter **ProducerConfigProperties**.
  5. Enter the following application properties and values:

Key	Value
<code>flink.inputstream.initpos</code>	<b>LATEST</b>
<code>aws.region</code>	<b>us-west-2</b>
<code>AggregationEnabled</code>	<b>false</b>

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

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**.

### 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 Apache Flink 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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "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:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream"
    },
    {
      "Sid": "WriteOutputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleOutputStream"
    }
  ]
}
```

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 Amazon 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 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" \(p. 279\)](#).

- 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 **KAReadSourceStreamWriteSinkStream** 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.

```
{
  "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:

```
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`.

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

2. Execute the [StartApplication](#) action with the preceding request to start the application:

```
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`.

```
{
  "ApplicationName": "test"
```

```
}
```

2. Execute the [StopApplication](#) action with the following request to stop the application:

```
aws kinesisanalyticstv2 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" \(p. 206\)](#).

### 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 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`.

```
{
  "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 kinesisanalyticstv2 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 file name, 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 [the section called "Create Two Amazon Kinesis Data Streams" \(p. 274\)](#) section.

```
{
  "ApplicationName": "test",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationCodeConfigurationUpdate": {
      "CodeContentUpdate": {
        "S3ContentLocationUpdate": {
          "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
          "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
          "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhyvpvDU"
        }
      }
    }
  }
}
```

## Next Step

[Step 4: Clean Up AWS Resources \(p. 284\)](#)

## 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 \(p. 284\)](#)
- [Delete Your Kinesis Data Streams \(p. 285\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 285\)](#)
- [Delete Your IAM Resources \(p. 285\)](#)
- [Delete Your CloudWatch Resources \(p. 285\)](#)

## Delete Your Kinesis Data Analytics Application

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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.

# Getting Started: Flink 1.6.2

This topic contains a version of the [Getting Started \(DataStream API\) \(p. 75\)](#) Tutorial that uses Apache Flink 1.6.2.

### Topics

- [Components of a Kinesis Data Analytics for Apache Flink application \(p. 286\)](#)
- [Prerequisites for Completing the Exercises \(p. 286\)](#)
- [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 286\)](#)
- [Step 2: Set Up the AWS Command Line Interface \(AWS CLI\) \(p. 289\)](#)
- [Step 3: Create and Run a Kinesis Data Analytics application \(p. 290\)](#)

- [Step 4: Clean Up AWS Resources \(p. 300\)](#)

## Components of a Kinesis Data Analytics for Apache Flink application

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

A Kinesis Data Analytics for Apache Flink 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 \(p. 10\)](#).
- **Operators:** The application processes data by using one or more *operators*. An operator can transform, enrich, or aggregate data. For more information, see [DataStream API Operators \(p. 14\)](#).
- **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 \(p. 11\)](#).

After you create, compile, and package your 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
```

To get started, go to [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 286\)](#).

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

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

1. [Sign Up for AWS \(p. 287\)](#)
2. [Create an IAM User \(p. 287\)](#)

## Sign Up for AWS

When you sign up for AWS, your AWS account is automatically signed up for all services in AWS, including 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

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

Part of the sign-up procedure involves receiving a phone call 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 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 \(AWS CLI\) \(p. 289\)](#)



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

In this step, you download and configure the AWS CLI to use with Kinesis Data Analytics for Apache Flink.

### 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 [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 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 application \(p. 290\)](#)

## Step 3: Create and Run a Kinesis Data Analytics application

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

**This section contains the following steps:**

- [Create Two Amazon Kinesis Data Streams \(p. 290\)](#)
- [Write Sample Records to the Input Stream \(p. 290\)](#)
- [Download and Examine the Apache Flink Streaming Java Code \(p. 291\)](#)
- [Compile the Application Code \(p. 292\)](#)
- [Upload the Apache Flink Streaming Java Code \(p. 292\)](#)
- [Create and Run the Kinesis Data Analytics Application \(p. 293\)](#)

### Create Two Amazon Kinesis Data Streams

Before you create a Kinesis Data Analytics 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:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"

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

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

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

2. Later in the tutorial, you run the `stock.py` script to send data to the application.

```
$ 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:

```
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 Apache Flink 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:

```
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. 21).

## 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. 286).

### Note

**In order to use the Kinesis connector with versions of Apache Flink prior to 1.11, 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. 4).

