



Managed Service for Apache Flink Developer Guide

Managed Service for Apache Flink



Managed Service for Apache Flink: Managed Service for Apache Flink Developer Guide

Copyright © 2024 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

.....	xvi
What is Managed Service for Apache Flink?	1
Choosing Managed Service for Apache Flink or Managed Service for Apache Flink Studio	1
Choosing which Apache Flink APIs to use in Managed Service for Apache Flink	3
Choosing a Flink API	3
Getting started with streaming data applications	5
How it works	6
Programming your Apache Flink application	6
DataStream API	6
Table API	7
Creating your Managed Service for Apache Flink application	7
Creating applications	8
Building your Managed Service for Apache Flink application code	8
Creating your Managed Service for Apache Flink application	9
Starting your Managed Service for Apache Flink application	10
Verifying your Managed Service for Apache Flink application	10
Enabling system rollbacks for your Managed Service for Apache Flink application	11
Running applications	13
Application and job status	14
Batch workloads	15
Application resources	15
Managed Service for Apache Flink application resources	16
Apache Flink application resources	16
DataStream API	17
DataStream API connectors	18
DataStream API operators	28
DataStream API timestamps	29
Table API	30
Table API connectors	30
Table API time attributes	32
Using Python	32
Programming an application	33
Creating an application	36
Monitoring	37

Runtime properties	38
Working with runtime properties in the console	39
Working with runtime properties in the CLI	39
Accessing runtime properties in a Managed Service for Apache Flink application	42
Apache Flink connectors	43
Fault tolerance	45
Configuring Checkpointing	46
Checkpointing API examples	47
Snapshots	49
Automatic snapshot creation	50
Restoring from a snapshot that contains incompatible state data	51
Snapshot API examples	52
In-place version upgrades	55
Upgrading applications using in-place version upgrades for Apache Flink	55
Upgrading your application to a new Apache Flink version	56
Rollback	62
General best practices and recommendations	63
Precautions and known issues	63
Scaling	65
Configuring application parallelism and ParallelismPerKPU	65
Allocating Kinesis Processing Units	66
Updating your application's parallelism	67
Automatic scaling	68
Tagging	70
Adding tags when an application is created	71
Adding or u tags for an existing application	72
Listing tags for an application	72
Removing tags from an application	72
Using CloudFormation with Managed Service for Apache Flink	73
Before you begin	73
Writing a Lambda function	73
Creating a Lambda role	75
Invoking the Lambda function	76
Invoking the Lambda function	76
Apache Flink Dashboard	82
Accessing your application's Apache Flink Dashboard	82

Release versions	84
Amazon Managed Service for Apache Flink 1.19	85
Supported features	86
Changes in Amazon Managed Service for Apache Flink 1.19.1	88
Components	89
Known issues	89
Amazon Managed Service for Apache Flink 1.18	89
Changes in Amazon Managed Service for Apache Flink with Apache Flink 1.15	91
Components	92
Bug fixes	92
Known issues	92
Amazon Managed Service for Apache Flink 1.15	93
Changes in Amazon Managed Service for Apache Flink with Apache Flink 1.15	91
Components	92
Earlier versions	96
Using the Apache Flink Kinesis Streams connector with previous Apache Flink versions	96
Building applications with Apache Flink 1.8.2	98
Building applications with Apache Flink 1.6.2	98
Upgrading applications	99
Available connectors in Apache Flink 1.6.2 and 1.8.2	100
Getting Started: Flink 1.13.2	100
Getting Started: Flink 1.11.1	126
Getting started: Flink 1.8.2 - deprecating	152
Getting started: Flink 1.6.2 - deprecating	177
Legacy examples	202
Studio notebooks	372
Studio notebook versions	373
Creating a Studio notebook	374
Interactive analysis of streaming data	375
Flink interpreters	375
Apache Flink table environment variables	376
Deploying as an application with durable state	377
Scala/Python criteria	379
SQL criteria	379
IAM permissions	379
Connectors and dependencies	380

Default connectors	380
Adding dependencies and custom connectors	382
User-defined functions	383
Considerations with user-defined functions	384
Enabling checkpointing	385
Setting the checkpointing interval	385
Setting the checkpointing type	386
Upgrading Studio Runtime	386
Upgrading your notebook to a new Studio Runtime	386
Working with AWS Glue	391
Table properties	391
Examples and tutorials	393
Creating a Studio notebook tutorial	393
Deploying as an application with durable state tutorial	413
Examples	416
Troubleshooting	428
Stopping a stuck application	428
Deploying as an application with durable state in a VPC with no internet access	428
Deploy-as-app size and build time reduction	429
Canceling jobs	431
Restarting the Apache Flink interpreter	432
Appendix: Creating custom IAM policies	433
AWS Glue	433
CloudWatch Logs	434
Kinesis streams	435
Amazon MSK clusters	437
Getting started (DataStream API)	438
Application Components	153
Prerequisites	439
Step 1: Set Up an Account	439
Sign up for an AWS account	101
Create a user with administrative access	102
Grant programmatic access	441
Next Step	443
Step 2: Set Up the AWS CLI	443
Next step	445

Step 3: Create an application	445
Create two Amazon Kinesis data streams	446
Write sample records to the input stream	446
Download and examine the Apache Flink streaming Java code	447
Compile the application code	448
Upload the Apache Flink streaming Java code	449
Create and run the Managed Service for Apache Flink application	450
Next step	461
Step 4: Clean Up	462
Delete your Managed Service for Apache Flink application	462
Delete your Kinesis data streams	462
Delete your Amazon S3 object and bucket	462
Delete your IAM resources	463
Delete your CloudWatch resources	463
Next Step	463
Step 5: Use resources to complete next steps	463
Getting started (Table API)	465
Application Components	465
Prerequisites	466
Create an Application	466
Create dependent resources	466
Write samplerRecords to the input stream	468
Download and examine the Apache Flink streaming Java code	469
Compile the application code	471
Upload the Apache Flink streaming Java code	472
Create and run the Managed Service for Apache Flink application	472
Next step	477
Clean Up	477
Delete your Managed Service for Apache Flink application	477
Delete your Amazon MSK cluster	477
Delete your VPC	478
Delete your Amazon S3 objects and bucket	478
Delete your IAM resources	478
Delete your CloudWatch resources	479
Next step	479
Next steps	479

Getting started (Python)	480
Getting started with Pyflink - The Python Interpreter for Apache Amazon Web Services	480
Application Components	480
Prerequisites	481
Create an Application	481
Create dependent resources	482
Write sample records to the input stream	483
Create and examine the Apache Flink streaming Python code	484
Adding third-party dependencies to Python apps	486
Upload the Apache Flink streaming Python code	487
Create and run the Managed Service for Apache Flink application	489
Next step	493
Clean Up	494
Delete your Managed Service for Apache Flink application	494
Delete your Kinesis data streams	494
Delete your Amazon S3 objects and bucket	494
Delete your IAM resources	495
Delete your CloudWatch resources	495
Getting started (Scala)	496
Create dependent resources	496
Write sample records to the input stream	497
Download and examine the application code	499
Compile and upload the application code	500
Create and run the application (console)	501
Create the Application	501
Configure the application	502
Edit the IAM policy	504
Run the application	505
Stop the application	506
Create and run the application (CLI)	506
Create a permissions policy	506
Create an IAM policy	508
Create the application	509
Start the application	510
Stop the application	367
Add a CloudWatch logging option	368

Update environment properties	368
Update the application code	369
Clean Up	514
Delete your Managed Service for Apache Flink application	514
Delete your Kinesis data streams	514
Delete your Amazon S3 object and bucket	515
Delete your IAM resources	515
Delete your CloudWatch resources	515
Using Apache Beam	516
Using Apache Beam with Managed Service for Apache Flink	516
Beam capabilities	517
Creating an application using Apache Beam	517
Create dependent resources	517
Write sample records to the input stream	518
Download and examine the application code	519
Compile the application code	520
Upload the Apache Flink streaming Java code	521
Create and run the Managed Service for Apache Flink application	521
Clean Up	525
Next steps	526
Training workshops, labs, and solution implementations	527
Managed Service for Apache Flink workshop	527
Develop Apache Flink applications locally before deploying to Managed Service for Apache Flink	527
Event detection with Managed Service for Apache Flink Studio	528
AWS Streaming Data Solution	528
Clickstream Lab	528
Custom Scaling	529
CloudWatch Dashboard	529
Amazon MSK	529
Explore more Managed Service for Apache Flink solutions on GitHub	529
Utilities	531
Snapshot manager	531
Benchmarking	531
Examples	532
Java examples	532

Python examples	534
.....	534
.....	534
Scala examples	536
Security	537
Data protection	538
Data Encryption	538
Identity and Access Management	539
Audience	539
Authenticating with identities	540
Managing access using policies	543
How Amazon Managed Service for Apache Flink works with IAM	546
Identity-based policy examples	553
Troubleshooting	556
Cross-service confused deputy prevention	557
Monitoring	559
Compliance validation	559
FedRAMP	560
Resilience	561
Disaster recovery	561
Versioning	561
Infrastructure security	562
Security best practices	562
Implement least privilege access	562
Use IAM roles to access other Amazon services	563
Implement server-side encryption in dependent resources	563
Use CloudTrail to monitor API calls	563
Logging and monitoring	564
Logging	565
Querying Logs with CloudWatch Logs Insights	565
Monitoring	565
Setting up logging	567
Setting up CloudWatch logging using the console	567
Setting up CloudWatch logging using the CLI	568
Application monitoring levels	573
Logging best practices	574

Logging troubleshooting	574
Next step	575
Analyzing logs	575
Run a sample query	575
Example queries	576
Metrics and dimensions in Managed Service for Apache Flink	579
Application metrics	580
Kinesis Data Streams connector metrics	608
Amazon MSK connector metrics	609
Apache Zeppelin metrics	611
Viewing CloudWatch metrics	612
Metrics	613
Custom metrics	614
Alarms	618
Writing custom messages	630
Write to CloudWatch logs using Log4J	630
Write to CloudWatch logs using SLF4J	631
Using AWS CloudTrail	632
Managed Service for Apache Flink information in CloudTrail	633
Understanding Managed Service for Apache Flink log file entries	634
Performance	636
Troubleshooting performance	636
The data path	636
Performance troubleshooting solutions	637
Performance best practices	639
Manage scaling properly	639
Monitor external dependency resource usage	641
Run your Apache Flink application locally	641
Monitoring performance	642
Performance monitoring using CloudWatch metrics	642
Performance monitoring using CloudWatch logs and alarms	642
Quota	643
Maintenance	645
Set a UUID for all operators	647
Identify maintenance instances	647
Production readiness	648

Load testing applications	648
Max parallelism	648
Set a UUID for all operators	649
Best practices	650
Fault tolerance: checkpoints and savepoints	650
Unsupported connector versions	651
Performance and parallelism	651
Setting per-operator parallelism	652
Logging	652
Coding	653
Managing credentials	653
Reading from sources with few shards/partitions	654
Studio notebook refresh interval	654
Studio notebook optimum performance	654
How watermark strategies and idle shards affect time windows	655
Summary	656
Example	657
Set a UUID for all operators	666
Add ServiceResourceTransformer to the Maven shade plugin	666
Apache Flink stateful functions	668
Apache Flink application template	668
Location of the module configuration	669
Flink settings	670
Apache Flink configuration	670
State backend	670
Checkpointing	671
Savepointing	672
Heap sizes	672
Buffer debloating	673
Modifiable Flink configuration properties	673
Restart strategy	673
Checkpoints and state backends	673
Checkpointing	674
RocksDB native metrics	674
Advanced state backends options	675
Full TaskManager options	675

Memory configuration	676
RPC / Akka	676
Client	677
Advanced cluster options	677
Filesystem configurations	677
Advanced fault tolerance options	677
Memory configuration	676
Metrics	677
Advanced options for the REST endpoint and client	677
Advanced SSL security options	678
Advanced scheduling options	678
Advanced options for Flink web UI	678
Viewing configured Flink properties	678
Using an Amazon VPC	679
Amazon VPC concepts	679
VPC application permissions	680
Permissions policy for accessing an Amazon VPC	680
Internet and service access	681
Related information	683
VPC API	683
Create application	683
AddApplicationVpcConfiguration	684
DeleteApplicationVpcConfiguration	684
Update application	685
Example: Using a VPC	685
Troubleshooting	686
Development troubleshooting	686
FlinkRuntimeException: "Not allowed configuration change(s) were detected"	686
System rollback best practices	687
Hudi configuration best practices	689
Apache Flink Flame Graphs	689
Credential provider issue with EFO connector 1.15.2	689
Applications with unsupported Kinesis connectors	689
Compile error: "Could not resolve dependencies for project"	692
Invalid choice: "kinesisanalyticsv2"	693
UpdateApplication action isn't reloading application code	693

S3 StreamingFileSink FileNotFoundExceptions	693
FlinkKafkaConsumer issue with stop with savepoint	695
Flink 1.15 Async Sink Deadlock	695
Amazon Kinesis data streams source processing out of order during re-sharding	705
Runtime troubleshooting	706
Troubleshooting tools	706
Application issues	706
Application is restarting	711
Throughput is Too Slow	714
Unbounded state growth	715
I/O bound operators	716
Upstream or source throttling from a Kinesis data stream	716
Checkpoints	717
Checkpointing Timing Out	723
Checkpoint Failure (Beam)	725
Backpressure	727
Data skew	728
State skew	728
Integrating with resources in different regions	729
Document history	730
API example code	736
AddApplicationCloudWatchLoggingOption	737
AddApplicationInput	737
AddApplicationInputProcessingConfiguration	738
AddApplicationOutput	739
AddApplicationReferenceDataSource	739
AddApplicationVpcConfiguration	740
CreateApplication	740
CreateApplicationSnapshot	742
DeleteApplication	742
DeleteApplicationCloudWatchLoggingOption	742
DeleteApplicationInputProcessingConfiguration	742
DeleteApplicationOutput	743
DeleteApplicationReferenceDataSource	743
DeleteApplicationSnapshot	743
DeleteApplicationVpcConfiguration	744

DescribeApplication	744
DescribeApplicationSnapshot	744
DiscoverInputSchema	744
ListApplications	745
ListApplicationSnapshots	745
StartApplication	746
StopApplication	746
UpdateApplication	746
API Reference	748
.....	749

Amazon Managed Service for Apache Flink was previously known as Amazon Kinesis Data Analytics for Apache Flink.

What is Amazon Managed Service for Apache Flink?

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

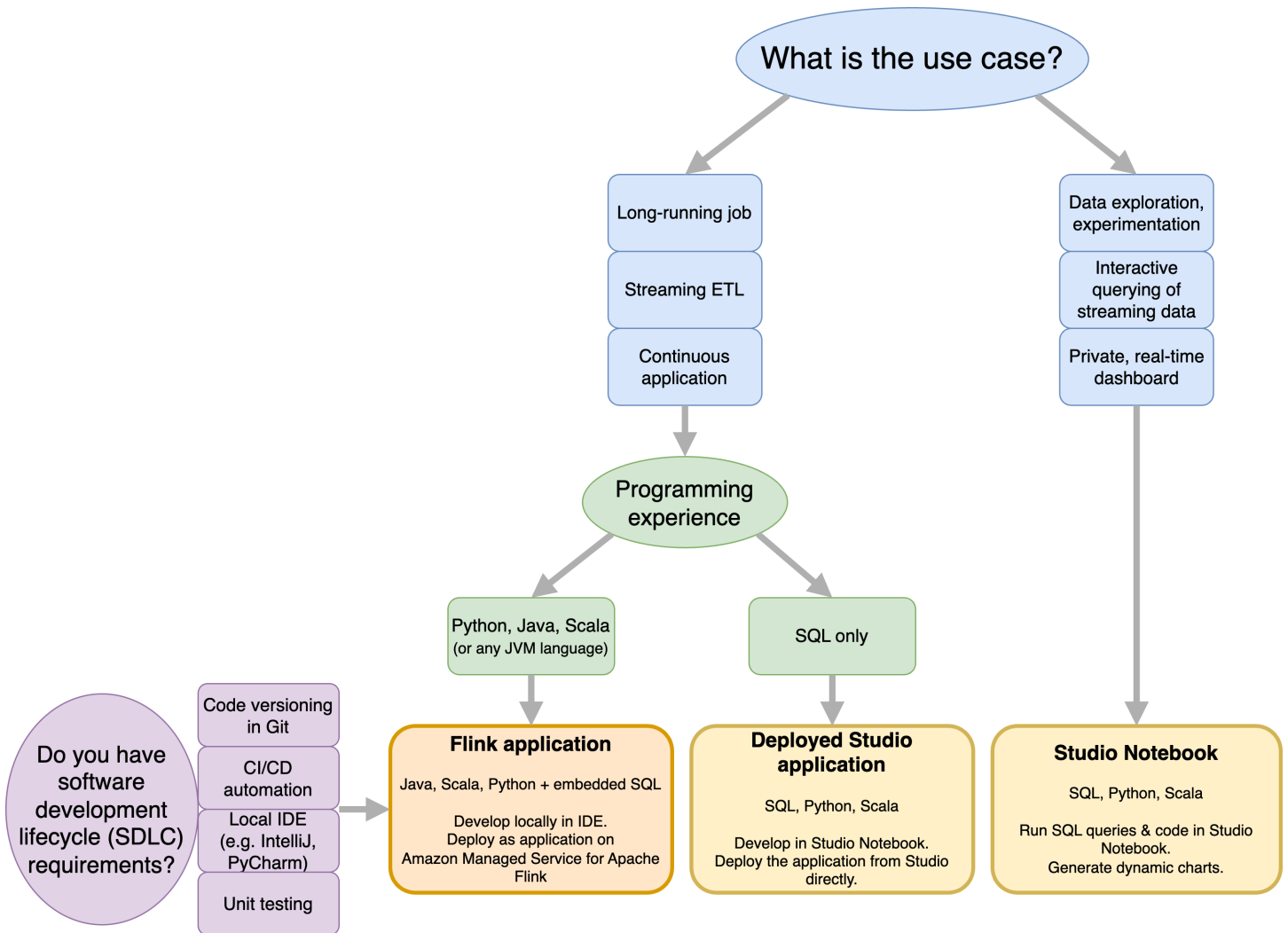
You can build applications with the language of your choice in Managed Service for Apache Flink using open-source libraries based on [Apache Flink](#). Apache Flink is a popular framework and engine for processing data streams.

Managed Service for Apache Flink provides the underlying infrastructure for your Apache Flink applications. It handles core capabilities like provisioning compute resources, AZ failover resilience, 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.

Choosing Managed Service for Apache Flink or Managed Service for Apache Flink Studio

You have two options for running your Flink jobs with Amazon Managed Service for Apache Flink. With [Managed Service for Apache Flink](#), you build Flink applications in Java, Scala, or Python (and embedded SQL) using an IDE of your choice and the Apache Flink Datastream or Table APIs. With [Managed Service for Apache Flink Studio](#), you can interactively query data streams in real time and easily build and run stream processing applications using standard SQL, Python, and Scala.

You can select which method that best suits your use case. If you are unsure, this section will offer high level guidance to help you.



Before deciding on whether to use Amazon Managed Service for Apache Flink or Amazon Managed Service for Apache Flink Studio you should consider your use case.

If you plan to operate a long running application that will undertake workloads such as Streaming ETL or Continuous Applications, you should consider using [Managed Service for Apache Flink](#). This is because you are able to create your Flink application using the Flink APIs directly in the IDE of your choice. Developing locally with your IDE also ensures you can leverage software development lifecycle (SDLC) common processes and tooling such as code versioning in Git, CI/CD automation, or unit testing.

If you are interested in ad-hoc data exploration, want to query streaming data interactively, or create private real-time dashboards, [Managed Service for Apache Flink Studio](#) will help you meet these goals in just a few clicks. Users familiar with SQL can consider deploying a long-running application from Studio directly.

Note

You can promote your Studio notebook to a long-running application. However, if you want to integrate with your SDLC tools such as code versioning on Git and CI/CD automation, or techniques such as unit-testing, we recommend Managed Service for Apache Flink using the IDE of your choice.

Choosing which Apache Flink APIs to use in Managed Service for Apache Flink

You can build applications using Java, Python, and Scala in Managed Service for Apache Flink using Apache Flink APIs in an IDE of your choice. You can find guidance on how to build applications using the Flink Datastream and Table API in the [documentation](#). You can select the language you create your Flink application in and the APIs you use to best meet the needs of your application and operations. If you are unsure, this section provides high level guidance to help you.

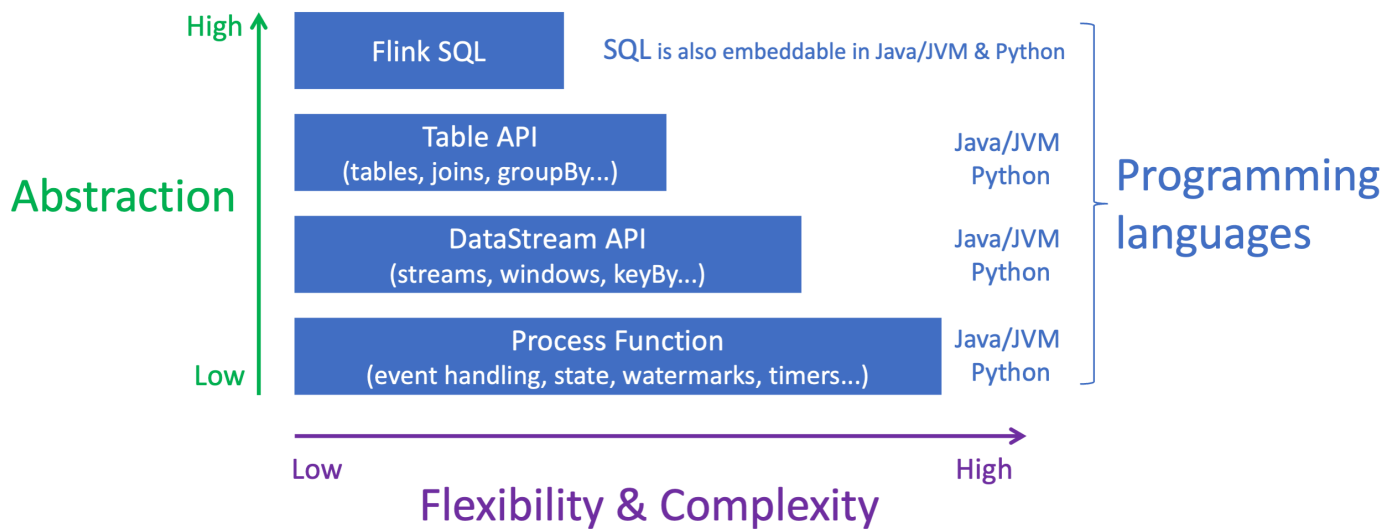
Choosing a Flink API

The Apache Flink APIs have differing levels of abstraction that may effect how you decide to build your application. They are expressive and flexible and can be used together to build your application. You do not have to use only one Flink API. You can learn more about the Flink APIs in the [Apache Flink documentation](#).

Flink offers four levels of API abstraction: Flink SQL, Table API, DataStream API, and Process Function, which is used in conjunction with the DataStream API. These are all supported in Amazon Managed Service for Apache Flink. It is advisable to start with a higher level of abstraction where possible, however some Flink features are only available with the [Datastream API](#) where you can create your application in Java, Python, or Scala. You should consider using the Datastream API if:

- You require fine-grained control over state
- You want to leverage the ability to call an external database or endpoint asynchronously (for example for inference)
- You want to use custom timers (for example to implement custom windowing or late event handling)
- You want to be able to modify the flow of your application without resetting the state

Apache Flink APIs



Note

Choosing a language with the DataStream API:

- SQL can be embedded in any Flink application, regardless the programming language chosen.
- If you are if planning to use the DataStream API, not all connectors are supported in Python.
- If you need low-latency/high-throughput you should consider Java/Scala regardless the API.
- If you plan to use Async IO in the Process Functions API you will need to use Java.

The choice of the API can also impact your ability to evolve the application logic without having to reset the state. This depends on a specific feature, the ability to set UID on operators, that is only available in the DataStream API for both Java and Python. For more information, see [Set UUIIDs For All Operators](#) in the Apache Flink Documentation.

Getting started with streaming data applications

You can start by creating a Managed Service for Apache Flink 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 Managed Service for Apache Flink to send the results.

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

- [Managed Service for Apache Flink: How it works](#)
- [Getting started with Amazon Managed Service for Apache Flink \(DataStream API\)](#)

Alternatively, you can start by creating a Managed Service for Apache Flink Studio notebook that 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. To get started, we recommend that you read the following sections:

- [Using a Studio notebook with Managed Service for Apache Flink](#)
- [Creating a Studio notebook](#)

Managed Service for Apache Flink: How it works

Managed Service 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](#).

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](#).

Creating your Managed Service for Apache Flink application

Managed Service for Apache Flink is an AWS service that creates an environment for hosting your Apache Flink application and provides it with the following settings::

- **[Runtime properties](#)**: Parameters that you can provide to your application. You can change these parameters without recompiling your application code.
- **[Fault tolerance](#)**: How your application recovers from interrupts and restarts.
- **[Logging and monitoring](#)**: How your application logs events to CloudWatch Logs.
- **[Scaling](#)**: How your application provisions computing resources.

You create your Managed Service for Apache Flink application using either the console or the AWS CLI. To get started creating a Managed Service for Apache Flink application, see [Getting started \(DataStream API\)](#).

Creating a Managed Service for Apache Flink application

This topic contains information about creating a Managed Service for Apache Flink.

This topic contains the following sections:

- [Building your Managed Service for Apache Flink application code](#)
- [Creating your Managed Service for Apache Flink application](#)
- [Starting your Managed Service for Apache Flink application](#)
- [Verifying your Managed Service for Apache Flink application](#)
- [Enabling system rollbacks for your Managed Service for Apache Flink application](#)

Building your Managed Service for Apache Flink application code

This section describes the components that you use to build the application code for your Managed Service for Apache Flink application.

We recommend that you use the latest supported version of Apache Flink for your application code. For information about upgrading Managed Service for Apache Flink applications, see [???](#).

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

Managed Service for Apache Flink supports JAR files up to 512 MB in size. If you use a JAR file larger than this, your application will fail to start.

Applications can now use the Java API from any Scala version. You must bundle the Scala standard library of your choice into your Scala applications.

For information about creating a Managed Service for Apache Flink application that uses **Apache Beam**, see [Using Apache Beam](#).

Specifying your application's Apache Flink version

When using Managed Service for Apache 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. For example, if you are using Apache Flink 1.19.1, provide the following:

```
mvn package -Dflink.version=1.19.1
```

For building applications with earlier versions of Apache Flink, see [Earlier versions](#).

Creating your Managed Service for Apache Flink application

After you have built your application code, you do the following to create your Managed Service for Apache Flink 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"](#) in the [Getting started \(DataStream API\)](#) tutorial.
- **Create your Managed Service for Apache Flink application:** Use one of the following methods to create your Managed Service for Apache Flink application:
 - **Create your Managed Service for Apache Flink 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 **Managed Service for Apache Flink - 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 \(Console\)”](#) in the [Getting started \(DataStream API\)](#) tutorial.

- **Create your Managed Service for Apache Flink 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”](#) in the [Getting started \(DataStream API\)](#) tutorial.

Note

You can change the `RuntimeEnvironment` of an existing application. To learn how, see [In-place version upgrades for Apache Flink](#).

Starting your Managed Service for Apache Flink application

After you have built your application code, uploaded it to S3, and created your Managed Service for Apache Flink application, you then start your application. Starting a Managed Service for Apache Flink application typically takes several minutes.

Use one of the following methods to start your application:

- **Start your Managed Service for Apache Flink application using the AWS console:** You can run your application by choosing **Run** on your application's page in the AWS console.
- **Start your Managed Service for Apache Flink application using the AWS API:** You can run your application using the [StartApplication](#) action.

Verifying your Managed Service for Apache Flink 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 Managed Service for Apache Flink application, see [Logging and monitoring](#).
- **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, Firehose 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.

Enabling system rollbacks for your Managed Service for Apache Flink application

With system-rollback capability, you can achieve higher availability of your running Apache Flink application on Amazon Managed Service for Apache Flink. Opting into this configuration enables the service to automatically revert the application to the previously running version when an action such as `UpdateApplication` or `autoscaling` runs into code or configurations bugs.

Note

To use the system rollback feature, you must opt in by updating your application. Existing applications will not automatically use system rollback by default.

How it works

When you initiate an application operation, such as an update or scaling action, the Amazon Managed Service for Apache Flink first attempts to run that operation. If it detects issues that prevent the operation from succeeding, such as code bugs or insufficient permissions, the service automatically initiates a `RollbackApplication` operation.

The rollback attempts to restore the application to the previous version that ran successfully, along with the associated application state. If the rollback is successful, your application continues processing data with minimal downtime using the previous version. If the automatic rollback also fails, Amazon Managed Service for Apache Flink transitions the application to the `READY` status, so that you can take further actions, including fixing the error and retrying the operation.

You must opt in to use automatic system rollbacks. You can enable it using the console or API for all operations on your application from this point forward.

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

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationSystemRollbackConfigurationUpdate": {
      "RollbackEnabledUpdate": "true"
    }
  }
}
```

Common scenarios

The following scenarios illustrate where automatic system rollbacks are beneficial:

- **Application updates:** If you update your application with new code that has bugs when initializing the Flink job through the main method, the automatic rollback allows the previous working version to be restored. Other update scenarios where system rollbacks are helpful include:
 - If your application is updated to run with a parallelism higher than [maxParallelism](#).
 - If your application is updated to run with incorrect subnets for a VPC application that results in a failure during the Flink job startup.
- **Flink version upgrades:** When you upgrade to a new Apache Flink version and the upgraded application encounters a snapshot compatibility issue, system rollback lets you revert to the prior Flink version automatically.
- **AutoScaling:** When the application scales up but runs into issues restoring from a savepoint, due to operator mismatch between the snapshot and the Flink job graph.

Operation APIs

To provide better visibility, Amazon Managed Service for Apache Flink has two APIs related to application operations that can help you track failures and related system rollbacks.

ListApplicationOperations

This API lists all operations performed on the application, including `UpdateApplication`, `Maintenance`, `RollbackApplication`, and others in reverse chronological order. The following example request for the `ListApplicationOperations` action lists the first 10 application operations for the application:

```
{
  "ApplicationName": "MyApplication",
  "Limit": 10
}
```

This following example request for `ListApplicationOperations` helps filter the list to previous updates on the application:

```
{
  "ApplicationName": "MyApplication",
  "operation": "UpdateApplication"
}
```

DescribeApplicationOperation

This API provides detailed information about a specific operation listed by `ListApplicationOperations`, including the reason for failure, if applicable. The following example request for the `DescribeApplicationOperation` action lists details for a specific application operation:

```
{
  "ApplicationName": "MyApplication",
  "OperationId": "xyzoperation"
}
```

For troubleshooting information, see [System rollback best practices](#).

Running a Managed Service for Apache Flink application

This topic contains information about running a Managed Service for Apache Flink.

When you run your Managed Service for Apache Flink application, the service creates an Apache Flink job. An Apache Flink job is the execution lifecycle of your Managed Service for Apache Flink 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](#).
 - **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.
- **MAINTENANCE**: Occurs while Managed Service for Apache Flink applies patches to your application. For more information, see [Maintenance](#).

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.

Batch workloads

Managed Service for Apache Flink supports running Apache Flink batch workloads. In a batch job, when an Apache Flink job gets to the **FINISHED** status, Managed Service for Apache Flink application status is set to **READY**. For more information about Flink job statuses, see [Jobs and Scheduling](#).

Application resources

This section describes the system resources that your application uses. Understanding how Managed Service for Apache Flink provisions and uses resources will help you design, create, and maintain a performant and stable Managed Service for Apache Flink application.

Managed Service for Apache Flink application resources

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

One KPIU 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

KPIUs 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 KPIUs available to an application is equal to the application's `Parallelism` setting, divided by the application's `ParallelismPerKPIU` setting.

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

Apache Flink application resources

The Apache Flink environment allocates resources for your application using units called **task slots**. When Managed Service for Apache Flink allocates resources for your application, it assigns one or more Apache Flink task slots to a single KPIU. The number of slots assigned to a single KPIU is equal to your application's `ParallelismPerKPIU` 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 operators, see [Operators](#) 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink with the DataStream API](#): These components move data between your application and external data sources and destinations.
- [Transforming data using operators in Managed Service for Apache Flink with the DataStream API](#): These components transform or group data elements within your application.
- [Tracking events in Managed Service for Apache Flink using the DataStream API](#): This topic describes how Managed Service for Apache Flink tracks events when using the DataStream API.

Using connectors to move data in Managed Service for Apache Flink with the DataStream API

In the Amazon Managed Service for Apache Flink DataStream API, *connectors* are software components that move data into and out of a Managed Service for Apache Flink 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](#): Provide data to your application from a Kinesis data stream, file, or other data source.
- [Sinks](#): Send data from your application to a Kinesis data stream, Firehose stream, or other data destination.
- [Asynchronous I/O](#): 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](#).

Warning

If you have applications running on Flink 1.6, 1.8, 1.11 or 1.13 and would like to run in Middle East (UAE), Asia Pacific (Hyderabad), Israel (Tel Aviv), Europe (Zurich), Middle East (UAE), Asia Pacific (Melbourne) or Asia Pacific (Jakarta) Regions you may need to rebuild your application archive with an updated connector or upgrade to Flink 1.18.

Apache Flink connectors are stored in their own open source repositories. If you're upgrading to version 1.18 or later, you must to update your dependencies. To access the repository for Apache Flink AWS connectors, see [flink-connector-aws](#).

Following are recommended guidelines:

Connector upgrades

Connector used	Resolution
1 EFO	When upgrading to Amazon Managed Service for Apache Flink version 1.15, make sure that you are using the most recent EFO connector. That must be any version 1.15.3 or later. For more information, see: FLINK-29324 .
1 Amazon Data Firehose Sink	When upgrading to Amazon Managed Service for Apache Flink version 1.15, make sure that you are using the most recent Amazon Data Firehose Sink. Amazon Data Firehose Sink

Connector used	Resolution
1 Kafka connectors	<p>When upgrading to Amazon Managed Service for Apache Flink version 1.15, make sure that you are using the most recent Kafka connector APIs. Apache Flink has deprecated FlinkKafkaConsumer and FlinkKafkaProducer. These APIs for the Kafka sink cannot commit to Kafka for Flink 1.15. Make sure that you are using KafkaSource and KafkaSink.</p>
1 Firehose	<p>Your application depends on an outdated version of Firehose connector that is not aware of newer AWS Regions. Rebuild your application archive with Firehose connector version 2.1.0.</p> <p>v2.1.0</p>

F Connector used	Resolution
1 Kinesis	<p>Your application depends on an outdated version of Flink Kinesis connector that is not aware of newer AWS Regions. Rebuild your application archive with Flink Kinesis connector version 1.6.1.</p> <p>https://github.com/aws-labs/amazon-kinesis-connector-flink/tree/1.6.1</p>
1 Kinesis	<p>Your application depends on an outdated version of Flink Kinesis connector that is not aware of newer AWS Regions. Rebuild your application archive with Flink Kinesis connector version 2.4.1.</p> <p>https://github.com/aws-labs/amazon-kinesis-connector-flink/tree/2.4.1</p>

Connector used	Resolution
Kinesis	Your application depends on an outdated version of Flink Kinesis connector that is not aware of newer AWS Regions. Unfortunately, Flink no longer releases patches or bug fixes for 1.6/1.13 connectors. We suggest updating to Flink 1.15 by rebuilding your application archive with Flink 1.15.

Adding streaming data sources to Managed Service 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](#).

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 Managed Service for Apache Flink application that uses an EFO consumer, see [EFO Consumer](#).

Amazon MSK

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

Creating a `KafkaSource`

The following code example demonstrates creating a `KafkaSource`:

```
KafkaSource<String> source = KafkaSource.<String>builder()
    .setBootstrapServers(brokers)
```

```
.setTopics("input-topic")
.setGroupId("my-group")
.setStartingOffsets(OffsetsInitializer.earliest())
.setValueOnlyDeserializer(new SimpleStringSchema())
.build();

env.fromSource(source, WatermarkStrategy.noWatermarks(), "Kafka Source");
```

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

Writing data using sinks in Managed Service for Apache Flink

In your application code, you can use any [Apache Flink sink](#) connector to write into external systems, including AWS services, such as Kinesis Data Streams and DynamoDB.

Apache Flink also provides sinks for files and sockets, and you can implement custom sinks. Among the several supported sinks, the following are frequently used:

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 \(DataStream API\)](#).

Apache Kafka and Amazon Managed Streaming for Apache Kafka (MSK)

The [Apache Flink Kafka connector](#) provides extensive support for publishing data to Apache Kafka and Amazon MSK, including exactly-once guarantees. To learn how to write to Kafka, see [Kafka Connectors examples](#) in the Apache Flink documentation.

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"](#).

Firehose

The `FlinkKinesisFirehoseProducer` is a reliable, scalable Apache Flink sink for storing application output using the [Firehose](#) service. This section describes how to set up a Maven project to create and use a `FlinkKinesisFirehoseProducer`.

Topics

- [Creating a FlinkKinesisFirehoseProducer](#)
- [FlinkKinesisFirehoseProducer Code Example](#)

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 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 Firehose sink, see [the section called "Firehose sink"](#).

Using Asynchronous I/O in Managed Service for Apache Flink

An Asynchronous I/O operator enriches stream data using an external data source such as a database. Managed Service for Apache Flink 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 Managed Service for Apache Flink with the DataStream API

To transform incoming data in a Managed Service for Apache Flink, 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](#)
- [Aggregation operators](#)

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 more code examples, see [Examples](#).

Tracking events in Managed Service for Apache Flink using the DataStream API

Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 [Generating Watermarks](#) 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](#): These components move data between your application and external data sources and destinations.
- [Table API time attributes](#): This topic describes how Managed Service for Apache Flink 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](#): You use Table API source connectors to create tables within your `TableEnvironment` using either API calls or SQL queries.
- [Table API sinks](#): 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 & SQL 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 Managed Service for Apache Flink uses to write the output to the sink. For information about formats, see [Supported 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](#) and [Timely Stream Processing](#) in the Apache Flink Documentation.

Using Python with Managed Service for Apache Flink

Note

If you are developing Python Flink application on a new Mac with Apple Silicon chip, you may encounter some [known issues](#) with Python dependencies of PyFlink 1.15. In this case we recommend running the Python interpreter in Docker. For step-by-step instructions, see [PyFlink 1.15 development on Apple Silicon Mac](#).

Apache Flink version 1.18.1 includes support for creating applications using Python version 3.10. For more information, see [Flink Python Docs](#). You create a Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 the [Table API Tutorial](#) in the Apache Flink Documentation.

Programming your Managed Service for Apache Flink for Python application

You code your Managed Service for Apache Flink 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 Managed Service for Apache Flink, see [Getting started with Amazon Managed Service for Apache Flink for Python](#).

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.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)
```

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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink Python 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 Interface](#) in the Apache Flink Documentation.

Note

Managed Service for Apache Flink 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 Managed Service for Apache Flink application

You use your application's CloudWatch log to monitor your Managed Service for Apache Flink Python application.

Managed Service for Apache Flink 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 Managed Service 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](#)
- [Working with runtime properties in the CLI](#)
- [Accessing runtime properties in a Managed Service for Apache Flink application](#)

Working with runtime properties in the console

You can add, update, or remove runtime properties from your Managed Service for Apache Flink application using the AWS Management Console.