- 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 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 \(Console\) \(p. 293\)](#)
- [Create and Run the Application \(AWS CLI\) \(p. 296\)](#)

## 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 uses Apache Flink version 1.8.2 or 1.6.2.
- Change the version pulldown to **Apache Flink 1.6**.
  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 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.
3. On the **Summary** page, choose **Edit policy**. Choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account IDs (**012345678901**) with your account ID.

```
{
  "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": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
      ]
    },
    {
      "Sid": "DescribeLogStreams",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:*"
      ]
    },
    {
      "Sid": "PutLogEvents",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:kinesis-analytics-log-stream"
      ]
    },
    {
      "Sid": "ReadInputStream",
```

```
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    },
    {
        "Sid": "WriteOutputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
    }
  ]
}
```

## 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:

Key	Value
<b>flink.inputstream.initpos</b>	<b>LATEST</b>
<b>aws:region</b>	<b>us-west-2</b>
<b>AggregationEnabled</b>	<b>false</b>

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

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**.

## 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 Apache Flink 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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "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:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream"
    },
    {
      "Sid": "WriteOutputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleOutputStream"
    }
  ]
}
```

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 Amazon 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 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" \(p. 296\)](#).

- 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 **KAReadSourceStreamWriteSinkStream** 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.

```
{
  "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:

```
aws kinesisanalyticstv2 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`.

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

2. Execute the [StartApplication](#) action with the preceding request to start the application:

```
aws kinesisanalyticstv2 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`.

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

2. Execute the [StopApplication](#) action with the following request to stop the application:

```
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" \(p. 206\)](#).

### 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 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`.

```
{
  "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.

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 [the section called "Create Two Amazon Kinesis Data Streams" \(p. 290\)](#) section.

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

## 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 \(p. 300\)](#)
- [Delete Your Kinesis Data Streams \(p. 301\)](#)
- [Delete Your Amazon S3 Object and Bucket \(p. 301\)](#)
- [Delete Your IAM Resources \(p. 301\)](#)
- [Delete Your CloudWatch Resources \(p. 301\)](#)

## Delete Your Kinesis Data Analytics Application

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
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 Apache Flink 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 code, and others cannot be changed.

**This topic contains the following sections:**

- [State Backend \(p. 302\)](#)
- [Checkpointing \(p. 302\)](#)
- [Savepointing \(p. 303\)](#)
- [Heap Sizes \(p. 303\)](#)

## 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 Apache Flink uses a default checkpoint configuration with the following values. Some of these vales can be changed. You must set [CheckpointConfiguration.ConfigurationType](#) to `CUSTOM` for Kinesis Data Analytics to use modified checkpointing values.

Setting	Can be modified?	Default Value
CheckpointingEnabled	Modifiable	True
CheckpointInterval	Modifiable	60000
MinPauseBetweenCheckpoints	Modifiable	5000
Number of Concurrent Checkpoints	Not Modifiable	1
Checkpointing Mode	Not Modifiable	Exactly Once

Setting	Can be modified?	Default Value
Checkpoint Retention Policy	Not Modifiable	On Failure
Checkpoint Timeout	Not Modifiable	60 minutes
Max Checkpoints Retained	Not Modifiable	1
Restart Strategy	Not Modifiable	Fixed Delay, with infinite retries every 10 seconds.
Checkpoint and Savepoint Location	Not Modifiable	We store durable checkpoint and savepoint data to a service-owned S3 bucket.
State Backend Memory Threshold	Not Modifiable	1048576

## 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 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" \(p. 30\)](#).

For more information about JVM heap sizes, see [Configuration](#) in the [Apache Flink documentation](#).

# Configuring Kinesis Data Analytics for Apache Flink 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 \(p. 304\)](#)
- [VPC Application Permissions \(p. 305\)](#)
- [Internet and Service Access for a VPC-Connected Kinesis Data Analytics application \(p. 305\)](#)
- [Kinesis Data Analytics VPC API \(p. 306\)](#)
- [Example: Using a VPC to Access Data in an Amazon MSK Cluster \(p. 308\)](#)

## 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 \(p. 30\)](#).

**Note**

VPC configurations are not supported for SQL applications.



**Note**

The Kinesis Data Analytics service manages the checkpoint and snapshot state for applications that have a VPC configuration.