Note

If you are using an earlier supported version of Apache Flink and want to upgrade your existing applications to Apache Flink 1.19.1, you can do so using in-place Apache Flink version upgrades. With in-place version upgrades, you retain application traceability against a single ARN across Apache Flink versions, including snapshots, logs, metrics, tags, Flink configurations, and more. You can use this feature in RUNNING and READY state. For more information, see [In-place version upgrades for Apache Flink](#).

Update Runtime Properties for a Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. Choose your Managed Service for Apache Flink 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 [Managed Service for Apache Flink API example code](#).

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_19",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-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, Managed Service for Apache Flink adds the key-value pair as a new property. If you use a key for an existing runtime property in a property group, Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 code examples, see [Managed Service for Apache Flink: Examples](#).

Using Apache Flink connectors

Apache Flink connectors are software components that move data into and out of an Amazon Managed Service for Apache Flink application. Connectors are flexible integrations that let you 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:** Provide data to your application from a Kinesis data stream, file, Apache Kafka topic, file, or other data sources.
- **Sinks:** Send data from your application to a Kinesis data stream, Firehose stream, Apache Kafka topic, or other data destinations.
- **Asynchronous I/O:** Provides asynchronous access to a data source such as a database to enrich streams.

Apache Flink connectors are stored in their own source repositories. The version and artifact for Apache Flink connectors changes depending on the Apache Flink version you are using, and whether you are using the `DataStream`, `Table`, or `SQL API`.

Amazon Managed Service for Apache Flink supports over 40 pre-built Apache Flink source and sink connectors. The following table provides a summary of the most popular connectors and their associated versions. You can also build custom sinks using the Async-sink framework. For more information, see [The Generic Asynchronous Base Sink](#) in the Apache Flink documentation.

To access the repository for Apache Flink AWS connectors, see [flink-connector-aws](#).

Connectors for Flink versions

Connector	Flink version 1.15	Flink version 1.18	Flink version 1.19
Kinesis Data Stream - Source - DataStream and Table API	flink-connector-kinesis, 1.15.4	flink-connector-kinesis, 4.3.0-1.18	flink-connector-kinesis, 4.3.0-1.19
Kinesis Data Stream - Sink - DataStream and Table API	flink-connector-aws-kinesis-streams, 1.15.4	flink-connector-aws-kinesis-streams, 4.3.0-1.18	flink-connector-aws-kinesis-streams, 4.3.0-1.19
Kinesis Data Streams - Source/Sink - SQL	flink-sql-connector-kinesis, 1.15.4	flink-sql-connector-kinesis, 4.3.0-1.18	flink-sql-connector-kinesis, 4.3.0-1.19
Kafka - DataStream and Table API	flink-connector-kafka, 1.15.4	flink-connector-kafka, 3.2.0-1.18	flink-connector-kafka, 3.2.0-1.19
Kafka - SQL	flink-sql-connector-kafka, 1.15.4	flink-sql-connector-kafka, 3.2.0-1.18	flink-sql-connector-kafka, 3.2.0-1.19
Firehose - DataStream and Table API	flink-connector-aws-kinesis-firehose, 1.15.4	flink-connector-aws-firehose, 4.3.0-1.18	flink-connector-aws-firehose, 4.3.0-1.19
Firehose - SQL	flink-sql-connector-aws-kinesis-firehose, 1.15.4	flink-sql-connector-aws-firehose, 4.3.0-1.18	flink-sql-connector-aws-firehose, 4.3.0-1.19
DynamoDB - DataStream and Table API	flink-connector-dynamodb, 3.0.0-1.15	flink-connector-dynamodb, 4.3.0-1.18	flink-connector-dynamodb, 4.3.0-1.19

Connector	Flink version 1.15	Flink version 1.18	Flink version 1.19
DynamoDB - SQL	flink-sql-connector-dynamodb, 3.0.0-1.15	flink-sql-connector-dynamodb, 4.3.0-1.18	flink-sql-connector-dynamodb, 4.3.0-1.19
OpenSearch - DataStream and Table API	-	flink-connector-opensearch, 1.2.0-1.18	flink-connector-opensearch, 1.2.0-1.19
OpenSearch - SQL	-	flink-sql-connector-opensearch, 1.2.0-1.18	flink-sql-connector-opensearch, 1.2.0-1.19

To learn more about connectors in Amazon Managed Service for Apache Flink, see:

- [DataStream API connectors](#)
- [Table API connectors](#)

Implementing fault tolerance in Managed Service for Apache Flink

Checkpointing is the method that is used for implementing fault tolerance in Amazon Managed Service 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](#).

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 Managed Service 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.

Note

For Flink 1.15 onward, Managed Service for Apache Flink will use `stop-with-savepoint` during Automatic Snapshot Creation, that is, application update, scaling or stopping.

- `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 [Managed Service for Apache Flink API example code](#).

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_19",
  "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_19",
  "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 Managed Service for Apache Flink 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 the 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 lets you recover more recent data, but increases cost and requires more system resources.

In Managed Service for Apache Flink, you manage snapshots using the following API actions:

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

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

Managed Service for Apache Flink 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](#)
- [Restoring from a snapshot that contains incompatible state data](#)
- [Snapshot API examples](#)

Automatic snapshot creation

If `SnapshotsEnabled` is set to `true` in the [ApplicationSnapshotConfiguration](#) for the application, Managed Service for Apache Flink 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.

Note

Managed Service for Apache Flink triggers intermediate savepoints during snapshot creation. For Flink version 1.15 or greater, intermediate savepoints no longer commit any side effects. See [Triggering savepoints](#).

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 Managed Service for Apache Flink 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. For two-phase-commit sinks, we recommend using system snapshot (SwS) instead of user-created snapshot (`CreateApplicationSnapshot`).

For Flink, Managed Service for Apache Flink triggers intermediate savepoints during snapshot creation. For Flink 1.15 onward, intermediate savepoints no longer commit any side effects. See [Triggering Savepoints](#).

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 [Managed Service for Apache Flink API example code](#).

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": "2020-01-01T00:00:00Z"
}
```

```
"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"  
    }  
}  
}
```

In-place version upgrades for Apache Flink

With in-place version upgrades for Apache Flink, you retain application traceability against a single ARN across Apache Flink versions. This includes snapshots, logs, metrics, tags, Flink configurations, resource limit increases, VPCs, and more. You can perform in-place version upgrades for Apache Flink to upgrade existing applications to a new Flink version in Amazon Managed Service for Apache Flink. To perform this task, you can use the AWS CLI, AWS CloudFormation, AWS SDK, or the AWS Management Console.

Note

You can't use in-place version upgrades for Apache Flink with Amazon Managed Service for Apache Flink Studio.

This topic contains the following sections:

- [Upgrading applications using in-place version upgrades for Apache Flink](#)
- [Upgrading your application to a new Apache Flink version](#)
- [Rollback](#)
- [General best practices and recommendations](#)
- [Precautions and known issues](#)

Upgrading applications using in-place version upgrades for Apache Flink

Before you begin, we recommend that you watch this video: [In-Place Version Upgrades](#).

To perform in-place version upgrades for Apache Flink, you can use the AWS CLI, AWS CloudFormation, AWS SDK, or the AWS Management Console. You can use this feature with any existing applications that you use with Managed Service for Apache Flink in a READY or RUNNING state. It uses the UpdateApplication API to add the ability to change the Flink runtime.

Before upgrading: Updating your Apache Flink application

When you write your Flink applications, you bundle them with their dependencies into an application JAR and upload the JAR to your Amazon S3 bucket. From there, Amazon Managed Service for Apache Flink runs the job in the new Flink runtime that you've selected. You might have to update your applications to achieve compatibility with the Flink runtime you want to upgrade to. There can be inconsistencies between Flink versions that cause the version upgrade to fail. Most commonly, this will be with connectors for sources (ingress) or destinations (sinks, egress) and Scala dependencies. Flink 1.15 and later versions in Managed Service for Apache Flink are Scala-agnostic, and your JAR must contain the version of Scala you plan to use.

To update your application

1. Read the advice from the Flink community on upgrading applications with state. See [Upgrading Applications and Flink Versions](#).
2. Read the list of knowing issues and limitations. See [Precautions and known issues](#).
3. Update your dependencies and test your applications locally. These dependencies typically are:
 1. The Flink runtime and API.
 2. Connectors recommended for the new Flink runtime. You can find these on [Release versions](#) for the specific runtime you want to update to.
 3. Scala – Apache Flink is Scala-agnostic starting with and including Flink 1.15. You must include the Scala dependencies you want to use in your application JAR.
4. Build a new application JAR on zipfile and upload it to Amazon S3. We recommend that you use a different name from the previous JAR/zipfile. If you need to roll back, you will use this information.
5. If you are running stateful applications, we strongly recommend that you take a snapshot of your current application. This lets you roll back statefully if you encounter issues during or after the upgrade.

Upgrading your application to a new Apache Flink version

You can upgrade your Flink application by using the [UpdateApplication](#) action.

You can call the `UpdateApplication` API in multiple ways:

- Use the existing **Configuration** workflow on the AWS Management Console.

- Go to your app page on the AWS Management Console.
- Choose **Configure**.
- Select the new runtime and the snapshot that you want to start from, also known as restore configuration. Use the latest setting as the restore configuration to start the app from the latest snapshot. Point to the new upgraded application JAR/zip on Amazon S3.
- Use the AWS CLI [update-application](#) action.
- Use AWS CloudFormation (CFN).
 - Update the [RuntimeEnvironment](#) field. Previously, AWS CloudFormation deleted the application and created a new one, causing your snapshots and other app history to be lost. Now AWS CloudFormation updates your RuntimeEnvironment in place and does not delete your application.
- Use the AWS SDK.
 - Consult the SDK documentation for the programming language of your choice. See [UpdateApplication](#).

You can perform the upgrade while the application is in RUNNING state or while the application is stopped in READY state. Amazon Managed Service for Apache Flink validates to verify the compatibility between the original runtime version and the target runtime version. This compatibility check runs when you perform [UpdateApplication](#) while in RUNNING state or at the next [StartApplication](#) if you upgrade while in READY state.

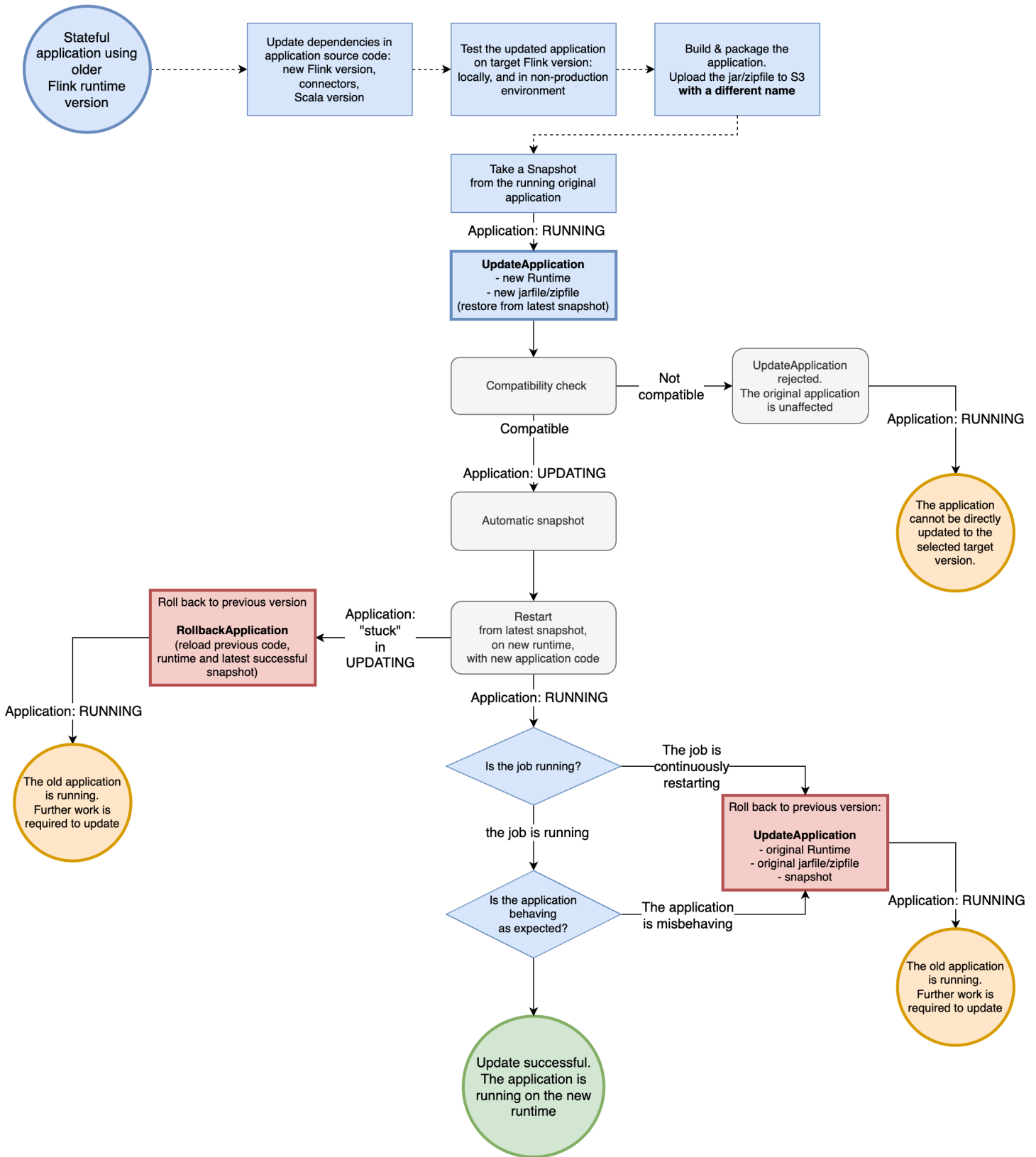
Upgrading an application in RUNNING state

The following example shows upgrading an app in RUNNING state named UpgradeTest to Flink 1.18 in US East (N. Virginia) using the AWS CLI and starting the upgraded app from the latest snapshot.

```
aws --region us-east-1 kinesisanalyticstv2 update-application \
--application-name UpgradeTest --runtime-environment-update "FLINK-1_18" \
--application-configuration-update '{"ApplicationCodeConfigurationUpdate": '\
'{"CodeContentUpdate": {"S3ContentLocationUpdate": '\
'{"FileKeyUpdate": "flink_1_18_app.jar"}}}}' \
--run-configuration-update '{"ApplicationRestoreConfiguration": '\
'{"ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"}}' \
--current-application-version-id ${current_application_version}
```

- If you enabled service snapshots and want to continue the application from the latest snapshot, Amazon Managed Service for Apache Flink verifies that the current RUNNING application's runtime is compatible with the selected target runtime.
- If you have specified a snapshot from which to continue the target runtime, Amazon Managed Service for Apache Flink verifies that the target runtime is compatible with the specified snapshot. If the compatibility check fails, your update request is rejected and your application remains untouched in the RUNNING state.
- If you choose to start your application without a snapshot, Amazon Managed Service for Apache Flink doesn't run any compatibility checks.
- If your upgraded application fails or gets stuck in a transitive UPDATING state, follow the instructions in the [Rollback](#) section to return to the healthy state.

Process flow for running state applications



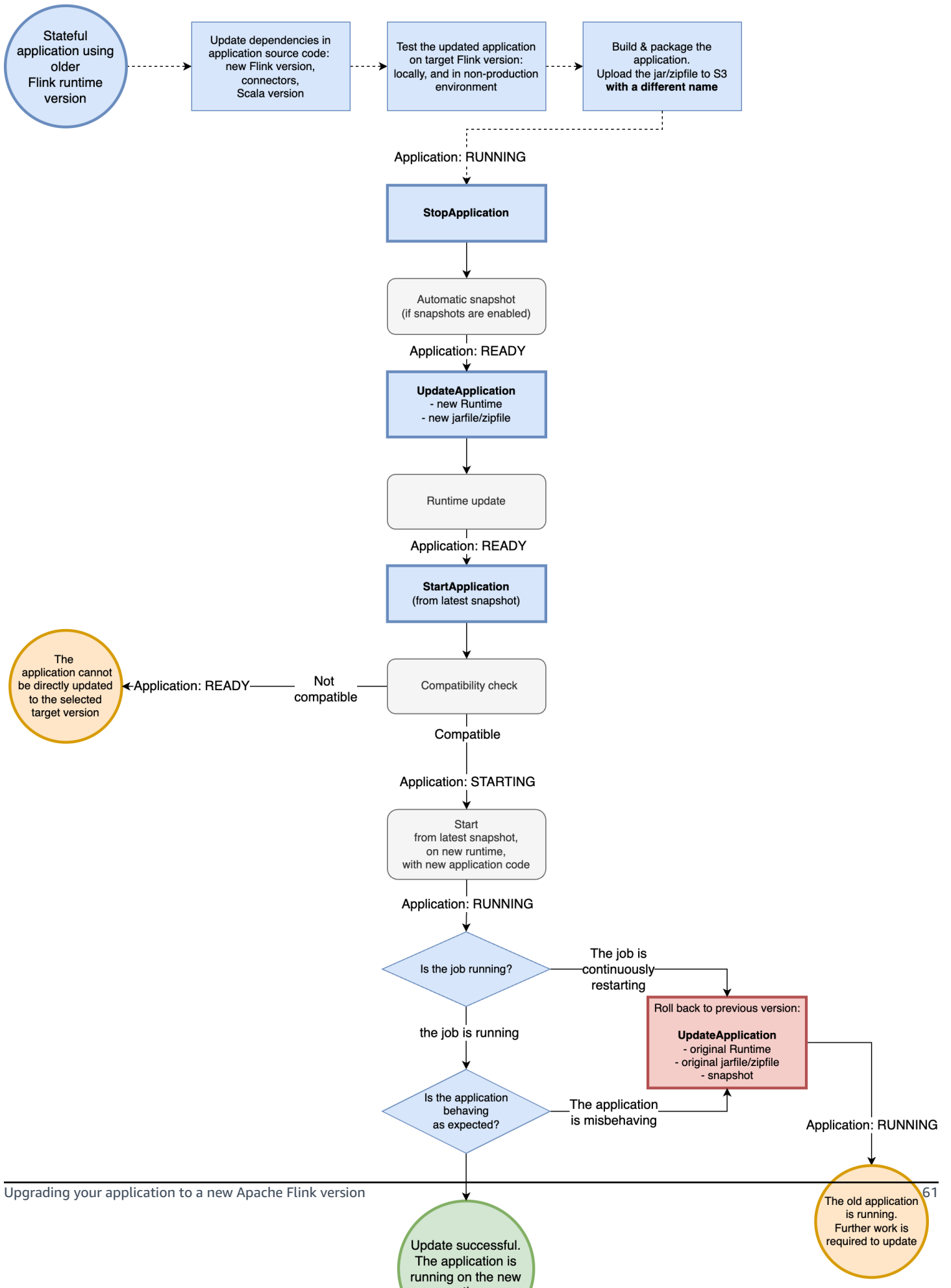
Upgrading an application in READY state

The following example shows upgrading an app in READY state named UpgradeTest to Flink 1.18 in US East (N. Virginia) using the AWS CLI. There is no specified snapshot to start the app because the application is not running. You can specify a snapshot when you issue the start application request.

```
aws --region us-east-1 kinesisanalyticstv2 update-application \  
--application-name UpgradeTest --runtime-environment-update "FLINK-1_18" \  
--application-configuration-update '{"ApplicationCodeConfigurationUpdate": '\'  
'{"CodeContentUpdate": {"S3ContentLocationUpdate": '\'  
'{"FileKeyUpdate": "flink_1_18_app.jar"}}}}' \  
--current-application-version-id ${current_application_version}
```

- You can update the runtime of your applications in READY state to any Flink version. Amazon Managed Service for Apache Flink does not run any checks until you start your application.
- Amazon Managed Service for Apache Flink only runs compatibility checks against the snapshot you selected to start the app. These are basic compatibility checks following the [Flink Compatibility Table](#). They only check the Flink version with which the snapshot was taken and the Flink version you are targeting. If the Flink runtime of the selected snapshot is incompatible with the app's new runtime, the start request might be rejected.

Process flow for ready state applications



Rollback

If you have issues with your application or find inconsistencies in your application code between Flink versions, you can roll back using the AWS CLI, AWS CloudFormation, AWS SDK, or the AWS Management Console. The following examples show what rolling back looks like in different failure scenarios.

Runtime upgrade succeeded, the application is in **RUNNING** state, but the job is failing and continuously restarting

Assume you are trying to upgrade a stateful application named `TestApplication` from Flink 1.15 to Flink 1.18 in US East (N. Virginia). However, the upgraded Flink 1.18 application is failing to start or is constantly restarting, even though the application is in **RUNNING** state. This is a common failure scenario. To avoid further downtime, we recommend that you roll back your application immediately to the previous running version (Flink 1.15), and diagnose the issue later.

To roll back the application to the previous running version, use the [rollback-application](#) AWS CLI command or the [RollbackApplication](#) API action. This API action rolls back the changes you've made that resulted in the latest version. Then it restarts your application using the latest successful snapshot.

We strongly recommend that you take a snapshot with your existing app before you attempt to upgrade. This will help to avoid data loss or having to reprocess data.

In this failure scenario, AWS CloudFormation will not roll back the application for you. You must update the CloudFormation template to point to the previous runtime and to the previous code to force CloudFormation to update the application. Otherwise, CloudFormation assumes that your application has been updated when it transitions to the **RUNNING** state.

Rolling back an application that is stuck in **UPDATING**

If your application gets stuck in the **UPDATING** or **AUTOSCALING** state after an upgrade attempt, Amazon Managed Service for Apache Flink offers the [rollback-applications](#) AWS CLI command, or the [RollbackApplications](#) API action that can roll back the application to the version before the stuck **UPDATING** or **AUTOSCALING** state. This API rolls back the changes that you've made that caused the application to get stuck in **UPDATING** or **AUTOSCALING** transitive state.

General best practices and recommendations

- Test the new job/runtime without state on a non-production environment before attempting a production upgrade.
- Consider testing the stateful upgrade with a non-production application first.
- Make sure that your new job graph has a compatible state with the snapshot you will be using to start your upgraded application.
 - Make sure that the types stored in operator states stay the same. If the type has changed, Apache Flink can't restore the operator state.
 - Make sure that the Operator IDs you set using the `uid` method remain the same. Apache Flink has a strong recommendation for assigning unique IDs to operators. For more information, see [Assigning Operator IDs](#) in the Apache Flink documentation.

If you don't assign IDs to your operators, Flink automatically generates them. In that case, they might depend on the program structure and, if changed, can cause compatibility issues. Flink uses Operator IDs to match state in snapshot to operator. Changing Operator IDs results in the application not starting, or state stored in the snapshot being dropped, and the new operator starting without state.

- Don't change the key used to store the keyed state.
- Don't modify the input type of stateful operators like window or join. This implicitly changes the type of the internal state of the operator, causing a state incompatibility.

Precautions and known issues

Not allowed configuration changes from Flink 1.19 and later

- If you are updating your runtime from Flink 1.18 or earlier to Flink 1.19 or later, Flink job configuration changes using Flink job code are no longer permitted. As a result, the application will fail to submit the job. An error log indicates which not-allowed configurations have been modified at runtime. For more information, see [FlinkRuntimeException: "Not allowed configuration change\(s\) were detected"](#).

Known limitations of state compatibility

- If you are using the Table API, Apache Flink doesn't guarantee state compatibility between Flink versions. For more information, see [Stateful Upgrades and Evolution](#) in the Apache Flink documentation.
- Flink 1.6 states are not compatible with Flink 1.18. The API rejects your request if you try to upgrade from 1.6 to 1.18 and later with state. You can upgrade to 1.8, 1.11, 1.13 and 1.15 and take a snapshot, and then upgrade to 1.18 and later. For more information, see [Upgrading Applications and Flink Versions](#) in the Apache Flink documentation.

Known issues with the Flink Kinesis Connector

- If you are using Flink 1.11 or earlier and using the `amazon-kinesis-connector-flink` connector for Enhanced-fan-out (EFO) support, you must take extra steps for a stateful upgrade to Flink 1.13 or later. This is because of the change in the package name of the connector. For more information, see [amazon-kinesis-connector-flink](#).

The `amazon-kinesis-connector-flink` connector for Flink 1.11 and earlier uses the packaging software `amazon.kinesis`, whereas the Kinesis connector for Flink 1.13 and later uses `org.apache.flink.streaming.connectors.kinesis`. Use this tool to support your migration: [amazon-kinesis-connector-flink-state-migrator](#).

- If you are using Flink 1.13 or earlier with `FlinkKinesisProducer` and upgrading to Flink 1.15 or later, for a stateful upgrade you must continue to use `FlinkKinesisProducer` in Flink 1.15 or later, instead of the newer `KinesisStreamsSink`. However, if you already have a custom `uid` set on your sink, you should be able to switch to `KinesisStreamsSink` because `FlinkKinesisProducer` doesn't keep state. Flink will treat it as the same operator because a custom `uid` is set.

Flink applications written in Scala

- As of Flink 1.15, Apache Flink doesn't include Scala in the runtime. You must include the version of Scala you want to use and other Scala dependencies in your code JAR/zip when upgrading to Flink 1.15 or later. For more information, see [Amazon Managed Service for Apache Flink for Apache Flink 1.15.2 release](#).

- If your application uses Scala and you are upgrading it from Flink 1.11 or earlier (Scala 2.11) to Flink 1.13 (Scala 2.12), make sure that your code uses Scala 2.12. Otherwise, your Flink 1.13 application may fail to find Scala 2.11 classes in the Flink 1.13 runtime.

Things to consider when downgrading Flink application

- Downgrading Flink applications is possible, but limited to cases when the application was previously running with the older Flink version. For a stateful upgrade Managed Service for Apache Flink will require using a snapshot taken with matching or earlier version for the downgrade
- If you are updating your runtime from Flink 1.13 or later to Flink 1.11 or earlier, and if your app uses the HashMap state backend, your application will continuously fail.

Application scaling in Managed Service for Apache Flink

You can configure the parallel execution of tasks and the allocation of resources for Amazon Managed Service 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](#)
- [Allocating Kinesis Processing Units](#)
- [Updating your application's parallelism](#)
- [Automatic scaling](#)

Configuring application parallelism and ParallelismPerKPU

You configure the parallel execution for your Managed Service for Apache Flink 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 64). 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

Managed Service for Apache Flink 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.

Managed Service for Apache Flink 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
```

Managed Service for Apache Flink 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](#).

The default limit for KPUs for your application is 64. For instructions on how to request an increase to this limit, see "To request a limit increase" in [Service Quotas](#).

Note

An additional KPU is charged for orchestrations purposes. For more information, see [Managed Service for Apache Flink pricing](#).

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 [Managed Service for Apache Flink API example code](#).

The following example request for the [CreateApplication](#) action sets parallelism when you are creating an application:

```
{
  "ApplicationName": "string",
  "RuntimeEnvironment": "FLINK-1_18",
  "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

Managed Service for Apache Flink elastically scales your application's parallelism to accommodate the data throughput of your source and your operator complexity for most scenarios. Managed Service for Apache Flink 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) if CloudWatch metric `maximumContainerCPUUtilization` is larger than 75 percent or above for 15 minutes. That means the `ScaleUp` action is triggered when there are 15 consecutive datapoints with 1 minute period equal to or over 75 percent.
- Your application scales down (decreases parallelism) when your CPU usage remains below 10 percent for six hours. That means the `ScaleDown` action is triggered when there are 360 consecutive datapoints with 1 minute period less than 10 percent.

Note

Max of `containerCPUUtilization` over 1 minute period can be referenced to find the correlation with a datapoint used for Scaling action, but it's not necessary to reflect the exact moment when the action is triggered.

Managed Service for Apache Flink will not reduce your application's `CurrentParallelism` value to less than your application's `Parallelism` setting.

When the Managed Service for Apache Flink 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 KPIUs that Managed Service for Apache Flink provisions which is a function of your application's `parallelism` and `parallelismPerKPIU` settings. An activity spike increases your Managed Service for Apache Flink costs.

For information about pricing, see [Amazon Managed Service for Apache Flink 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 64 KPIUs. For more information, see [Quota](#).

- 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”](#) 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 in Managed Service for Apache Flink”](#).

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.

For additional information, see [Enhanced monitoring and automatic scaling for Apache Flink](#)

For an example, see [kda-flink-app-autoscaling](#).

Using tagging

This section describes how to add key-value metadata tags to Managed Service for Apache Flink applications. These tags can be used for the following purposes:

- Determining billing for individual Managed Service for Apache Flink 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](#)
- [Adding or u tags for an existing application](#)
- [Listing tags for an application](#)
- [Removing tags from an application](#)

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:kinesisanalyticsus-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:kinesisanalyticsus-west-2:012345678901:application/MyApplication",
"TagKeys": [ "KeyOfFirstTagToRemove", "KeyOfSecondTagToRemove" ]
}
```

Using CloudFormation with Managed Service for Apache Flink

The following exercises shows how to start a Flink application created via AWS CloudFormation using a Lambda function in the same stack.

Before you begin

Before you begin this exercise, follow the steps on creating a Flink application using AWS CloudFormation at [AWS::KinesisAnalytics::Application](#).

Writing a Lambda function

To start a Flink application after creation or update, we use the `kinesisanalyticsv2 start-application` API. The call will be triggered by an AWS CloudFormation event after Flink application creation. We'll discuss how to set up the stack to trigger the Lambda function later in this exercise, but first we focus on the Lambda function declaration and its code. We use Python3.8 runtime in this example.

```
StartApplicationLambda:
  Type: AWS::Lambda::Function
  DependsOn: StartApplicationLambdaRole
  Properties:
    Description: Starts an application when invoked.
    Runtime: python3.8
    Role: !GetAtt StartApplicationLambdaRole.Arn
    Handler: index.lambda_handler
    Timeout: 30
    Code:
      ZipFile: |
        import logging
        import cfnresponse
        import boto3

        logger = logging.getLogger()
        logger.setLevel(logging.INFO)
```

```
def lambda_handler(event, context):
    logger.info('Incoming CFN event {}'.format(event))

    try:
        application_name = event['ResourceProperties']['ApplicationName']

        # filter out events other than Create or Update,
        # you can also omit Update in order to start an application on Create
        only.

        if event['RequestType'] not in ["Create", "Update"]:
            logger.info('No-op for Application {} because CFN RequestType {} is
            filtered'.format(application_name, event['RequestType']))
            cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

            return

        # use kinesisanalyticv2 API to start an application.
        client_kda = boto3.client('kinesisanalyticv2',
            region_name=event['ResourceProperties']['Region'])

        # get application status.
        describe_response =
        client_kda.describe_application(ApplicationName=application_name)
        application_status = describe_response['ApplicationDetail']
        ['ApplicationStatus']

        # an application can be started from 'READY' status only.
        if application_status != 'READY':
            logger.info('No-op for Application {} because ApplicationStatus {} is
            filtered'.format(application_name, application_status))
            cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

            return

        # create RunConfiguration.
        run_configuration = {
            'ApplicationRestoreConfiguration': {
                'ApplicationRestoreType': 'RESTORE_FROM_LATEST_SNAPSHOT',
            }
        }

        logger.info('RunConfiguration for Application {}:
        {}'.format(application_name, run_configuration))
```

```

        # this call doesn't wait for an application to transfer to 'RUNNING'
state.
        client_kda.start_application(ApplicationName=application_name,
RunConfiguration=run_configuration)

        logger.info('Started Application: {}'.format(application_name))
        cfnresponse.send(event, context, cfnresponse.SUCCESS, {})
except Exception as err:
    logger.error(err)
    cfnresponse.send(event, context, cfnresponse.FAILED, {"Data": str(err)})

```

In the preceding code, Lambda processes incoming AWS CloudFormation events, filters out everything besides Create and Update, gets the application state and start it if the state is READY. To get the application state, you must create the Lambda role, as shown following.

Creating a Lambda role

You create a role for Lambda to successfully “talk” to the application and write logs. This role uses default managed policies, but you might want to narrow it down to using custom policies.

```

StartApplicationLambdaRole:
  Type: AWS::IAM::Role
  DependsOn: TestFlinkApplication
  Properties:
    Description: A role for lambda to use while interacting with an application.
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service:
              - lambda.amazonaws.com
          Action:
            - sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/Amazonmanaged-flinkFullAccess
      - arn:aws:iam::aws:policy/CloudWatchLogsFullAccess
    Path: /

```

Note that the Lambda resources will be created after creation of the Flink application in the same stack because they depend on it.

Invoking the Lambda function

Now all that is left is to invoke the Lambda function. You do this by using a [custom resource](#).

```
StartApplicationLambdaInvoke:
  Description: Invokes StartApplicationLambda to start an application.
  Type: AWS::CloudFormation::CustomResource
  DependsOn: StartApplicationLambda
  Version: "1.0"
  Properties:
    ServiceToken: !GetAtt StartApplicationLambda.Arn
    Region: !Ref AWS::Region
    ApplicationName: !Ref TestFlinkApplication
```

This is all you need to start your Flink application using Lambda. You are now ready to create your own stack or use the full example below to see how all those steps work in practice.

Full example

The following example is a slightly extended version of the previous steps with an additional RunConfiguration adjusting done via [template parameters](#). This is a working stack for you to try. Be sure to read the accompanying notes:

stack.yaml

```
Description: 'kinesisanalyticsv2 CloudFormation Test Application'
Parameters:
  ApplicationRestoreType:
    Description: ApplicationRestoreConfiguration option, can
    be SKIP_RESTORE_FROM_SNAPSHOT, RESTORE_FROM_LATEST_SNAPSHOT or
    RESTORE_FROM_CUSTOM_SNAPSHOT.
    Type: String
    Default: SKIP_RESTORE_FROM_SNAPSHOT
    AllowedValues: [ SKIP_RESTORE_FROM_SNAPSHOT, RESTORE_FROM_LATEST_SNAPSHOT,
    RESTORE_FROM_CUSTOM_SNAPSHOT ]
  SnapshotName:
    Description: ApplicationRestoreConfiguration option, name of a snapshot to restore
    to, used with RESTORE_FROM_CUSTOM_SNAPSHOT ApplicationRestoreType.
    Type: String
    Default: ''
  AllowNonRestoredState:
    Description: FlinkRunConfiguration option, can be true or false.
```

```
Default: true
Type: String
AllowedValues: [ true, false ]
CodeContentBucketArn:
  Description: ARN of a bucket with application code.
  Type: String
CodeContentFileKey:
  Description: A jar filename with an application code inside a bucket.
  Type: String
Conditions:
  IsSnapshotNameEmpty: !Equals [ !Ref SnapshotName, '' ]
Resources:
  TestServiceExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - kinesisanalytics.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonKinesisFullAccess
        - arn:aws:iam::aws:policy/AmazonS3FullAccess
      Path: /
  InputKinesisStream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
  OutputKinesisStream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
  TestFlinkApplication:
    Type: 'AWS::kinesisanalyticsv2::Application'
    Properties:
      ApplicationName: 'CFNTestFlinkApplication'
      ApplicationDescription: 'Test Flink Application'
      RuntimeEnvironment: 'FLINK-1_18'
      ServiceExecutionRole: !GetAtt TestServiceExecutionRole.Arn
      ApplicationConfiguration:
        EnvironmentProperties:
```

```
PropertyGroups:
  - PropertyGroupId: 'KinesisStreams'
    PropertyMap:
      INPUT_STREAM_NAME: !Ref InputKinesisStream
      OUTPUT_STREAM_NAME: !Ref OutputKinesisStream
      AWS_REGION: !Ref AWS::Region
FlinkApplicationConfiguration:
  CheckpointConfiguration:
    ConfigurationType: 'CUSTOM'
    CheckpointingEnabled: True
    CheckpointInterval: 1500
    MinPauseBetweenCheckpoints: 500
  MonitoringConfiguration:
    ConfigurationType: 'CUSTOM'
    MetricsLevel: 'APPLICATION'
    LogLevel: 'INFO'
  ParallelismConfiguration:
    ConfigurationType: 'CUSTOM'
    Parallelism: 1
    ParallelismPerKPU: 1
    AutoScalingEnabled: True
  ApplicationSnapshotConfiguration:
    SnapshotsEnabled: True
  ApplicationCodeConfiguration:
    CodeContent:
      S3ContentLocation:
        BucketARN: !Ref CodeContentBucketArn
        FileKey: !Ref CodeContentFileKey
      CodeContentType: 'ZIPFILE'
StartApplicationLambdaRole:
  Type: AWS::IAM::Role
  DependsOn: TestFlinkApplication
  Properties:
    Description: A role for lambda to use while interacting with an application.
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service:
              - lambda.amazonaws.com
          Action:
            - sts:AssumeRole
  ManagedPolicyArns:
```

```

    - arn:aws:iam::aws:policy/Amazonmanaged-flinkFullAccess
    - arn:aws:iam::aws:policy/CloudWatchLogsFullAccess
  Path: /
StartApplicationLambda:
  Type: AWS::Lambda::Function
  DependsOn: StartApplicationLambdaRole
  Properties:
    Description: Starts an application when invoked.
    Runtime: python3.8
    Role: !GetAtt StartApplicationLambdaRole.Arn
    Handler: index.lambda_handler
    Timeout: 30
  Code:
    ZipFile: |
      import logging
      import cfnresponse
      import boto3

      logger = logging.getLogger()
      logger.setLevel(logging.INFO)

      def lambda_handler(event, context):
          logger.info('Incoming CFN event {}'.format(event))

          try:
              application_name = event['ResourceProperties']['ApplicationName']

              # filter out events other than Create or Update,
              # you can also omit Update in order to start an application on Create
              # only.
              if event['RequestType'] not in ["Create", "Update"]:
                  logger.info('No-op for Application {} because CFN RequestType {} is
filtered'.format(application_name, event['RequestType']))
                  cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

              return

              # use kinesisanalyticsv2 API to start an application.
              client_kda = boto3.client('kinesisanalyticsv2',
region_name=event['ResourceProperties']['Region'])

              # get application status.
              describe_response =
client_kda.describe_application(ApplicationName=application_name)

```

```

        application_status = describe_response['ApplicationDetail']
['ApplicationStatus']

        # an application can be started from 'READY' status only.
        if application_status != 'READY':
            logger.info('No-op for Application {} because ApplicationStatus {} is
filtered'.format(application_name, application_status))
            cfresponse.send(event, context, cfresponse.SUCCESS, {})

        return

        # create RunConfiguration from passed parameters.
        run_configuration = {
            'FlinkRunConfiguration': {
                'AllowNonRestoredState': event['ResourceProperties']
['AllowNonRestoredState'] == 'true'
            },
            'ApplicationRestoreConfiguration': {
                'ApplicationRestoreType': event['ResourceProperties']
['ApplicationRestoreType'],
            }
        }

        # add SnapshotName to RunConfiguration if specified.
        if event['ResourceProperties']['SnapshotName'] != '':
            run_configuration['ApplicationRestoreConfiguration']['SnapshotName'] =
event['ResourceProperties']['SnapshotName']

        logger.info('RunConfiguration for Application {}:
{}'.format(application_name, run_configuration))

        # this call doesn't wait for an application to transfer to 'RUNNING'
state.
        client_kda.start_application(ApplicationName=application_name,
RunConfiguration=run_configuration)

        logger.info('Started Application: {}'.format(application_name))
        cfresponse.send(event, context, cfresponse.SUCCESS, {})
    except Exception as err:
        logger.error(err)
        cfresponse.send(event, context, cfresponse.FAILED, {"Data": str(err)})
StartApplicationLambdaInvoke:
    Description: Invokes StartApplicationLambda to start an application.
    Type: AWS::CloudFormation::CustomResource

```



```

DependsOn: StartApplicationLambda
Version: "1.0"
Properties:
  ServiceToken: !GetAtt StartApplicationLambda.Arn
  Region: !Ref AWS::Region
  ApplicationName: !Ref TestFlinkApplication
  ApplicationRestoreType: !Ref ApplicationRestoreType
  SnapshotName: !Ref SnapshotName
  AllowNonRestoredState: !Ref AllowNonRestoredState

```

Again, you might want to adjust the roles for Lambda as well as an application itself.