## 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 \(p. 199\)](#).

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

```
{
  "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 application

By default, when you connect a Kinesis Data Analytics 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 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 Amazon 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 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 \(p. 321\)](#).

### 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:

```
{
  "ApplicationName": "MyApplication",
  "ApplicationDescription": "My-Application-Description",
  "RuntimeEnvironment": "FLINK-1_13",
  "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": {
        "ConfigurationType": "CUSTOM",
        "Parallelism": 2,
        "ParallelismPerKPU": 1,
        "AutoScalingEnabled": true
      }
    },
    "VpcConfigurations": [
      {
        "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
        "SubnetIds": [ "subnet-0123456789abcdef0" ]
      }
    ]
  }
}
```

## 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:

```
{
  "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 `AddApplicationVpcConfiguration` action removes an existing VPC configuration from an application:

```
{
  "ApplicationName": "MyApplication",
```

```
"CurrentApplicationVersionId": 9,  
"VpcConfigurationId": "1.1"  
}
```

## 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 \(p. 135\)](#).

# Troubleshooting Kinesis Data Analytics for Apache Flink

The following can help you troubleshoot problems that you might encounter with Amazon Kinesis Data Analytics for Apache Flink.

## Topics

- [Development Troubleshooting](#) (p. 309)
- [Runtime Troubleshooting](#) (p. 310)

## Development Troubleshooting

### Topics

- [Enabling Flamegraps](#) (p. 309)
- [Issue with EFO connector 1.13.2](#) (p. 309)
- [Compile Error: "Could not resolve dependencies for project"](#) (p. 309)
- [Invalid Choice: "kinesisanalyticsv2"](#) (p. 310)
- [UpdateApplication Action Isn't Reloading Application Code](#) (p. 310)

## Enabling Flamegraps

Due to its experimental nature in Flink, Flamegraphs are currently disabled by default as it may affect application performance. If you want to enable Flamegraphs for your application, create a case to request it to be enabled for you application ARN. For more information, see the [AWS Support Center](#).

## Issue with EFO connector 1.13.2

There is a known issue with the 1.13.2 Kinesis Data Streams EFO connector showing performance degradation if the application suffers from high backpressure. To mitigate, use the Flink 1.13.3 connector. You can use the [1.13-SNAPSHOT](#) on Maven.

## Compile Error: "Could not resolve dependencies for project"

In order to compile the Kinesis Data Analytics for Apache Flink 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 with previous Apache Flink versions"](#) (p. 249).

## 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*.

## UpdateApplication Action Isn't Reloading Application Code

The [UpdateApplication](#) action will not reload application code with the same file name if no S3 object version is specified. To reload application code with the same file name, 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](#).

# Runtime Troubleshooting

This section contains information about diagnosing and fixing runtime issues with your Kinesis Data Analytics application.

### Topics

- [Troubleshooting Tools](#) (p. 310)
- [Application Issues](#) (p. 310)
- [Application is Restarting](#) (p. 313)
- [Throughput is Too Slow](#) (p. 315)
- [Application State Data is Accumulating](#) (p. 316)
- [Checkpointing is timing out](#) (p. 316)

## Troubleshooting Tools

The primary tool for detecting application issues is CloudWatch alarms. Using CloudWatch alarms, you can set thresholds for CloudWatch metrics that indicate error or bottleneck conditions in your application. For information about recommended CloudWatch alarms, see [Using CloudWatch Alarms with Amazon Kinesis Data Analytics for Apache Flink](#) (p. 229).

## Application Issues

This section contains solutions for error conditions that you may encounter with your Kinesis Data Analytics application.

### Topics

- [Application Is Stuck in a Transient Status \(p. 311\)](#)
- [Snapshot Creation Fails \(p. 312\)](#)
- [Cannot Access Resources in a VPC \(p. 312\)](#)
- [Data Is Lost When Writing to an Amazon S3 Bucket \(p. 312\)](#)
- [Application Is in the RUNNING Status But Isn't Processing Data \(p. 312\)](#)
- [Snapshot, Application Update, or Application Stop Error: InvalidApplicationConfigurationException \(p. 313\)](#)
- [java.nio.file.NoSuchFileException: /usr/local/openjdk-8/lib/security/cacerts \(p. 313\)](#)

## Application Is Stuck in a Transient Status

If your application stays in a transient status (STARTING, UPDATING, STOPPING, or AUTOSCALING), you can stop your application by using the [StopApplication](#) action with the `Force` parameter set to `true`. You can't force stop an application in the DELETING status. Alternatively, if the application is in the UPDATING or AUTOSCALING status, you can roll it back to the previous running version. When you roll back an application, it loads state data from the last successful snapshot. If the application has no snapshots, Kinesis Data Analytics rejects the rollback request. For more information about rolling back an application, see [RollbackApplication](#) action.

### Note

Force-stopping your application may lead to data loss or duplication. To prevent data loss or duplicate processing of data during application restarts, we recommend you to take frequent snapshots of your application.