Before creating the stack above, don't forget to specify your parameters.

parameters.json

```

[
  {
    "ParameterKey": "CodeContentBucketArn",
    "ParameterValue": "YOUR_BUCKET_ARN"
  },
  {
    "ParameterKey": "CodeContentFileKey",
    "ParameterValue": "YOUR_JAR"
  },
  {
    "ParameterKey": "ApplicationRestoreType",
    "ParameterValue": "SKIP_RESTORE_FROM_SNAPSHOT"
  },
  {
    "ParameterKey": "AllowNonRestoredState",
    "ParameterValue": "true"
  }
]

```

Replace YOUR_BUCKET_ARN and YOUR_JAR with your specific requirements. You can follow this [guide](#) to create an Amazon S3 bucket and an application jar.

Now create the stack (replace YOUR_REGION with a region of your choice, e.g. us-east-1):

```

aws cloudformation create-stack --region YOUR_REGION --template-body "file://
stack.yaml" --parameters "file://parameters.json" --stack-name "TestManaged Service for
Apache FlinkStack" --capabilities CAPABILITY_NAMED_IAM

```

You can now navigate to <https://console.aws.amazon.com/cloudformation> and view the progress. Once created you should see your Flink application in Starting state. It may take a few minutes until it will start Running.

For more information, see the following:

- [Four ways to retrieve any AWS service property using AWS CloudFormation \(Part 1 of 3\)](#).
- [Walkthrough: Looking up Amazon Machine Image IDs](#).

Using the Apache Flink Dashboard with Managed Service for Apache Flink

You can use your application's Apache Flink Dashboard to monitor your Managed Service for Apache Flink 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 Managed Service for Apache Flink applications:

- The Apache Flink Dashboard for Managed Service for Apache Flink applications is read-only. You can't make changes to your Managed Service for Apache Flink 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 Managed Service for Apache Flink console, or by requesting a secure URL endpoint using the CLI.

Accessing your application's Apache Flink Dashboard using the Managed Service for Apache Flink console

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

Note

When you open the dashboard from the Managed Service for Apache Flink console, the URL that the console generates will be valid for 12 hours.

Accessing your application's Apache Flink Dashboard using the Managed Service for Apache Flink CLI

You can use the Managed Service for Apache Flink 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.

Release versions

This topic contains information about the features supported and component versions recommended for the each release of Managed Service for Apache Flink.

Note

If you are using a version of Apache Flink that is deprecating, we recommend that you upgrade your application to the most recent supported Flink version using the [In-place version upgrades for Apache Flink](#) feature in Managed Service for Apache Flink.

Apache Flink version	Status - Amazon Managed Service for Apache Flink	Status - Apache Flink community	Link
1.19.1	Supported	Supported	Amazon Managed Service for Apache Flink 1.19
1.18.1	Supported	Supported	Amazon Managed Service for Apache Flink 1.18
1.15.2	Supported	Unsupported	Amazon Managed Service for Apache Flink 1.15
1.13.1	Supported	Unsupported	Getting started: Flink 1.13.2
1.11.1	Deprecating	Unsupported	Earlier version information for Managed Service for Apache Flink (deprecating November 5, 2024)

Apache Flink version	Status - Amazon Managed Service for Apache Flink	Status - Apache Flink community	Link
1.8.2	Deprecating	Unsupported	Earlier version information for Managed Service for Apache Flink (deprecating November 5, 2024)
1.6.2	Deprecating	Unsupported	Earlier version information for Managed Service for Apache Flink (deprecating November 5, 2024)

Topics

- [Amazon Managed Service for Apache Flink 1.19](#)
- [Amazon Managed Service for Apache Flink 1.18](#)
- [Amazon Managed Service for Apache Flink 1.15](#)
- [Earlier version information for Managed Service for Apache Flink](#)

Amazon Managed Service for Apache Flink 1.19

Managed Service for Apache Flink now supports Apache Flink version 1.19.1. This section introduces you to the key new features and changes introduced with Managed Service for Apache Flink support of Apache Flink 1.19.1.

Note

If you are using an earlier supported version of Apache Flink and want to upgrade your existing applications to Apache Flink 1.19.1, you can do so using in-place Apache Flink version upgrades. For more information, see [In-place version upgrades for Apache Flink](#).

With in-place version upgrades, you retain application traceability against a single ARN across Apache Flink versions, including snapshots, logs, metrics, tags, Flink configurations, and more.


Supported features

Apache Flink 1.19.1 introduces improvements in the SQL API, such as named parameters, custom source parallelism, and different state TTLs for various Flink operators.

Supported features and related documentation

Supported features	Description	Apache Flink documentation reference
SQL API: Support Configuring Different State TTLs using SQL Hint	Users can now configure state TTL on stream regular joins and group aggregate.	FLIP-373: Configuring Different State TTLs using SQL Hint
SQL API: Support named parameters for functions and call procedures	Users can now use named parameters in functions, rather than relying on the order of parameters.	FLIP-378: Support named parameters for functions and call procedures
SQL API: Setting parallelism for SQL sources	Users can now specify parallelism for SQL sources.	FLIP-367: Support Setting Parallelism for Table/SQL Sources
SQL API: Support Session Window TVF	Users can now use session window Table-Valued Functions.	FLINK-24024: Support session Window TVF
SQL API: Window TVF Aggregation Supports Changelog Inputs	Users can now perform window aggregation on changelog inputs.	FLINK-20281: Window aggregation supports changelog stream input
Support Python 3.11	Flink now supports Python 3.11, which is 10-60% faster compared to Python 3.10. For	FLINK-33030: Add python 3.11 support

Supported features	Description	Apache Flink documentation reference
	more information, see What's New in Python 3.11 .	
Provide metrics for TwoPhaseCommitting sink	Users can view statistics around the status of committers in two phase committing sinks.	FLIP-371: Provide initialization context for Committer creation in TwoPhaseCommittingSink
Trace Reporters for job restart and checkpointing	Users can now monitor traces around checkpoint duration and recovery trends. In Amazon Managed Service for Apache Flink, we enable Slf4j trace reporters by default, so users can monitor checkpoint and job traces through application CloudWatch Logs.	FLIP-384: Introduce TraceReporter and use it to create checkpointing and recovery traces

 **Note**

You can opt into the following features by submitting a [support case](#):

Opt-in features and related documentation

Opt-in features	Description	Apache Flink documentation reference
Support using larger checkpointing interval when source is processing backlog	This is an opt-in feature, because users must tune the configuration for their specific job requirements.	FLIP-309: Support using larger checkpointing interval when source is processing backlog

Opt-in features	Description	Apache Flink documentation reference
Redirect System.out and System.err to Java logs	This is an opt-in feature. On Amazon Managed Service for Apache Flink, the default behavior is to ignore output from System.out and System.err because best practice in production is to use the native Java logger.	FLIP-390: Support System out and err to be redirected to LOG or discarded

For the Apache Flink 1.19.1 release documentation, see [Apache Flink Documentation v1.19.1](#).

Changes in Amazon Managed Service for Apache Flink 1.19.1

Logging Trace Reporter enabled by default

Apache Flink 1.19.1 introduced checkpoint and recovery traces, enabling users to better debug checkpoint and job recovery issues. In Amazon Managed Service for Apache Flink, these traces are logged into the CloudWatch log stream, allowing users to break down the time spent on job initialization, and record historical size of checkpoints.

Default restart strategy is now exponential-delay

In Apache Flink 1.19.1, there are significant improvements to the exponential-delay restart strategy. In Amazon Managed Service for Apache Flink from Flink 1.19.1 onwards, Flink jobs use the exponential-delay restart strategy by default. This means that user jobs will recover quicker from transient errors, but will not overload external systems if job restarts persist.

Specific Flink job configuration using Flink job code is disabled

In Apache Flink 1.18 and earlier versions, all programmatic configuration changes were tolerated by Amazon Managed Service for Apache Flink, with some configurations being silently overridden. From Apache Flink 1.19 and later, we have disabled Flink job configuration changes using Flink job code. If users specify this configuration, their job will throw a `ProgramInvocationException` listing. For more information, see [FlinkRuntimeException: "Not allowed configuration change\(s\) were detected"](#).

Components

Component	Version
Java	11 (recommended)
Python	3.11
Kinesis Data Analytics Flink Runtime (aws-kinesisanalytics-runtime)	1.2.0
Connectors	For information about available connectors, see Apache Flink connectors .
Apache Beam (Beam applications only)	There is no compatible Apache Flink Runner for Flink 1.19. For more information, see Flink Version Compatibility .

Known issues

Apache Beam

There is presently no compatible Apache Flink Runner for Flink 1.19 in Apache Beam. For more information, see [Flink Version Compatibility](#).

Amazon Managed Service for Apache Flink Studio

Studio uses Apache Zeppelin notebooks to provide a single-interface development experience for developing, debugging code, and running Apache Flink stream processing applications. An upgrade is required to Zeppelin's Flink Interpreter to enable support of Flink 1.19. This work is scheduled with the Zeppelin community and we will update these notes when it is complete. You can continue to use Flink 1.15 with Amazon Managed Service for Apache Flink Studio. For more information, see [Creating a Studio notebook](#).

Amazon Managed Service for Apache Flink 1.18

Managed Service for Apache Flink now supports Apache Flink version 1.18.1. Learn about the key new features and changes introduced with Managed Service for Apache Flink support of Apache Flink 1.18.1.

Note

If you are using an earlier supported version of Apache Flink and want to upgrade your existing applications to Apache Flink 1.18.1, you can do so using in-place Apache Flink version upgrades. With in-place version upgrades, you retain application traceability against a single ARN across Apache Flink versions, including snapshots, logs, metrics, tags, Flink configurations, and more. You can use this feature in RUNNING and READY state. For more information, see [In-place version upgrades for Apache Flink](#).

Supported Features	Description	Apache Flink documentation reference
Opensearch connector	This connector includes a sink that provides at-least-once guarantees.	github: Opensearch Connector
Amazon DynamoDB connector	This connector includes a sink that provides at-least-once guarantees.	Amazon DynamoDB Sink
MongoDB connector	This connector includes a source and sink that provide at-least-once guarantees.	MongoDB Connector
Decouple Hive with Flink planner	You can use the Hive dialect directly without the extra JAR swapping.	FLINK-26603: Decouple Hive with Flink planner
Disable WAL in RocksDBWriteBatchWrapper by default	This provides faster recovery times.	FLINK-32326: Disable WAL in RocksDBWriteBatchWrapper by default
Improve the watermark aggregation performance when enabling the watermark alignment	Improves the watermark aggregation performance when enabling the watermark	FLINK-32524: Watermark aggregation performance

Supported Features	Description	Apache Flink documentation reference
	alignment, and adds the related benchmark.	
Make watermark alignment ready for production use	Removes risk of large jobs overloading JobManager	FLINK-32548: Make watermark alignment ready
Configurable RateLimitingStrategy for Async Sink	RateLimitingStrategy lets you configure the decision of what to scale, when to scale, and how much to scale.	FLIP-242: Introduce configurable RateLimitingStrategy for Async Sink
Bulk fetch table and column statistics	Improved query performance.	FLIP-247: Bulk fetch of table and column statistics for given partitions

For the Apache Flink 1.18.1 release documentation, see [Apache Flink 1.18.1 Release Announcement](#).

Changes in Amazon Managed Service for Apache Flink with Apache Flink 1.18

Akka replaced with Pekko

Apache Flink replaced Akka with Pekko in Apache Flink 1.18. This change is fully supported in Managed Service for Apache Flink from Apache Flink 1.18.1 and later. You don't need to modify your applications as a result of this change. For more information, see [FLINK-32468: Replace Akka by Pekko](#).

Support PyFlink Runtime execution in Thread Mode

This Apache Flink change introduces a new execution mode for the Pyflink Runtime framework, Process Mode. Process Mode can now execute Python user-defined functions in the same thread instead of a separate process.

Components

Component	Version
Java	11 (recommended)
Scala	Since version 1.15, Flink is Scala-agnostic. For reference, MSF Flink 1.18 has been verified against Scala 3.3 (LTS).
Managed Service for Apache Flink Flink Runtime (aws-kinesisanalytics-runtime)	1.2.0
AWS Kinesis Connector (flink-connector-kinesis)[Source]	4.2.0-1.18
AWS Kinesis Connector (flink-connector-kinesis)[Sink]	4.2.0-1.18
Apache Beam (Beam applications only)	Earlier and up to version 2.75.0. For more information, see Flink Version Compatibility .

Bug fixes

State compression in Apache Flink 1.18.1

Apache Flink offers optional compression (default: off) for all checkpoints and savepoints. Apache Flink identified a bug in Flink 1.18.1 where the operator state couldn't be properly restored when snapshot compression was enabled. This could result in either data loss or inability to restore from checkpoint. For more information, see [FLINK-34063: When snapshot compression is enabled, rescaling of a source operator leads to some splits getting lost](#).

To resolve this, Amazon Managed Service for Apache Flink has backported the fix that will be included in future versions of Apache Flink. For more information, see [github: Always flush compression buffers](#).

Known issues

Amazon Managed Service for Apache Flink Studio

Studio uses Apache Zeppelin notebooks to provide a single-interface development experience for developing, debugging code, and running Apache Flink stream processing applications. An upgrade is required to Zeppelin's Flink Interpreter to enable support of Flink 1.18. This work is scheduled with the Zeppelin community and we will update these notes when it is complete. You can continue to use Flink 1.15 with Amazon Managed Service for Apache Flink Studio. For more information, see [Creating a Studio notebook](#).

Amazon Managed Service for Apache Flink 1.15

Managed Service for Apache Flink supports the following new features in Apache 1.15.2:

Feature	Description	Apache FLIP reference
Async Sink	An AWS contributed framework for building async destinations that allows developers to build custom AWS connectors with less than half the previous effort. For more information, see The Generic Asynchronous Base Sink .	FLIP-171: Async Sink .
Kinesis Data Firehose Sink	AWS has contributed a new Amazon Kinesis Firehose Sink using the Async framework.	Amazon Kinesis Data Firehose Sink .
Stop with Savepoint	Stop with Savepoint ensures a clean stop operation, most importantly supporting exactly-once semantics for customers that rely on them.	FLIP-34: Terminate/Suspend Job with Savepoint .
Scala Decoupling	Users can now leverage the Java API from any Scala version, including Scala 3. Customers will need to bundle the Scala standard	FLIP-28: Long-term goal of making flink-table Scala-free .

Feature	Description	Apache FLIP reference
	library of their choice in their Scala applications.	
Scala	See Scala decoupling above	FLIP-28: Long-term goal of making flink-table Scala-free.
Unified Connector Metrics	Flink has defined standard metrics for jobs, tasks and operators. Managed Service for Apache Flink will continue to support sink and source metrics and in 1.15 introduce <code>numRestarts</code> in parallel with <code>fullRestarts</code> for Availability Metrics.	FLIP-33: Standardize Connector Metrics and FLIP-179: Expose Standardized Operator Metrics.
Checkpointing finished tasks	This feature is enabled by default in Flink 1.15 and makes it possible to continue performing checkpoints even if parts of the job graph have finished processing all data, which might happen if it contains bounded (batch) sources.	FLIP-147: Support Checkpoints After Tasks Finished.

Changes in Amazon Managed Service for Apache Flink with Apache Flink 1.15

Studio notebooks

Managed Service for Apache Flink Studio now supports Apache Flink 1.15. Managed Service for Apache Flink Studio utilizes Apache Zeppelin notebooks to provide a single-interface development experience for developing, debugging code, and running Apache Flink stream processing

applications. You can learn more about Managed Service for Apache Flink Studio and how to get started at [Using a Studio notebook with Managed Service for Apache Flink](#).

EFO connector

When upgrading to Managed Service for Apache Flink version 1.15, ensure that you are using the most recent EFO Connector, that is any version 1.15.3 or newer. For more information as to why, see [FLINK-29324](#).

Scala Decoupling

Starting with Flink 1.15.2, you will need to bundle the Scala standard library of your choice in your Scala applications.

Kinesis Data Firehose Sink

When upgrading to Managed Service for Apache Flink version 1.15, ensure that you are using the most recent [Amazon Kinesis Data Firehose Sink](#).

Kafka Connectors

When upgrading to Amazon Managed Service for Apache Flink for Apache Flink version 1.15, ensure that you are using the most recent Kafka connector APIs. Apache Flink has deprecated [FlinkKafkaConsumer](#) and [FlinkKafkaProducer](#). These APIs for the Kafka sink cannot commit to Kafka for Flink 1.15. Ensure you are using [KafkaSource](#) and [KafkaSink](#).

Components

Component	Version
Java	11 (recommended)
Scala	2.12
Managed Service for Apache Flink Flink Runtime (aws-kinesisanalytics-runtime)	1.2.0
AWS Kinesis Connector (flink-connector-kinesis)	1.15.4
Apache Beam (Beam applications only)	2.33.0, with Jackson version 2.12.2

Earlier version information for Managed Service for Apache Flink

Note

Apache Flink versions **1.6**, **1.8**, and **1.11** have not been supported by the Apache Flink community for over three years. We plan to deprecate these versions in Amazon Managed Service for Apache Flink on **November 5, 2024**. Starting from this date, you will not be able to create new applications for these Flink versions. You can continue running existing applications at this time. You can upgrade your applications statefully using the in-place version upgrades feature in Amazon Managed Service for Apache Flink. For more information, see [In-place version upgrades for Apache Flink](#).

Versions **1.15.2**, and **1.13.2** of Apache Flink are supported by Managed Service for Apache Flink, but are no longer supported by the Apache Flink community.

This topic contains the following sections:

- [Using the Apache Flink Kinesis Streams connector with previous Apache Flink versions](#)
- [Building applications with Apache Flink 1.8.2](#)
- [Building applications with Apache Flink 1.6.2](#)
- [Upgrading applications](#)
- [Available connectors in Apache Flink 1.6.2 and 1.8.2](#)
- [Getting started: Flink 1.13.2](#)
- [Getting started: Flink 1.11.1 - deprecating](#)
- [Getting started: Flink 1.8.2 - deprecating](#)
- [Getting started: Flink 1.6.2 - deprecating](#)
- [Earlier version \(legacy\) examples for Managed Service for Apache Flink](#)

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 Managed Service for Apache Flink applications that work with Apache Flink 1.8.2.

Use the following component versions for Managed Service for Apache Flink applications:

Component	Version
Java	1.8 (recommended)
Apache Flink	1.8.2
Managed Service for Apache Flink for Flink Runtime (aws-kinesisanalytics-runtime)	1.0.1
Managed Service for Apache Flink 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 Managed Service for Apache Flink application that uses Apache Flink version 1.8.2, see the [Managed Service for Apache Flink 1.8.2 Getting Started Application](#).

For information about how to build and use application code for a Managed Service for Apache Flink application, see [Creating applications](#).

Building applications with Apache Flink 1.6.2

This section contains information about components that you use for building Managed Service for Apache Flink applications that work with Apache Flink 1.6.2.

Use the following component versions for Managed Service for Apache Flink applications:

Component	Version
Java	1.8 (recommended)
AWS Java SDK	1.11.379
Apache Flink	1.6.2
Managed Service for Apache Flink for Flink Runtime (aws-kinesisanalytics-runtime)	1.0.1
Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application that uses Apache Flink version 1.6.2, see the [Managed Service for Apache Flink 1.6.2 Getting Started Application](#).

For information about how to build and use application code for a Managed Service for Apache Flink application, see [Creating applications](#).

Upgrading applications

To upgrade the Apache Flink version of an Amazon Managed Service for Apache Flink application, use the in-place Apache Flink version upgrade feature using the AWS CLI, AWS SDK, AWS CloudFormation, or the AWS Management Console. For more information, see [In-place version upgrades for Apache Flink](#).

You can use this feature with any existing applications you use with Amazon Managed Service for Apache Flink in READY or RUNNING state.

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 1.13.2

This section introduces you to the fundamental concepts of Managed Service 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 Managed Service for Apache Flink application](#)
- [Prerequisites for completing the exercises](#)
- [Step 1: Set up an AWS account and create an administrator user](#)
- [Next step](#)
- [Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)
- [Step 3: Create and run a Managed Service for Apache Flink application](#)
- [Step 4: Clean up AWS resources](#)
- [Step 5: Next steps](#)

Components of a Managed Service for Apache Flink application

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

Managed Service 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](#).
- **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](#).
- **Sink:** The application produces data to external sources by using *sinks*. A sink connector writes data to a Kinesis data stream, a Firehose stream, an Amazon S3 bucket, etc. For more information, see [Sinks](#).

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 Managed Service for Apache Flink 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](#).

Step 1: Set up an AWS account and create an administrator user

Sign up for an AWS account

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

To sign up for an AWS account

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

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

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

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

Create a user with administrative access

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

Secure your AWS account root user

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

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

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

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

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

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

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

Sign in as the user with administrative access

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

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

Assign access to additional users

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

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

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

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. <ul style="list-style-type: none"> For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the <i>AWS Command Line Interface User Guide</i>. For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the <i>AWS SDKs and Tools Reference Guide</i>.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the <i>IAM User Guide</i> .
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. <ul style="list-style-type: none"> For the AWS CLI, see Authenticating using IAM user credentials in the <i>AWS Command Line Interface User Guide</i>. For AWS SDKs and tools, see Authenticate using long-term credentials in

Which user needs programmatic access?	To	By
		the <i>AWS SDKs and Tools Reference Guide</i> . <ul style="list-style-type: none">• For AWS APIs, see Managing access keys for IAM users in the <i>IAM User Guide</i>.

Next step

[Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)

Next step

[Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)

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

In this step, you download and configure the AWS CLI to use with Managed Service 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 Managed Service for Apache Flink application](#)

Step 3: Create and run a Managed Service for Apache Flink application

In this exercise, you create a Managed Service for Apache Flink application with data streams as a source and a sink.

This section contains the following steps:

- [Create two Amazon Kinesis data streams](#)
- [Write sample records to the input stream](#)
- [Download and examine the Apache Flink streaming Java code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Next step](#)

Create two Amazon Kinesis data streams

Before you create a Managed Service for Apache Flink 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 kinesys 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', region_name='us-west-2'))
```

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-examples.git
```

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 Managed Service 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](#).

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 [Fulfill the prerequisites for completing the exercises](#).

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 Managed Service for Apache Flink application

You can create and run a Managed Service for Apache Flink 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 \(console\)](#)
- [Create and run the application \(AWS CLI\)](#)

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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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**.
4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
5. Choose **Create application**.

Note

When you create a Managed Service 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: `kinesisanalytics-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. Enter the following:

Group ID	Key	Value
ProducerConfigProperties	flink.inputstream.initpos	LATEST
ProducerConfigProperties	aws.region	us-west-2

Group ID	Key	Value
ProducerConfigProperties	AggregationEnabled	false

5. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
6. For **CloudWatch logging**, select the **Enable** check box.
7. Choose **Update**.

Note

When you choose to enable Amazon CloudWatch logging, Managed Service for Apache Flink 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 Managed Service for Apache Flink application. Managed Service for Apache Flink uses the `kinesisanalyticsv2` AWS CLI command to create and interact with Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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. Managed Service for Apache Flink 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 Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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 Managed Service for Apache Flink permission to assume the role, and the permissions policy determines what Managed Service for Apache Flink 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 **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-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, Managed Service for Apache Flink 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"](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the **AKReadSourceStreamWriteSinkStream** 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 Managed Service for Apache Flink 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_15",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-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 Managed Service for Apache Flink 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"](#).

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"](#) 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": "SAMPLEUehYngP87ex1nzYIGYgfhyvDU"
      }
    }
  }
}
```

Next step

[Step 4: Clean up AWS resources](#)

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)
- [Next step](#)

Delete your Managed Service for Apache Flink application

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Managed Service for Apache Flink panel, choose **MyApplication**.
3. In the application's page, choose **Delete** and then confirm the deletion.

Delete your Kinesis data streams

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

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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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](#)

Step 5: Next steps

Now that you've created and run a basic Managed Service for Apache Flink application, see the following resources for more advanced Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink for Apache Flink applications for stream processing.
- [Amazon Managed Service for Apache Flink 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.
- [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 Managed Service for Apache Flink does not support the Apache Flink version (1.12) used in this training. You can use Flink 1.15.2 in Flink Managed Service for Apache Flink.

Getting started: Flink 1.11.1 - deprecating

Note

Apache Flink versions **1.6, 1.8, and 1.11** have not been supported by the Apache Flink community for over three years. We plan to deprecate these versions in Amazon Managed Service for Apache Flink on **November 5, 2024**. Starting from this date, you will not be able to create new applications for these Flink versions. You can continue running existing applications at this time. You can upgrade your applications statefully using the in-place version upgrades feature in Amazon Managed Service for Apache Flink. For more information, see [In-place version upgrades for Apache Flink](#).

This topic contains a version of the [Getting started \(DataStream API\)](#) tutorial that uses Apache Flink 1.11.1.

This section introduces you to the fundamental concepts of Managed Service 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 Managed Service for Apache Flink application](#)
- [Prerequisites for completing the exercises](#)
- [Step 1: Set up an AWS account and create an administrator user](#)
- [Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)
- [Step 3: Create and run a Managed Service for Apache Flink application](#)
- [Step 4: Clean up AWS resources](#)
- [Step 5: Next steps](#)

Components of a Managed Service for Apache Flink application

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

An Managed Service 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](#).
- **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](#).
- **Sink:** The application produces data to external sources by using *sinks*. A sink connector writes data to a Kinesis data stream, a Firehose stream, an Amazon S3 bucket, etc. For more information, see [Sinks](#).

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 Managed Service for Apache Flink 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](#).

Step 1: Set up an AWS account and create an administrator user

Sign up for an AWS account

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

To sign up for an AWS account

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

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

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

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

Create a user with administrative access

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

Secure your AWS account root user

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

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

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

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

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

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

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

Sign in as the user with administrative access

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

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

Assign access to additional users

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

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

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

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. <ul style="list-style-type: none"> For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the <i>AWS Command Line Interface User Guide</i>. For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the <i>AWS SDKs and Tools Reference Guide</i>.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the <i>IAM User Guide</i> .
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. <ul style="list-style-type: none"> For the AWS CLI, see Authenticating using IAM user credentials in the <i>AWS Command Line Interface User Guide</i>. For AWS SDKs and tools, see Authenticate using long-term credentials in

Which user needs programmatic access?	To	By
		<p>the <i>AWS SDKs and Tools Reference Guide</i>.</p> <ul style="list-style-type: none">• For AWS APIs, see Managing access keys for IAM users in the <i>IAM User Guide</i>.

Next step

[Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)

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

In this step, you download and configure the AWS CLI to use with Managed Service 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 Managed Service for Apache Flink application](#)

Step 3: Create and run a Managed Service for Apache Flink application

In this exercise, you create a Managed Service for Apache Flink application with data streams as a source and a sink.

This section contains the following steps:

- [Create two Amazon Kinesis data streams](#)
- [Write sample records to the input stream](#)
- [Download and examine the Apache Flink streaming Java code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Next step](#)

Create two Amazon Kinesis data streams

Before you create a Managed Service for Apache Flink 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-examples.git
```

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 Managed Service 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](#).

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 [Fulfill the prerequisites for completing the exercises](#).

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 Managed Service for Apache Flink application

You can create and run a Managed Service for Apache Flink 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 \(console\)](#)
- [Create and run the application \(AWS CLI\)](#)

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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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 Managed Service 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: `kinesisanalytics-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:

Group ID	Key	Value
ProducerConfigProperties	<code>flink.inputstream.initpos</code>	LATEST
ProducerConfigProperties	<code>aws.region</code>	us-west-2
ProducerConfigProperties	AggregationEnabled	false

- 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, Managed Service for Apache Flink 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 Managed Service for Apache Flink application. A Managed Service for Apache Flink uses the `kinesisanalyticsv2` AWS CLI command to create and interact with Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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. Managed Service for Apache Flink 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 Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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

Managed Service for Apache Flink permission to assume the role, and the permissions policy determines what Managed Service for Apache Flink 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 **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-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, Managed Service for Apache Flink 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"](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the **AKReadSourceStreamWriteSinkStream** 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 Managed Service for Apache Flink 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_11",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-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 Managed Service for Apache Flink 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"](#).

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"](#) 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": "SAMPLEUehYngP87ex1nzYIGYgfhyvDU"
        }
      }
    }
  }
}
```

Next step

[Step 4: Clean up AWS resources](#)

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)
- [Next step](#)

Delete your Managed Service for Apache Flink application

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

2. In the Managed Service for Apache Flink panel, choose **MyApplication**.
3. In the application's page, choose **Delete** and then confirm the deletion.

Delete your Kinesis data streams

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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](#)

Step 5: Next steps

Now that you've created and run a basic Managed Service for Apache Flink application, see the following resources for more advanced Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink for Apache Flink applications for stream processing.
- [Amazon Managed Service for Apache Flink 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.
- [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 Managed Service for Apache Flink does not support the Apache Flink version (1.12) used in this training. You can use Flink 1.15.2 in Flink Managed Service for Apache Flink.

- [Apache Flink Code Examples](#): A GitHub repository of a wide variety of Apache Flink application examples.

Getting started: Flink 1.8.2 - deprecating

Note

Apache Flink versions **1.6**, **1.8**, and **1.11** have not been supported by the Apache Flink community for over three years. We plan to deprecate these versions in Amazon Managed Service for Apache Flink on **November 5, 2024**. Starting from this date, you will not be able to create new applications for these Flink versions. You can continue running existing applications at this time. You can upgrade your applications statefully using the in-place version upgrades feature in Amazon Managed Service for Apache Flink. For more information, see [In-place version upgrades for Apache Flink](#).

This topic contains a version of the [Getting started \(DataStream API\)](#) tutorial that uses Apache Flink 1.8.2.

Topics

- [Components of Managed Service for Apache Flink application](#)
- [Prerequisites for completing the exercises](#)
- [Step 1: Set up an AWS account and create an administrator user](#)
- [Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)
- [Step 3: Create and run a Managed Service for Apache Flink application](#)
- [Step 4: Clean up AWS resources](#)

Components of Managed Service for Apache Flink application

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

An Managed Service 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](#).
- **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](#).
- **Sink:** The application produces data to external sources by using *sinks*. A sink connector writes data to a Kinesis data stream, a Firehose stream, an Amazon S3 bucket, etc. For more information, see [Sinks](#).

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 Managed Service for Apache Flink 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](#).
- 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](#).

Step 1: Set up an AWS account and create an administrator user

Sign up for an AWS account

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

To sign up for an AWS account

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

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

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

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

Create a user with administrative access

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

Secure your AWS account root user

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

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

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

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

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

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

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

Sign in as the user with administrative access

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

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

Assign access to additional users

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

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

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

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. <ul style="list-style-type: none"> For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the <i>AWS Command Line Interface User Guide</i>. For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the <i>AWS SDKs and Tools Reference Guide</i>.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the <i>IAM User Guide</i> .
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. <ul style="list-style-type: none"> For the AWS CLI, see Authenticating using IAM user credentials in the <i>AWS</i>

Which user needs programmatic access?	To	By
		<p><i>Command Line Interface User Guide.</i></p> <ul style="list-style-type: none">• For AWS SDKs and tools, see Authenticate using long-term credentials in the <i>AWS SDKs and Tools Reference Guide</i>.• For AWS APIs, see Managing access keys for IAM users in the <i>IAM User Guide</i>.

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

In this step, you download and configure the AWS CLI to use with Managed Service 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 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 Managed Service for Apache Flink application](#)

Step 3: Create and run a Managed Service for Apache Flink application

In this exercise, you create a Managed Service for Apache Flink application with data streams as a source and a sink.

This section contains the following steps:

- [Create two Amazon Kinesis data streams](#)
- [Write sample records to the input stream](#)
- [Download and examine the Apache Flink streaming Java code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Next step](#)

Create two Amazon Kinesis data streams

Before you create a Managed Service for Apache Flink 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 kineses 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, kineses_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-examples.git
```

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 Managed Service 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](#).

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](#).

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”](#).

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 Managed Service for Apache Flink application

You can create and run a Managed Service for Apache Flink 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 \(console\)](#)
- [Create and run the application \(AWS CLI\)](#)

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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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 Managed Service 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: `kinesisanalytics-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. Enter the following application properties and values:

Group ID	Key	Value
ProducerConfigProperties	flink.inputstream.initpos	LATEST
ProducerConfigProperties	aws.region	us-west-2
ProducerConfigProperties	AggregationEnabled	false

5. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
6. For **CloudWatch logging**, select the **Enable** check box.
7. Choose **Update**.

Note

When you choose to enable Amazon CloudWatch logging, Managed Service for Apache Flink 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 Managed Service for Apache Flink application. Managed Service for Apache Flink uses the `kinesisanalyticsv2` AWS CLI command to create and interact with Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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. Managed Service for Apache Flink 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 Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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 Managed Service for Apache Flink permission to assume the role, and the permissions policy determines what Managed Service for Apache Flink 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 **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-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, Managed Service for Apache Flink 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"](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the **AKReadSourceStreamWriteSinkStream** 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 Managed Service for Apache Flink 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/MF-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 Managed Service for Apache Flink 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”](#).

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"](#) 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](#)**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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Managed Service for Apache Flink 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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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 - deprecating

Note

Apache Flink versions **1.6**, **1.8**, and **1.11** have not been supported by the Apache Flink community for over three years. We plan to deprecate these versions in Amazon Managed Service for Apache Flink on **November 5, 2024**. Starting from this date, you will not be able to create new applications for these Flink versions. You can continue running existing applications at this time. You can upgrade your applications statefully using the in-place version upgrades feature in Amazon Managed Service for Apache Flink For more information, see [In-place version upgrades for Apache Flink](#).

This topic contains a version of the [Getting started \(DataStream API\)](#) tutorial that uses Apache Flink 1.6.2.

Topics

- [Components of a Managed Service for Apache Flink application](#)
- [Prerequisites for completing the exercises](#)

- [Step 1: Set up an AWS account and create an administrator user](#)
- [Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)
- [Step 3: Create and run a Managed Service for Apache Flink application](#)
- [Step 4: Clean up AWS resources](#)

Components of a Managed Service for Apache Flink application

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

a Managed Service for Apache Flink 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](#).
- **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](#).
- **Sink:** The application produces data to external sources by using *sinks*. A sink connector writes data to a Kinesis data stream, a Firehose stream, an Amazon S3 bucket, etc. For more information, see [Sinks](#).

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 Managed Service for Apache Flink 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](#).

Step 1: Set up an AWS account and create an administrator user

Sign up for an AWS account

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

To sign up for an AWS account

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

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

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

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

Create a user with administrative access

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

Secure your AWS account root user

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

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

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

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

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

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

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

Sign in as the user with administrative access

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

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

Assign access to additional users

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

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

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

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	<p>Following the instructions for the interface that you want to use.</p> <ul style="list-style-type: none"> For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the <i>AWS Command Line Interface User Guide</i>. For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the <i>AWS SDKs and Tools Reference Guide</i>.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the <i>IAM User Guide</i> .

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. <ul style="list-style-type: none">• For the AWS CLI, see Authenticating using IAM user credentials in the <i>AWS Command Line Interface User Guide</i>.• For AWS SDKs and tools, see Authenticate using long-term credentials in the <i>AWS SDKs and Tools Reference Guide</i>.• For AWS APIs, see Managing access keys for IAM users in the <i>IAM User Guide</i>.

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

In this step, you download and configure the AWS CLI to use with a Managed Service 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 Managed Service for Apache Flink application](#)

Step 3: Create and run a Managed Service for Apache Flink application

In this exercise, you create a Managed Service for Apache Flink application with data streams as a source and a sink.

This section contains the following steps:

- [Create two Amazon Kinesis data streams](#)
- [Write sample records to the input stream](#)
- [Download and examine the Apache Flink streaming Java code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)

Create two Amazon Kinesis data streams

Before you create a Managed Service for Apache Flink 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-examples.git
```

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 a Managed Service 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](#).

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](#).

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 Managed Service for Apache Flink 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”](#).

- 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 Managed Service for Apache Flink application

You can create and run a Managed Service for Apache Flink 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 \(console\)](#)
- [Create and run the application \(AWS CLI\)](#)

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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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

Managed Service for Apache Flink 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 Managed Service 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: `kinesisanalytics-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. Enter the following application properties and values:

Group ID	Key	Value
ProducerConfigProperties	flink.inputstream.initpos	LATEST
ProducerConfigProperties	aws.region	us-west-2
ProducerConfigProperties	AggregationEnabled	false

5. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
6. For **CloudWatch logging**, select the **Enable** check box.
7. Choose **Update**.

Note

When you choose to enable Amazon CloudWatch logging, Managed Service for Apache Flink 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 Managed Service for Apache Flink application. Managed Service for Apache Flink uses the `kinesisanalyticsv2` AWS CLI command to create and interact with Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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. Managed Service for Apache Flink 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 Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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 Managed Service for Apache Flink permission to assume the role, and the permissions policy determines what Managed Service for Apache Flink 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 **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-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, Managed Service for Apache Flink 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"](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the **AKReadSourceStreamWriteSinkStream** 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 Managed Service for Apache Flink 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/MF-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 Managed Service for Apache Flink 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”](#).

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.

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"](#) 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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Managed Service for Apache Flink 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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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.

Earlier version (legacy) examples for Managed Service for Apache Flink

Note

For current examples, see [Examples](#).

This section provides examples of creating and working with applications in Managed Service for Apache Flink. They include example code and step-by-step instructions to help you create Managed Service for Apache Flink applications and test your results.

Before you explore these examples, we recommend that you first review the following:

- [How it works](#)
- [Getting started \(DataStream API\)](#)

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](#)
- [Python examples](#)
- [Scala examples](#)

DataStream API examples

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

Topics

- [Example: Tumbling window](#)
- [Example: Sliding window](#)
- [Example: Writing to an Amazon S3 bucket](#)
- [Tutorial: Using a Managed Service for Apache Flink application to replicate data from one topic in an MSK cluster to another in a VPC](#)
- [Example: Use an EFO consumer with a Kinesis data stream](#)
- [Example: Writing to Firehose](#)
- [Example: Read from a Kinesis stream in a different account](#)
- [Tutorial: Using a custom truststore with Amazon MSK](#)

Example: Tumbling window

Note

For current examples, see [Examples](#).

In this exercise, you create a Managed Service for Apache Flink application that aggregates data using a tumbling window. Aggregation is enabled by default in Flink. To disable it, use the following:

```
sink.producer.aggregation-enabled' = 'false'
```

Note

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

This topic contains the following sections:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the application code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Clean up AWS resources](#)

Create dependent resources

Before you create a Managed Service for Apache Flink 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', region_name='us-west-2'))
```

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-examples.git
```

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 onward
```

- 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  
  
        .window(TumblingProcessingTimeWindows.of(Time.seconds(5))) //  
Flink 1.13 onward  
  
        .sum(1) // Sum the number of words per partition  
        .map(value -> value.f0 + "," + value.f1.toString() + "\n")  
        .addSink(createSinkFromStaticConfig());
```

Compile the application code

To compile the application, do the following:

1. Install Java and Maven if you haven't already. For more information, see [Prerequisites](#) in the [Getting started \(DataStream API\)](#) tutorial.
2. Compile the application with the following command:

```
mvn package -Dflink.version=1.15.3
```

 **Note**

The provided source code relies on libraries from Java 11.

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](#) 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 Managed Service for Apache Flink application

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

Create the application


1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.

- For **Runtime**, choose **Apache Flink**.

 **Note**

Managed Service for Apache Flink uses Apache Flink version 1.15.2.

- Leave the version pulldown as **Apache Flink version 1.15.2 (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 Managed Service 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: `kinesisanalytics-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, Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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

Note

For current examples, see [Examples](#).

Note

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

This topic contains the following sections:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the application code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Clean up AWS resources](#)

Create dependent resources

Before you create a Managed Service for Apache Flink 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-examples.git
```

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 onward
```

- 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

      .window(TumblingProcessingTimeWindows.of(Time.seconds(5))) //Flink 1.13 onward
      .sum(1) // Sum the number of words per partition
      .map(value -> value.f0 + "," + value.f1.toString() + "\n")
      .addSink(createSinkFromStaticConfig());
```

Compile the application code

To compile the application, do the following:

1. Install Java and Maven if you haven't already. For more information, see [Prerequisites](#) in the [Getting started \(DataStream API\)](#) tutorial.
2. Compile the application with the following command:

```
mvn package -Dflink.version=1.15.3
```

Note

The provided source code relies on libraries from Java 11.