Causes for stuck applications include the following:

- **Application state is too large:** Having an application state that is too large or too persistent can cause the application to become stuck during a checkpoint or snapshot operation. Check your application's `lastCheckpointDuration` and `lastCheckpointSize` metrics for steadily increasing values or abnormally high values.
- **Application code is too large:** Verify that your application JAR file is smaller than 512 MB. JAR files larger than 512 MB are not supported.
- **Application snapshot creation fails:** 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. If automatic snapshot creation fails, see [Snapshot Creation Fails \(p. 312\)](#) following.
- **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 status. 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.
- **Application initialization taking longer:** Kinesis Data Analytics uses an internal timeout of 5 minutes (soft setting) while waiting for a Flink job to start. If your job is failing to start within this timeout, you will see a CloudWatch log as follows:

```
Flink job did not start within a total timeout of 5 minutes for application: %s under
account: %s
```

If you encounter the above error, it means that your operations defined under Flink job's `main` method are taking more than 5 minutes, causing the Flink job creation to time out on the Kinesis Data Analytics end. We suggest you check the Flink **JobManager** logs as well as your application code to see

if this delay in the `main` method is expected. If not, you need to take steps to address the issue so it completes in under 5 minutes.

You can check your application status using either the [ListApplications](#) or the [DescribeApplication](#) actions.

## Snapshot Creation Fails

The Kinesis Data Analytics service can't take a snapshot under the following circumstances:

- The application exceeded the snapshot limit. The limit for snapshots is 1,000. For more information, see [Snapshots \(p. 26\)](#).
- The application doesn't have permissions to access its source or sink.
- The application code isn't 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, set the `SnapshotsEnabled` property of your application's [ApplicationSnapshotConfiguration](#) to `false` and retry the request.

Snapshots can fail if your application's operators are not properly provisioned. For information about tuning operator performance, see [Operator scaling \(p. 241\)](#).

After the application returns to a healthy state, we recommend that you set the application's `SnapshotsEnabled` property to `true`.

## Cannot Access Resources in a VPC

If your application uses a VPC running on 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:

```
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 \(p. 206\)](#).

## Data Is Lost When Writing to an Amazon S3 Bucket

Some data loss might occur when writing output to an Amazon S3 bucket using Apache Flink version 1.6.2. We recommend using the latest supported version of Apache Flink when using Amazon 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 \(p. 146\)](#).

## Application Is in the RUNNING Status But Isn't Processing Data

You can check your application status by using either the [ListApplications](#) or the [DescribeApplication](#) actions. If your application enters the `RUNNING` status but isn't 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 \(p. 208\)](#). The log stream contains messages that you can use to troubleshoot application issues.

## Snapshot, Application Update, or Application Stop Error: `InvalidApplicationConfigurationException`

An error similar to the following might 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 [Application Is Stuck in a Transient Status \(p. 311\)](#).
- Reenable snapshots when the application returns to a healthy state.

## `java.nio.file.NoSuchFileException: /usr/local/openjdk-8/lib/security/cacerts`

The location of the SSL truststore was updated in a previous deployment. Use the following value for the `ssl.truststore.location` parameter instead:

```
/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
```

## Application is Restarting

If your application is not healthy, its Apache Flink job continually fails and restarts. This section describes symptoms and troubleshooting steps for this condition.

### Symptoms

This condition can have the following symptoms:

- The `FullRestarts` metric is not zero. This metric represents the number of times the application's job has restarted since you started the application.
- The `Downtime` metric is not zero. This metric represents the number of milliseconds that the application is in the `FAILING` or `RESTARTING` status.

- The application log contains status changes to `RESTARTING` or `FAILED`. You can query your application log for these status changes using the following CloudWatch Logs Insights query: [Analyze Errors: Application Task-Related Failures](#) (p. 215).

## Causes and Solutions

The following conditions may cause your application to become unstable and repeatedly restart:

- **Operator is Throwing an Exception:** 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). The application restarts from the latest checkpoint to maintain "exactly-once" processing semantics. As a result, `Downtime` is 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.

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. 215).

- **Kinesis Data Streams are not properly provisioned:** If a source or sink for your application is a Kinesis data stream, check the [metrics](#) for the stream for `ReadProvisionedThroughputExceeded` or `WriteProvisionedThroughputExceeded` errors.

If you see these errors, you can increase the available throughput for the Kinesis stream by increasing the stream's number of shards. For more information, see [How do I change the number of open shards in Kinesis Data Streams?](#)

- **Other sources or sinks are not properly provisioned or available:** Verify that your application is correctly provisioning sources and sinks. Check that any sources or sinks used in the application (such as other AWS services, or external sources or destinations) are well provisioned, are not experiencing read or write throttling, or are periodically unavailable.

If you are experiencing throughput-related issues with your dependent services, either increase resources available to those services, or investigate the cause of any errors or unavailability.

- **Operators are not properly provisioned:** If the workload on the threads for one of the operators in your application is not correctly distributed, the operator can become overloaded and the application can crash. For information about tuning operator parallelism, see [Manage operator scaling properly](#) (p. 241).
- **Application fails with `DaemonException`:** This error appears in your application log if you are using a version of Apache Flink prior to 1.11. You may need to upgrade to a later version of Apache Flink so that a KPL version of 0.14 or later is used.
- **Application fails with `TimeoutException`, `FlinkException`, or `RemoteTransportException`:** These errors may appear in your application log if your task managers are crashing. If your application is overloaded, your task managers can experience CPU or memory resource pressure, causing them to fail.

These errors may look like the following:

- `java.util.concurrent.TimeoutException: The heartbeat of JobManager with id xxx timed out`
- `org.apache.flink.util.FlinkException: The assigned slot xxx was removed`
- `org.apache.flink.runtime.io.network.netty.exception.RemoteTransportException: Connection unexpectedly closed by remote task manager`

To troubleshoot this condition, check the following:

- Check your CloudWatch metrics for unusual spikes in CPU or memory usage.
- Check your application for throughput issues. For more information, see [Troubleshooting Performance](#) (p. 238).

- Examine your application log for unhandled exceptions that your application code is raising.
- **Application fails with JaxbAnnotationModule Not Found error:** This error occurs if your application uses Apache Beam, but doesn't have the correct dependencies or dependency versions. Kinesis Data Analytics applications that use Apache Beam must use the following versions of dependencies:

```
<jackson.version>2.10.2</jackson.version>
...
<dependency>
  <groupId>com.fasterxml.jackson.module</groupId>
  <artifactId>jackson-module-jaxb-annotations</artifactId>
  <version>2.10.2</version>
</dependency>
```

If you do not provide the correct version of `jackson-module-jaxb-annotations` as an explicit dependency, your application loads it from the environment dependencies, and since the versions do not match, the application crashes at runtime.

For more information about using Apache Beam with Kinesis Data Analytics, see [Apache Beam \(p. 168\)](#).

## Throughput is Too Slow

If your application is not processing incoming streaming data quickly enough, it will perform poorly and become unstable. This section describes symptoms and troubleshooting steps for this condition.

### Symptoms

This condition can have the following symptoms:

- If the data source for your application is a Kinesis stream, the stream's `MillisBehindLatest` metric continually increases.
- If the data source for your application is an Amazon MSK cluster, the cluster's consumer lag metrics continually increase. For more information, see [Consumer-Lag Monitoring](#) in the [Amazon MSK Developer Guide](#).
- If the data source for your application is a different service or source, check any available consumer lag metrics or data available.

### Causes and Solutions

There can be many causes for slow application throughput. If your application is not keeping up with input, check the following:

- If throughput lag is spiking and then tapering off, check if the application is restarting. Your application will stop processing input while it restarts, causing lag to spike. For information about application failures, see [Application is Restarting \(p. 313\)](#).
- If throughput lag is consistent, check to see if your application is optimized for performance. For information on optimizing your application's performance, see [Troubleshooting Performance \(p. 238\)](#).
- If throughput lag is not spiking but continuously increasing, and your application is optimized for performance, you must increase your application resources. For information on increasing application resources, see [Scaling \(p. 30\)](#).

For troubleshooting steps for slow throughput or consumer lag increasing in the application source, see [Troubleshooting Performance \(p. 238\)](#).

## Application State Data is Accumulating

If your application is not properly disposing of outdated state information, it will continually accumulate and lead to application performance or stability issues. This section describes symptoms and troubleshooting steps for this condition.

### Symptoms

This condition can have the following symptoms:

- The `lastCheckpointDuration` metric is gradually increasing or spiking.
- The `lastCheckpointSize` metric is gradually increasing or spiking.

### Causes and Solutions

The following conditions may cause your application to accumulate state data:

- Your application is retaining state data longer than it is needed.
- Your application uses window queries with too long a duration.
- You did not set TTL for your state data. For more information, see [State Time-To-Live \(TTL\)](#) in the [Apache Flink Documentation](#).
- You are running an application that depends on Apache Beam version 2.25.0 or newer. You can opt out of the new version of the read transform by [extending your BeamApplicationProperties](#) with the key `experiments` and value `use_deprecated_read`. For more information, see the [Apache Beam Documentation](#).

## Checkpointing is timing out

If your application is not optimized or properly provisioned, checkpoints can fail. This section describes symptoms and troubleshooting steps for this condition.

### Symptoms

If checkpoints fail for your application, the `numberOfFailedCheckpoints` will be greater than zero.

Checkpoints can fail due to either direct failures, such as application errors, or due to transient failures, such as running out of application resources. Check your application logs and metrics for the following symptoms:

- Errors in your code.
- Errors accessing your application's dependent services.
- Errors serializing data. If the default serializer can't serialize your application data, the application will fail. For information about using a custom serializer in your application, see [Custom Serializers](#) in the [Apache Flink Documentation](#).
- Out of Memory errors.
- Spikes or steady increases in the following metrics:
  - `heapMemoryUtilization`
  - `oldGenerationGCTime`
  - `oldGenerationGCCount`
  - `lastCheckpointSize`
  - `lastCheckpointDuration`

For more information about monitoring checkpoints, see [Monitoring Checkpointing](#) in the [Apache Flink Documentation](#).

## Causes and Solutions

Your application log error messages show the cause for direct failures. Transient failures can have the following causes:

- Your application has insufficient KPU provisioning. For information about increasing application provisioning, see [Scaling \(p. 30\)](#).
- Your application state size is too large. You can monitor your application state size using the `lastCheckpointSize` metric.
- Your application's state data is unequally distributed between keys. If your application uses the `KeyBy` operator, ensure that your incoming data is being divided equally between keys. If most of the data is being assigned to a single key, this creates a bottleneck that causes failures.
- Your application is experiencing memory or garbage collection backpressure. Monitor your application's `heapMemoryUtilization`, `oldGenerationGCTime`, and `oldGenerationGCCount` for spikes or steadily increasing values.

# Document History for Amazon Kinesis Data Analytics for Apache Flink

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:** October 13, 2021

Change	Description	Date