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](#) 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 Managed Service for Apache Flink application

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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 1.15.2 (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 Managed Service 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: `kinesisanalytics-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, Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. In the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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 Managed Service for Apache Flink 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 \(DataStream API\)](#) exercise.

This topic contains the following sections:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the application code](#)

- [Modify the application code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Verify the application output](#)
- [Optional: Customize the source and sink](#)
- [Clean up AWS resources](#)

Create dependent resources

Before you create a Managed Service for Apache Flink 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

Managed Service for Apache Flink cannot write data to Amazon S3 with server-side encryption enabled on Managed Service for Apache Flink.

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>***. 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/KinesisAnalytics-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', region_name='us-west-2'))
```

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-examples.git
```

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(v -> v.f0) // Logically partition the stream for each word
    .window(TumblingProcessingTimeWindows.of(Time.minutes(1)))
    .sum(1) // Count the appearances by ticker per partition
    .map(value -> value.f0 + " count: " + value.f1.toString() + "\n")
    .addSink(createS3SinkFromStaticConfig());
```

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-code-<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](#) in the [Getting started \(DataStream API\)](#) tutorial.
2. Compile the application with the following command:

```
mvn package -Dflink.version=1.15.3
```

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.

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](#) section.

1. In the Amazon S3 console, choose the **ka-app-code-*<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 Managed Service for Apache Flink application

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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 1.15.2 (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 Managed Service 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:

- For **Application name**, enter **MyApplication**.
- For **Runtime**, choose **Apache Flink**.
- Leave the version as **Apache Flink version 1.15.2 (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 Managed Service for Apache Flink 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: `kinesisanalytics-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": "S3",
  "Effect": "Allow",
  "Action": [
    "s3:Abort*",
    "s3:DeleteObject*",
    "s3:GetObject*",
    "s3:GetBucket*",
    "s3:List*",
    "s3:ListBucket",
    "s3:PutObject"
  ],
  "Resource": [
    "arn:aws:s3:::ka-app-code-<username>",
    "arn:aws:s3:::ka-app-code-<username>/*"
  ]
},
{
  "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"
  },
]
}
```

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 **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, Managed Service for Apache Flink 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.

Note

Aggregation is enabled by default in Flink. To disable it, use the following:

```
sink.producer.aggregation-enabled' = 'false'
```

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”](#) section to compile the updated application code.
- Repeat the steps in the [the section called “Upload the Apache Flink streaming Java code”](#) 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](#)
- [Configure read frequency](#)
- [Configure write buffering](#)

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.DefaultRollingPol
import
  org.apache.flink.streaming.api.functions.sink.filesystem.bucketassigners.DateTimeBucketAss
```

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data stream](#)
- [Delete your Amazon S3 objects and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. In the Managed Service for Apache Flink 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-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. On the navigation bar, choose **Policies**.
3. In the filter control, enter **kinesis**.
4. Choose the **kinesis-analytics-service-MyApplication-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. On the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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 Managed Service for Apache Flink application to replicate data from one topic in an MSK cluster to another in a VPC

Note

For current examples, see [Examples](#).

The following tutorial demonstrates how to create an Amazon VPC with an Amazon MSK cluster and two topics, and how to create a Managed Service for Apache Flink 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 \(DataStream API\)](#) exercise.

This tutorial contains the following sections:

- [Create an Amazon VPC with an Amazon MSK cluster](#)
- [Create the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create the application](#)
- [Configure the application](#)
- [Run the application](#)
- [Test the application](#)

Create an Amazon VPC with an Amazon MSK cluster

To create a sample VPC and Amazon MSK cluster to access from a Managed Service for Apache Flink application, follow the [Getting Started Using Amazon MSK](#) tutorial.

When completing the tutorial, note the following:

- In [Step 3: 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-examples.git
```

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 Managed Service 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=1.15.3
```

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 \(DataStream API\)](#) 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"](#) 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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>.
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.
 - For **Runtime**, choose **Apache Flink version 1.15.2**.
4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
5. Choose **Create application**.

Note

When you create a Managed Service 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: `kinesisanalytics-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**. Enter the following properties:

Group ID	Key	Value
KafkaSource	topic	AWSKafkaTutorialTopic
KafkaSource	bootstrap.servers	<i>The bootstrap server list you saved previously</i>
KafkaSource	security.protocol	SSL
KafkaSource	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSource	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. Enter the following properties:

Group ID	Key	Value
KafkaSink	topic	AWSKafkaTutorialTopicDestination
KafkaSink	bootstrap.servers	<i>The bootstrap server list you saved previously</i>
KafkaSink	security.protocol	SSL
KafkaSink	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSink	ssl.truststore.password	changeit
KafkaSink	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](#).

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, Managed Service for Apache Flink 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](#) section in the [Troubleshooting](#) topic.

Example: Use an EFO consumer with a Kinesis data stream

Note

For current examples, see [Examples](#).

In this exercise, you create a Managed Service for Apache Flink 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) 1.15.3.

Note

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

This topic contains the following sections:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the application code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Clean up AWS resources](#)

Create dependent resources

Before you create a Managed Service for Apache Flink 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', region_name='us-west-2'))
```

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-examples.git
```

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

```
mvn package -Dflink.version=1.15.3
```

Note

The provided source code relies on libraries from Java 11.

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](#) 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 Managed Service for Apache Flink application

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.
 - For **Runtime**, choose **Apache Flink**.

Note

Managed Service for Apache Flink uses Apache Flink version 1.15.2.

- Leave the version pulldown as **Apache Flink version 1.15.2 (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 Managed Service 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: `kinesisanalytics-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-efo-flink-app",
            "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream/
consumer/my-efo-flink-app:*"
        ]
    }
]
}

```

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**.
5. Enter the following application properties and values:

Group ID	Key	Value
ConsumerConfigProperties	flink.stream.recorderpublisher	EF0

Group ID	Key	Value
ConsumerConfigProperties	flink.stream.efs.consumername	basic-efs-flink-app
ConsumerConfigProperties	INPUT_STREAM	ExampleInputStream
ConsumerConfigProperties	flink.inputstream.initpos	LATEST
ConsumerConfigProperties	AWS_REGION	us-west-2

- Under **Properties**, choose **Create Group**.
- Enter the following application properties and values:

Group ID	Key	Value
ProducerConfigProperties	OUTPUT_STREAM	ExampleOutputStream
ProducerConfigProperties	AWS_REGION	us-west-2
ProducerConfigProperties	AggregationEnabled	false

- 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, Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 (*basic-efo-flink-app*).

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete Your Amazon S3 Object and Bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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 Firehose

Note

For current examples, see [Examples](#).

In this exercise, you create a Managed Service for Apache Flink application that has a Kinesis data stream as a source and a Firehose 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 \(DataStream API\)](#) exercise.

This section contains the following steps:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the Apache Flink streaming Java code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Clean up AWS resources](#)

Create dependent resources

Before you create a Managed Service for Apache Flink for this exercise, you create the following dependent resources:

- A Kinesis data stream (ExampleInputStream)
- A Firehose 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 Firehose 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 Data Firehose Developer Guide*. Name your Firehose stream **ExampleDeliveryStream**. When you create the Firehose stream, also create the Firehose 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', region_name='us-west-2'))
```

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-examples.git
```

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 Firehose sink to write data to a Firehose stream. The following snippet creates the Firehose sink:

```
private static KinesisFirehoseSink<String> createFirehoseSinkFromStaticConfig() {
    Properties sinkProperties = new Properties();
    sinkProperties.setProperty(AWS_REGION, region);

    return KinesisFirehoseSink.<String>builder()
        .setFirehoseClientProperties(sinkProperties)
        .setSerializationSchema(new SimpleStringSchema())
        .setDeliveryStreamName(outputDeliveryStreamName)
        .build();
}
```

Compile the application code

To compile the application, do the following:

1. Install Java and Maven if you haven't already. For more information, see [Prerequisites](#) in the [Getting started \(DataStream API\)](#) 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”](#).**
3. Compile the application with the following command:

```
mvn package -Dflink.version=1.15.3
```

Note

The provided source code relies on libraries from Java 11.

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](#) 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 Managed Service for Apache Flink application

You can create and run a Managed Service for Apache Flink 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 \(console\)](#)
- [Create and run the application \(AWS CLI\)](#)

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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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

Managed Service for Apache Flink uses Apache Flink version 1.15.2.

- Leave the version pulldown as **Apache Flink version 1.15.2 (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: `kinesisanalytics-MyApplication-us-west-2`

Edit the IAM policy

Edit the IAM policy to add permissions to access the Kinesis data stream and Firehose 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": [  
    {  
      "Effect": "Allow",  
      "Action": "kinesis:*",  
      "Resource": "arn:aws:kinesis:012345678901:us-west-2:stream/*"  
    },  
    {  
      "Effect": "Allow",  
      "Action": "firehose:*",  
      "Resource": "arn:aws:firehose:012345678901:us-west-2:deliverystream/*"  
    }  
  ]  
}
```

```

    {
      "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, Managed Service for Apache Flink 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"](#) 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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. Managed Service for Apache Flink 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 Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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 Managed Service for Apache Flink permission to assume the role. The permissions policy determines what Managed Service for Apache Flink 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 **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-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, Managed Service for Apache Flink 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"](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the **AKReadSourceStreamWriteSinkStream** 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 Managed Service for Apache Flink 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"](#) 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_15",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-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 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 Managed Service for Apache Flink 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”](#).

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”](#) 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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data stream](#)
- [Delete your Firehose stream](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. In the Managed Service for Apache Flink 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 Firehose stream

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Firehose panel, choose **ExampleDeliveryStream**.
3. In the **ExampleDeliveryStream** page, choose **Delete Firehose 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 Firehose 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. If you created a new policy for your Firehose stream, delete that policy too.
7. In the navigation bar, choose **Roles**.
8. Choose the **kinesis-analytics-MyApplication-us-west-2** role.
9. Choose **Delete role** and then confirm the deletion.
10. If you created a new role for your Firehose 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

Note

For current examples, see [Examples](#).

This example demonstrates how to create an Managed Service for Apache Flink 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 Managed Service for Apache Flink application and sink Kinesis stream.

This topic contains the following sections:

- [Prerequisites](#)
- [Setup](#)
- [Create source Kinesis stream](#)
- [Create and update IAM roles and policies](#)
- [Update the Python script](#)
- [Update the Java application](#)
- [Build, upload, and run the application](#)

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 \(DataStream API\)](#) tutorial before proceeding.
- You need two AWS accounts to complete this tutorial: one for the source stream, and one for the application and the sink stream. Use the AWS account you used for the Getting Started tutorial for the application and sink stream. Use a different AWS account for the source stream.

Setup

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

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

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

[ka-sink-stream-account-profile]
aws_access_key_id=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 kinesys 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 `kinesisanalytics-<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 `MF-Source-Stream-Role`. Do the following to create the role using the **Managed Flink** 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 **Managed Service for Apache Flink**.
 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.managed-flink;

import com.amazonaws.services.managed-flink.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 Managed Service for Apache Flink 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 KinesisStreamsSink<String> createSinkFromStaticConfig() {
        Properties outputProperties = new Properties();
        outputProperties.setProperty(AWSConfigConstants.AWS_REGION, region);
    }
}
```

```
        return KinesisStreamsSink.<String>builder()
            .setKinesisClientProperties(outputProperties)
            .setSerializationSchema(new SimpleStringSchema())
            .setStreamName(outputProperties.getProperty("OUTPUT_STREAM",
"ExampleOutputStream"))
            .setPartitionKeyGenerator(element ->
String.valueOf(element.hashCode()))
            .build();
    }

    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=1.15.3
```

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 Managed Service for Apache Flink 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

Note

For current examples, see [Examples](#).

Current data source APIs

If you are using the current data source APIs, your application can leverage the Amazon MSK Config Providers utility described [here](#). This allows your `KafkaSource` function to access your keystore and truststore for mutual TLS in Amazon S3.

```
...
// define names of config providers:
builder.setProperty("config.providers", "secretsmanager,s3import");

// provide implementation classes for each provider:
builder.setProperty("config.providers.secretsmanager.class",
    "com.amazonaws.kafka.config.providers.SecretsManagerConfigProvider");
builder.setProperty("config.providers.s3import.class",
    "com.amazonaws.kafka.config.providers.S3ImportConfigProvider");

String region = appProperties.get(Helpers.S3_BUCKET_REGION_KEY).toString();
String keystoreS3Bucket = appProperties.get(Helpers.KEYSTORE_S3_BUCKET_KEY).toString();
String keystoreS3Path = appProperties.get(Helpers.KEYSTORE_S3_PATH_KEY).toString();
String truststoreS3Bucket =
    appProperties.get(Helpers.TRUSTSTORE_S3_BUCKET_KEY).toString();
String truststoreS3Path = appProperties.get(Helpers.TRUSTSTORE_S3_PATH_KEY).toString();
String keystorePassSecret =
    appProperties.get(Helpers.KEYSTORE_PASS_SECRET_KEY).toString();
String keystorePassSecretField =
    appProperties.get(Helpers.KEYSTORE_PASS_SECRET_FIELD_KEY).toString();

// region, etc..
builder.setProperty("config.providers.s3import.param.region", region);
```

```
// properties
builder.setProperty("ssl.truststore.location", "${s3import:" + region + ":" +
  truststoreS3Bucket + "/" + truststoreS3Path + "}");
builder.setProperty("ssl.keystore.type", "PKCS12");
builder.setProperty("ssl.keystore.location", "${s3import:" + region + ":" +
  keystoreS3Bucket + "/" + keystoreS3Path + "}");
builder.setProperty("ssl.keystore.password", "${secretsmanager:" + keystorePassSecret +
  ":" + keystorePassSecretField + "}");
builder.setProperty("ssl.key.password", "${secretsmanager:" + keystorePassSecret + ":" +
  keystorePassSecretField + "}");
...
```

More details and a walkthrough can be found [here](#).

Legacy SourceFunction APIs

If you are using the legacy SourceFunction APIs, your application will use custom serialization and deserialization schemas that override 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:

```
public static void initializeKafkaTruststore() {
    ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
    URL inputUrl = classLoader.getResource("kafka.client.truststore.jks");
    File dest = new File("/tmp/kafka.client.truststore.jks");

    try {
        FileUtils.copyURLToFile(inputUrl, dest);
    } catch (Exception ex) {
        throw new FlinkRuntimeException("Failed to initialize Kafka truststore", ex);
    }
}
```

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 \(DataStream API\)](#) exercise.

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 Managed Service for Apache Flink 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

This tutorial contains the following sections:

- [Create a VPC with an Amazon MSK cluster](#)

- [Create a custom truststore and apply it to your cluster](#)
- [Create the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create the application](#)
- [Configure the application](#)
- [Run the application](#)
- [Test the application](#)

Create a VPC with an Amazon MSK cluster

To create a sample VPC and Amazon MSK cluster to access from a Managed Service for Apache Flink application, follow the [Getting Started Using Amazon MSK](#) tutorial.

When completing the tutorial, also do the following:

- In [Step 3: Create a Topic](#), repeat the `kafka-topics.sh --create` command to create a destination topic named `AWSKafkaTutorialTopicDestination`:

```
bin/kafka-topics.sh --create --bootstrap-server 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-examples.git
```

3. The application code is located in the `amazon-kinesis-data-analytics-java-examples/CustomKeystore`. You can examine the code to familiarize yourself with the structure of Managed Service for Apache Flink 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=1.15.3
```

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.

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 \(DataStream API\)](#) 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"](#) 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 `flink-app-1.0-SNAPSHOT.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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.
 - For **Runtime**, choose **Apache Flink version 1.15.2**.
4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
5. Choose **Create application**.

Note

When you create a Managed Service for Apache Flink 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: `kinesisanalytics-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**. Enter the following properties:

Group ID	Key	Value
KafkaSource	topic	AWSKafkaTutorialTopic
KafkaSource	bootstrap.servers	<i>The bootstrap server list you saved previously</i>
KafkaSource	security.protocol	SSL
KafkaSource	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSource	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. Enter the following properties:

Group ID	Key	Value
KafkaSink	topic	AWSKafkaTutorialTopicDestination
KafkaSink	bootstrap.servers	<i>The bootstrap server list you saved previously</i>
KafkaSink	security.protocol	SSL
KafkaSink	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSink	ssl.truststore.password	changeit
KafkaSink	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](#).

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, Managed Service for Apache Flink 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](#) section in the [Troubleshooting](#) topic.

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](#)
- [Example: Creating a sliding window in Python](#)
- [Example: Send streaming data to Amazon S3 in Python](#)

Example: Creating a tumbling window in Python

Note

For current examples, see [Examples](#).

In this exercise, you create a Python Managed Service for Apache Flink application that aggregates data using a tumbling window.

Note

To set up required prerequisites for this exercise, first complete the [Getting started \(Python\)](#) exercise.

This topic contains the following sections:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the application code](#)
- [Compress and upload the Apache Flink streaming Python code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Clean up AWS resources](#)

Create dependent resources

Before you create a Managed Service for Apache Flink 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', region_name='us-west-2'))
```

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-examples.git
```

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_input_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_input_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}',  
        '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.min as price, to_string(ten_second_window.end) as  
    event_time")  
)
```

- The application uses the Kinesis Flink connector, from the [flink-sql-connector-kinesis-1.15.2.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](#) section.

1. Use your preferred compression application to compress the `tumbling-windows.py` and `flink-sql-connector-kinesis-1.15.2.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 Managed Service for Apache Flink application

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.
 - For **Runtime**, choose **Apache Flink**.

Note

Managed Service for Apache Flink uses Apache Flink version 1.15.2.

- Leave the version pulldown as **Apache Flink version 1.15.2 (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 Managed Service 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: `kinesisanalytics-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**.
5. Enter the following:

Group ID	Key	Value
<code>consumer.config.0</code>	<code>input.stream.name</code>	<code>ExampleInputStream</code>

Group ID	Key	Value
<code>consumer.config.0</code>	<code>aws.region</code>	<code>us-west-2</code>
<code>consumer.config.0</code>	<code>scan.stream.initpos</code>	<code>LATEST</code>

Choose **Save**.

- Under **Properties**, choose **Add group** again.
- Enter the following:

Group ID	Key	Value
<code>producer.config.0</code>	<code>output.stream.name</code>	<code>ExampleOutputStream</code>
<code>producer.config.0</code>	<code>aws.region</code>	<code>us-west-2</code>
<code>producer.config.0</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](#).
- Enter the following:

Group ID	Key	Value
<code>kinesis.analytics.flink.run.options</code>	<code>python</code>	<code>tumbling-windows.py</code>
<code>kinesis.analytics.flink.run.options</code>	<code>jarfile</code>	<code>flink-sql-connector-kinesis-1.15.2.jar</code>

- 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, Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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

Note

For current examples, see [Examples](#).

Note

To set up required prerequisites for this exercise, first complete the [Getting started \(Python\)](#) exercise.

This topic contains the following sections:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the application code](#)
- [Compress and upload the Apache Flink streaming Python code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Clean up AWS resources](#)

Create dependent resources

Before you create a Managed Service for Apache Flink 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', region_name='us-west-2'))
```

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_input_table` function to create the Kinesis table source:

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

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

```
def create_input_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}',
    '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, to_string(ten_second_window.end) as
event_time")
)

```

- The application uses the Kinesis Flink connector, from the [flink-sql-connector-kinesis-1.15.2.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](#) section.