Support for Apache Flink version 1.13.2	Kinesis Data Analytics now supports applications that use Apache Flink version 1.13.2. Create Kinesis Data Analytics applications using the Apache Flink Table API. For more information, see <a href="#">Creating Applications (p. 3)</a> .	October 13, 2021
Support for Python	Kinesis Data Analytics now supports applications that use Python with the Apache Flink Table API & SQL. For more information, see <a href="#">Using Python (p. 17)</a> .	March 25, 2021
Support for Apache Flink 1.11.1	Kinesis Data Analytics now supports applications that use Apache Flink 1.11.1. Create Kinesis Data Analytics applications using the Apache Flink Table API. For more information, see <a href="#">Creating Applications (p. 3)</a> .	November 19, 2020
Apache Flink Dashboard	Use the Apache Flink Dashboard to monitor application health and performance. For more information, see <a href="#">Apache Flink Dashboard (p. 34)</a> .	November 19, 2020
EFO Consumer	Create applications that use an Enhanced Fan-Out (EFO) consumer to read from a Kinesis Data Stream. For more information, see <a href="#">EFO Consumer (p. 139)</a> .	October 6, 2020

Change	Description	Date
Apache Beam	Create applications that use Apache Beam to process streaming data. For more information, see <a href="#">Apache Beam (p. 168)</a> .	September 15, 2020
Performance	How to troubleshoot application performance issues, and how to create a performant application. For more information, see <a href="#">Performance (p. 238)</a> .	July 21, 2020
Custom Keystore	How to access an Amazon MSK cluster that uses a custom keystore for encryption in transit. For more information, see <a href="#">Custom Truststore (p. 163)</a> .	June 10, 2020
CloudWatch Alarms	Recommendations for creating CloudWatch alarms with Kinesis Data Analytics for Apache Flink. For more information, see <a href="#">Alarms (p. 229)</a> .	June 5, 2020
New CloudWatch Metrics	Kinesis Data Analytics now emits 22 metrics to Amazon CloudWatch Metrics. For more information, see <a href="#">Metrics and Dimensions (p. 215)</a> .	May 12, 2020
Custom CloudWatch Metrics	Define application-specific metrics and emit them to Amazon CloudWatch Metrics. For more information, see <a href="#">Custom Metrics (p. 226)</a> .	May 12, 2020
Example: Read From a Kinesis Stream in a Different Account	Learn how to access a Kinesis stream in a different AWS account in your Kinesis Data Analytics application. For more information, see <a href="#">Cross-Account (p. 157)</a> .	March 30, 2020
Support for Apache Flink 1.8.2	Kinesis Data Analytics now supports applications that use Apache Flink 1.8.2. Use the Flink StreamingFileSink connector to write output directly to S3. For more information, see <a href="#">Creating Applications (p. 3)</a> .	December 17, 2019
Kinesis Data Analytics VPC	Configure a Kinesis Data Analytics application to connect to a virtual private cloud. For more information, see <a href="#">Using an Amazon VPC (p. 304)</a> .	November 25, 2019

Change	Description	Date
Kinesis Data Analytics Best Practices	Best practices for creating and administering Kinesis Data Analytics applications. For more information, see <a href="#">Best Practices (p. 246)</a> .	October 14, 2019
Analyze Kinesis Data Analytics Application Logs	Use CloudWatch Logs Insights to monitor your Kinesis Data Analytics application. For more information, see <a href="#">Analyzing Logs (p. 213)</a> .	June 26, 2019
Kinesis Data Analytics Application Runtime Properties	Work with Runtime Properties in Kinesis Data Analytics for Apache Flink. For more information, see <a href="#">Runtime Properties (p. 21)</a> .	June 24, 2019
Tagging Kinesis Data Analytics Applications	Use application tagging to determine per-application costs, control access, or for user-defined purposes. For more information, see <a href="#">Using Tagging (p. 33)</a> .	May 8, 2019
Kinesis Data Analytics Example Applications	Example applications for Amazon Kinesis Data Analytics demonstrating window operators and writing output to CloudWatch Logs. For more information, see <a href="#">Examples (p. 113)</a> .	May 1, 2019
Logging Kinesis Data Analytics API Calls with AWS CloudTrail	Amazon Kinesis Data Analytics is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Kinesis Data Analytics. For more information, see <a href="#">Using AWS CloudTrail (p. 235)</a> .	March 22, 2019
Create an Application (Kinesis Data Firehose Sink)	Exercise to create a Kinesis Data Analytics for Apache Flink 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 <a href="#">Kinesis Data Firehose Sink (p. 146)</a> .	December 13, 2018
Public release	This is the initial release of the <i>Kinesis Data Analytics Developer Guide for Java Applications</i> .	November 27, 2018



# 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 kinesisanalyticstv2 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. 321)
- [AddApplicationInput](#) (p. 322)
- [AddApplicationInputProcessingConfiguration](#) (p. 322)
- [AddApplicationOutput](#) (p. 323)
- [AddApplicationReferenceDataSource](#) (p. 323)
- [AddApplicationVpcConfiguration](#) (p. 324)
- [CreateApplication](#) (p. 324)
- [CreateApplicationSnapshot](#) (p. 325)
- [DeleteApplication](#) (p. 325)
- [DeleteApplicationCloudWatchLoggingOption](#) (p. 325)
- [DeleteApplicationInputProcessingConfiguration](#) (p. 326)
- [DeleteApplicationOutput](#) (p. 326)
- [DeleteApplicationReferenceDataSource](#) (p. 326)
- [DeleteApplicationSnapshot](#) (p. 326)
- [DeleteApplicationVpcConfiguration](#) (p. 326)
- [DescribeApplication](#) (p. 327)
- [DescribeApplicationSnapshot](#) (p. 327)
- [DiscoverInputSchema](#) (p. 327)
- [ListApplications](#) (p. 328)
- [ListApplicationSnapshots](#) (p. 328)
- [StartApplication](#) (p. 328)
- [StopApplication](#) (p. 328)
- [UpdateApplication](#) (p. 329)

## AddApplicationCloudWatchLoggingOption

The following example request code for the [AddApplicationCloudWatchLoggingOption](#) action adds an Amazon CloudWatch logging option to a Kinesis Data Analytics application:

```
{
  "ApplicationName": "MyApplication",
  "CloudWatchLoggingOption": {
    "LogStreamARN": "arn:aws:logs:us-east-1:123456789123:log-group:my-log-group:log-
stream:My-LogStream"
  },
  "CurrentApplicationVersionId": 2
}
```

## AddApplicationInput

The following example request code for the [AddApplicationInput](#) action adds an application input to a Kinesis Data Analytics application:

```
{
  "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:

```
{
```

```
"ApplicationName": "MyApplication",
"CurrentApplicationVersionId": 2,
"InputId": "2.1",
"InputProcessingConfiguration": {
  "InputLambdaProcessor": {
    "ResourceARN": "arn:aws:lambda:us-east-1:012345678901:function:MyLambdaFunction"
  }
}
}
```

## 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:

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 2,
  "Output": {
    "DestinationSchema": {
      "RecordFormatType": "JSON"
    },
    "KinesisStreamsOutput": {
      "ResourceARN": "arn:aws:kinesis:us-east-1:012345678901:stream/ExampleOutputStream"
    },
    "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:

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

```
        },
        "RecordFormatType": "CSV"
    },
    "S3ReferenceDataSource": {
        "BucketARN": "arn:aws:s3:::MyS3Bucket",
        "FileKey": "TickerReference.csv"
    },
    "TableName": "string"
}
}
```

## 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_13",
  "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
  "CloudWatchLoggingOptions": [
    {
      "LogStreamARN": "arn:aws:logs:us-east-1:123456789123:log-group:my-log-group:log-stream:My-LogStream"
    }
  ],
  "ApplicationConfiguration": {
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-east-1",
            "flink.stream.initpos": "LATEST"
          }
        },
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-east-1"
          }
        }
      ]
    }
  }
}
```

```
    ]
  },
  "ApplicationCodeConfiguration":{
    "CodeContent":{
      "S3ContentLocation":{
        "BucketARN":"arn:aws:s3:::mybucket",
        "FileKey":"myflink.jar",
        "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
      }
    },
    "CodeContentType":"ZIPFILE"
  },
  "FlinkApplicationConfiguration":{
    "ParallelismConfiguration":{
      "ConfigurationType":"CUSTOM",
      "Parallelism":2,
      "ParallelismPerKPU":1,
      "AutoScalingEnabled":true
    }
  }
}
```

## CreateApplicationSnapshot

The following example request code for the [CreateApplicationSnapshot](#) action creates a snapshot of application state:

```
{
  "ApplicationName": "MyApplication",
  "SnapshotName": "MySnapshot"
}
```

## DeleteApplication

The following example request code for the [DeleteApplication](#) action deletes a Kinesis Data Analytics application:

```
{"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:

```
{
  "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:

```
{
  "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:

```
{
  "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:

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 5,
  "ReferenceId": "5.1"
}
```

## DeleteApplicationSnapshot

The following example request code for the [DeleteApplicationSnapshot](#) action deletes a snapshot of application state:

```
{
  "ApplicationName": "MyApplication",
  "SnapshotCreationTimestamp": 12345678912,
  "SnapshotName": "MySnapshot"
}
```

## DeleteApplicationVpcConfiguration

The following example request code for the [DeleteApplicationVpcConfiguration](#) action removes an existing VPC configuration from an application:

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 9,
  "VpcConfigurationId": "1.1"
}
```

## DescribeApplication

The following example request code for the [DescribeApplication](#) action returns details about a Kinesis Data Analytics application:

```
{"ApplicationName": "MyApplication"}
```

## DescribeApplicationSnapshot

The following example request code for the [DescribeApplicationSnapshot](#) action returns details about a snapshot of application state:

```
{
  "ApplicationName": "MyApplication",
  "SnapshotName": "MySnapshot"
}
```

## DiscoverInputSchema

The following example request code for the [DiscoverInputSchema](#) action generates a schema from a streaming source:

```
{
  "InputProcessingConfiguration": {
    "InputLambdaProcessor": {
      "ResourceARN": "arn:aws:lambda:us-east-1:012345678901:function:MyLambdaFunction"
    }
  },
  "InputStartingPositionConfiguration": {
    "InputStartingPosition": "NOW"
  },
  "ResourceARN": "arn:aws:kinesis:us-east-1:012345678901:stream/ExampleInputStream",
  "S3Configuration": {
    "BucketARN": "string",
    "FileKey": "string"
  },
  "ServiceExecutionRole": "string"
}
```

The following example request code for the [DiscoverInputSchema](#) action generates a schema from a reference source:

```
{
```

```
"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:

```
{
  "ExclusiveStartApplicationName": "MyApplication",
  "Limit": 50
}
```

## ListApplicationSnapshots

The following example request code for the [ListApplicationSnapshots](#) action returns a list of snapshots of application state:

```
{ "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):

```
{
  "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:

```
{ "ApplicationName": "MyApplication" }
```



## UpdateApplication

The following example request code for the [UpdateApplication](#) action updates a Kinesis Data Analytics application to change the location of the application code:

```
{ "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 Apache Flink API Reference

For information about the APIs that Kinesis Data Analytics for Apache Flink provides, see [Kinesis Data Analytics API Reference](#).