This section describes how to package your Python application.

1. Use your preferred compression application to compress the `sliding-windows.py` and `flink-sql-connector-kinesis-1.15.2.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 Managed Service for Apache Flink application

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.
 - For **Runtime**, choose **Apache Flink**.

Note

Managed Service for Apache Flink uses Apache Flink version 1.15.2.

- Leave the version pulldown as **Apache Flink version 1.15.2 (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 Managed Service 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: `kinesisanalytics-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**.
5. Enter the following application properties and values:

Group ID	Key	Value
<code>consumer.config.0</code>	<code>input.stream.name</code>	<code>ExampleInputStream</code>
<code>consumer.config.0</code>	<code>aws.region</code>	<code>us-west-2</code>
<code>consumer.config.0</code>	<code>scan.stream.initpos</code>	<code>LATEST</code>

Choose **Save**.

6. Under **Properties**, choose **Add group** again.
7. Enter the following application properties and values:

Group ID	Key	Value
<code>producer.config.0</code>	<code>output.stream.name</code>	<code>ExampleOutputStream</code>

Group ID	Key	Value
<code>producer.config.0</code>	<code>aws.region</code>	<code>us-west-2</code>
<code>producer.config.0</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](#).
- Enter the following application properties and values:

Group ID	Key	Value
<code>kinesis.analytics.flink.run.options</code>	<code>python</code>	<code>sliding-windows.py</code>
<code>kinesis.analytics.flink.run.options</code>	<code>jarfile</code>	<code>flink-sql-connector-kinesis_1.15.2.jar</code>

- 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, Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)

- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.

7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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

Note

For current examples, see [Examples](#).

In this exercise, you create a Python Managed Service for Apache Flink 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 \(Python\)](#) exercise.

This topic contains the following sections:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the application code](#)
- [Compress and upload the Apache Flink streaming Python code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Clean up AWS resources](#)

Create dependent resources

Before you create a Managed Service 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-code-<username>`)

Note

Managed Service for Apache Flink cannot write data to Amazon S3 with server-side encryption enabled on Managed Service for Apache Flink.

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', region_name='us-west-2'))
```

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-examples.git
```

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_source_table` function to create the Kinesis table source:

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

The `create_source_table` function uses a SQL command to create a table that is backed by the streaming source

```
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', region_name='us-west-2'))
```

- 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 VARCHAR(64)
    )
    PARTITIONED BY (ticker)
    WITH (
        'connector'='filesystem',
        'path'='s3a://{1}/',
        'format'='json',
        '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 [flink-sql-connector-kinesis-1.15.2.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](#) section.

1. Use your preferred compression application to compress the `streaming-file-sink.py` and [flink-sql-connector-kinesis-1.15.2.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 Managed Service for Apache Flink application

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.
 - For **Runtime**, choose **Apache Flink**.

Note

Managed Service for Apache Flink uses Apache Flink version 1.15.2.

- Leave the version pulldown as **Apache Flink version 1.15.2 (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 Managed Service 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: `kinesisanalytics-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**.
5. Enter the following application properties and values:

Group ID	Key	Value
<code>consumer.config.0</code>	<code>input.stream.name</code>	<code>ExampleInputStream</code>
<code>consumer.config.0</code>	<code>aws.region</code>	<code>us-west-2</code>
<code>consumer.config.0</code>	<code>scan.stream.initpos</code>	<code>LATEST</code>

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](#).
7. Enter the following application properties and values:

Group ID	Key	Value
<code>kinesis.analytics.flink.run.options</code>	<code>python</code>	<code>streaming-file-sink.py</code>
<code>kinesis.analytics.flink.run.options</code>	<code>jarfile</code>	<code>S3Sink/lib/flink-sql-connector-kinesis-1.15.2.jar</code>

- 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](#).
- Enter the following application properties and values: (replace *bucket-name* with the actual name of your Amazon S3 bucket.)

Group ID	Key	Value
<code>sink.config.0</code>	<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, Managed Service for Apache Flink 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": "WriteObjects",
    "Effect": "Allow",
    "Action": [
        "s3:Abort*",
        "s3:DeleteObject*",
        "s3:GetObject*",
        "s3:GetBucket*",
        "s3:List*",
        "s3:ListBucket",
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::ka-app-code-<username>",
        "arn:aws:s3:::ka-app-code-<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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data stream](#)
- [Delete your Amazon S3 objects and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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.

Scala examples

The following examples demonstrate how to create applications using Scala with Apache Flink.

Topics

- [Example: Creating a tumbling window in Scala](#)
- [Example: Creating a sliding window in Scala](#)
- [Example: Send streaming data to Amazon S3 in Scala](#)

Example: Creating a tumbling window in Scala

Note

For current examples, see [Examples](#).

Note

Starting from version 1.15 Flink is Scala free. Applications can now use the Java API from any Scala version. Flink still uses Scala in a few key components internally but doesn't expose Scala into the user code classloader. Because of that, users need to add Scala dependencies into their jar-archives.

For more information about Scala changes in Flink 1.15, see [Scala Free in One Fifteen](#).

In this exercise, you will create a simple streaming application which uses Scala 3.2.0 and Flink's Java DataStream API. The application reads data from Kinesis stream, aggregates it using sliding windows and writes results to output Kinesis stream.

Note

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

This topic contains the following sections:

- [Download and examine the application code](#)
- [Compile and upload the application code](#)
- [Create and run the application \(console\)](#)
- [Create and run the application \(CLI\)](#)
- [Update the application code](#)
- [Clean up AWS resources](#)

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-examples.git
```

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

Note the following about the application code:

- A `build.sbt` file contains information about the application's configuration and dependencies, including the Managed Service for Apache Flink libraries.
- The `BasicStreamingJob.scala` 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:

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
    defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
}
```

The application also uses a Kinesis sink to write into the result stream. The following snippet creates the Kinesis sink:

```
private def createSink: KinesisStreamsSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val outputProperties = applicationProperties.get("ProducerConfigProperties")

  KinesisStreamsSink.builder[String]
    .setKinesisClientProperties(outputProperties)
    .setSerializationSchema(new SimpleStringSchema)
    .setStreamName(outputProperties.getProperty(streamNameKey,
    defaultOutputStreamName))
    .setPartitionKeyGenerator((element: String) => String.valueOf(element.hashCode))
    .build
}
```

- The application uses the window 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:

```
environment.addSource(createSource)
  .map { value =>
    val jsonNode = jsonParser.readValue(value, classOf[JsonNode])
    new Tuple2[String, Int](jsonNode.get("ticker").toString, 1)
  }
  .returns(Types.TUPLE(Types.STRING, Types.INT))
  .keyBy(v => v.f0) // Logically partition the stream for each ticker
  .window(TumblingProcessingTimeWindows.of(Time.seconds(10)))
  .sum(1) // Sum the number of tickers per partition
  .map { value => value.f0 + "," + value.f1.toString + "\n" }
  .sinkTo(createSink)
```

- The application creates source and sink connectors to access external resources using a `StreamExecutionEnvironment` object.
- The application creates source and sink connectors using dynamic application properties. Runtime application's properties are read to configure the connectors. For more information about runtime properties, see [Runtime Properties](#).

Compile and upload the application code

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

Compile the Application Code

Use the [SBT](#) build tool to build the Scala code for the application. To install SBT, see [Install sbt with cs setup](#). You also need to install the Java Development Kit (JDK). See [Prerequisites for Completing the Exercises](#).

1. To use your application code, you compile and package it into a JAR file. You can compile and package your code with SBT:

```
sbt assembly
```

2. If the application compiles successfully, the following file is created:

```
target/scala-3.2.0/tumbling-window-scala-1.0.jar
```

Upload the Apache Flink Streaming Scala Code

In this section, you create an Amazon S3 bucket and upload your 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 **Configure options**, keep the settings as they are, and choose **Next**.
5. In **Set permissions**, keep the settings as they are, and choose **Next**.
6. Choose **Create bucket**.
7. Choose the `ka-app-code-<username>` bucket, and then choose **Upload**.
8. In the **Select files** step, choose **Add files**. Navigate to the `tumbling-window-scala-1.0.jar` file that you created in the previous step.
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 application (console)

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.
 - For **Description**, enter **My Scala test app**.
 - For **Runtime**, choose **Apache Flink**.
 - Leave the version as **Apache Flink version 1.15.2 (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 Managed Service 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: `kinesisanalytics-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 `tumbling-window-scala-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 **Add group**.
5. Enter the following:

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

Choose **Save**.

6. Under **Properties**, choose **Add group** again.
7. Enter the following:

Group ID	Key	Value
ProducerConfigProperties	output.stream.name	ExampleOutputStream
ProducerConfigProperties	aws.region	us-west-2

8. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
9. For **CloudWatch logging**, choose the **Enable** check box.
10. Choose **Update**.

Note

When you choose to enable Amazon CloudWatch logging, Managed Service for Apache Flink 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

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/tumbling-window-scala-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"
}
]
}

```

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.

Create and run the application (CLI)

In this section, you use the AWS Command Line Interface to create and run the Managed Service for Apache Flink application. Use the *kinesisanalyticsv2* AWS CLI command to create and interact with Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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. The **MF-stream-rw-role** service execution role should be tailored to the customer-specific role.

```
{
  "ApplicationName": "tumbling_window",
  "ApplicationDescription": "Scala tumbling window application",
  "RuntimeEnvironment": "FLINK-1_15",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "tumbling-window-scala-1.0.jar"
        }
      },
      "CodeContentType": "ZIPFILE"
    },
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-west-2",

```

```

        "stream.name" : "ExampleInputStream",
        "flink.stream.initpos" : "LATEST"
    }
},
{
    "PropertyGroupId": "ProducerConfigProperties",
    "PropertyMap" : {
        "aws.region" : "us-west-2",
        "stream.name" : "ExampleOutputStream"
    }
}
]
}
},
"CloudWatchLoggingOptions": [
    {
        "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
    }
]
}
}

```

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*.

Create an IAM role

In this section, you create an IAM role that the Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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 Managed Service for Apache Flink permission to assume the role, and the permissions policy determines what Managed Service for Apache Flink 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** and then **Create Role**.
3. Under **Select type of trusted identity**, choose **AWS Service**

4. Under **Choose the service that will use this role**, choose **Kinesis**.
5. Under **Select your use case**, choose **Managed Service for Apache Flink**.
6. Choose **Next: Permissions**.
7. On the **Attach permissions policies** page, choose **Next: Review**. You attach permissions policies after you create the role.
8. On the **Create role** page, enter **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-stream-rw-role`. Next, you update the trust and permissions policies for the role

9. Attach the permissions policy to the role.

 **Note**

For this exercise, Managed Service for Apache Flink 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, [Create a Permissions Policy](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the `AKReadSourceStreamWriteSinkStream` 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 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. The `ServiceExecutionRole` should include the IAM user role you created in the previous section.

```
"ApplicationName": "tumbling_window",
  "ApplicationDescription": "Scala getting started application",
  "RuntimeEnvironment": "FLINK-1_15",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "tumbling-window-scala-1.0.jar"
        }
      },
      "CodeContentType": "ZIPFILE"
    },
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
            "stream.name" : "ExampleInputStream",
            "flink.stream.initpos" : "LATEST"
          }
        },
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
            "stream.name" : "ExampleOutputStream"
          }
        }
      ]
    }
  },
  "CloudWatchLoggingOptions": [
    {
      "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-group:MyApplication:log-stream:kinesis-analytics-log-stream"
    }
  ]
}
```

```
}
```

Execute the [CreateApplication](#) with the following 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": "tumbling_window",
  "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 Managed Service for Apache Flink 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": "tumbling_window"
}
```

2. Execute the `StopApplication` action with the preceding 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 [Setting Up Application Logging](#).

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": "tumbling_window",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
            "stream.name" : "ExampleInputStream",
            "flink.stream.initpos" : "LATEST"
          }
        },
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
```

```
        "stream.name" : "ExampleOutputStream"
      }
    }
  ]
}
```

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](#) 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 [Create dependent resources](#) section.

```
{
  "ApplicationName": "tumbling_window",
  "CurrentApplicationVersionId": 1,
```

```
"ApplicationConfigurationUpdate": {
  "ApplicationCodeConfigurationUpdate": {
    "CodeContentUpdate": {
      "S3ContentLocationUpdate": {
        "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
        "FileKeyUpdate": "tumbling-window-scala-1.0.jar",
        "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhyvpvDU"
      }
    }
  }
}
```

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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 Scala

Note

For current examples, see [Examples](#).

Note

Starting from version 1.15 Flink is Scala free. Applications can now use the Java API from any Scala version. Flink still uses Scala in a few key components internally but doesn't expose Scala into the user code classloader. Because of that, users need to add Scala dependencies into their jar-archives.

For more information about Scala changes in Flink 1.15, see [Scala Free in One Fifteen](#).

In this exercise, you will create a simple streaming application which uses Scala 3.2.0 and Flink's Java DataStream API. The application reads data from Kinesis stream, aggregates it using sliding windows and writes results to output Kinesis stream.

Note

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

This topic contains the following sections:

- [Download and examine the application code](#)
- [Compile and upload the application code](#)
- [Create and run the application \(console\)](#)
- [Create and run the application \(CLI\)](#)
- [Update the application code](#)
- [Clean up AWS resources](#)

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-examples.git
```

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

Note the following about the application code:

- A `build.sbt` file contains information about the application's configuration and dependencies, including the Managed Service for Apache Flink libraries.
- The `BasicStreamingJob.scala` 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:

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
    defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
}
```

The application also uses a Kinesis sink to write into the result stream. The following snippet creates the Kinesis sink:

```
private def createSink: KinesisStreamsSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val outputProperties = applicationProperties.get("ProducerConfigProperties")

  KinesisStreamsSink.builder[String]
    .setKinesisClientProperties(outputProperties)
    .setSerializationSchema(new SimpleStringSchema)
    .setStreamName(outputProperties.getProperty(streamNameKey,
    defaultOutputStreamName))
    .setPartitionKeyGenerator((element: String) => String.valueOf(element.hashCode))
    .build
}
```

- The application uses the window operator to find the count of values 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:

```
environment.addSource(createSource)
  .map { value =>
    val jsonNode = jsonParser.readValue(value, classOf[JsonNode])
    new Tuple2[String, Double](jsonNode.get("ticker").toString,
    jsonNode.get("price").asDouble)
  }
  .returns(Types.TUPLE(Types.STRING, Types.DOUBLE))
  .keyBy(v => v.f0) // Logically partition the stream for each word
  .window(SlidingProcessingTimeWindows.of(Time.seconds(10), Time.seconds(5)))
  .min(1) // Calculate minimum price per ticker over the window
  .map { value => value.f0 + String.format("%.2f", value.f1) + "\n" }
  .sinkTo(createSink)
```

- The application creates source and sink connectors to access external resources using a `StreamExecutionEnvironment` object.
- The application creates source and sink connectors using dynamic application properties. Runtime application's properties are read to configure the connectors. For more information about runtime properties, see [Runtime Properties](#).

Compile and upload the application code

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

Compile the Application Code

Use the [SBT](#) build tool to build the Scala code for the application. To install SBT, see [Install sbt with cs setup](#). You also need to install the Java Development Kit (JDK). See [Prerequisites for Completing the Exercises](#).

1. To use your application code, you compile and package it into a JAR file. You can compile and package your code with SBT:

```
sbt assembly
```

2. If the application compiles successfully, the following file is created:

```
target/scala-3.2.0/sliding-window-scala-1.0.jar
```

Upload the Apache Flink Streaming Scala Code

In this section, you create an Amazon S3 bucket and upload your 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 **Configure options**, keep the settings as they are, and choose **Next**.
5. In **Set permissions**, keep the settings as they are, and choose **Next**.
6. Choose **Create bucket**.
7. Choose the `ka-app-code-<username>` bucket, and then choose **Upload**.
8. In the **Select files** step, choose **Add files**. Navigate to the `sliding-window-scala-1.0.jar` file that you created in the previous step.
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 application (console)

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.
 - For **Description**, enter **My Scala test app**.
 - For **Runtime**, choose **Apache Flink**.
 - Leave the version as **Apache Flink version 1.15.2 (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 Managed Service 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: `kinesisanalytics-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 `sliding-window-scala-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 **Add group**.
5. Enter the following:

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

Choose **Save**.

- Under **Properties**, choose **Add group** again.
- Enter the following:

Group ID	Key	Value
ProducerConfigProperties	output.stream.name	ExampleOutputStream
ProducerConfigProperties	aws.region	us-west-2

- 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, Managed Service for Apache Flink 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/sliding-window-scala-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"
}
]
}

```

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.

Create and run the application (CLI)

In this section, you use the AWS Command Line Interface to create and run the Managed Service for Apache Flink application. Use the *kinesisanalyticsv2* AWS CLI command to create and interact with Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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.

```
{
  "ApplicationName": "sliding_window",
  "ApplicationDescription": "Scala sliding window application",
  "RuntimeEnvironment": "FLINK-1_15",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "sliding-window-scala-1.0.jar"
        }
      },
      "CodeContentType": "ZIPFILE"
    },
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-west-2",
            "stream.name": "ExampleInputStream",

```

```
        "flink.stream.initpos" : "LATEST"
      }
    },
    {
      "PropertyGroupId": "ProducerConfigProperties",
      "PropertyMap" : {
        "aws.region" : "us-west-2",
        "stream.name" : "ExampleOutputStream"
      }
    }
  ]
}
},
"CloudWatchLoggingOptions": [
  {
    "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
  }
]
}
```

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*.

Create an IAM role

In this section, you create an IAM role that the Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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 Managed Service for Apache Flink permission to assume the role, and the permissions policy determines what Managed Service for Apache Flink 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** and then **Create Role**.
3. Under **Select type of trusted identity**, choose **AWS Service**
4. Under **Choose the service that will use this role**, choose **Kinesis**.

5. Under **Select your use case**, choose **Managed Service for Apache Flink**.
6. Choose **Next: Permissions**.
7. On the **Attach permissions policies** page, choose **Next: Review**. You attach permissions policies after you create the role.
8. On the **Create role** page, enter **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-stream-rw-role`. Next, you update the trust and permissions policies for the role

9. Attach the permissions policy to the role.

 **Note**

For this exercise, Managed Service for Apache Flink 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, [Create a Permissions Policy](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the `AKReadSourceStreamWriteSinkStream` 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 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": "sliding_window",
  "ApplicationDescription": "Scala sliding_window application",
  "RuntimeEnvironment": "FLINK-1_15",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "sliding-window-scala-1.0.jar"
        }
      },
      "CodeContentType": "ZIPFILE"
    },
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
            "stream.name" : "ExampleInputStream",
            "flink.stream.initpos" : "LATEST"
          }
        },
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
            "stream.name" : "ExampleOutputStream"
          }
        }
      ]
    }
  },
  "CloudWatchLoggingOptions": [
    {
      "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
    }
  ]
}

```

Execute the [CreateApplication](#) with the following 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": "sliding_window",
  "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 Managed Service for Apache Flink 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": "sliding_window"
}
```

2. Execute the StopApplication action with the preceding 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 [Setting Up Application Logging](#).

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": "sliding_window",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
            "stream.name" : "ExampleInputStream",
            "flink.stream.initpos" : "LATEST"
          }
        },
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
            "stream.name" : "ExampleOutputStream"
          }
        }
      ]
    }
  }
}
```

```
    ]
  }
}
}
```

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](#) 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 [Create dependent resources](#) section.

```
{
  "ApplicationName": "sliding_window",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationCodeConfigurationUpdate": {
      "CodeContentUpdate": {
```



```
        "S3ContentLocationUpdate": {
            "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
            "FileKeyUpdate": "-1.0.jar",
            "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhyvpDU"
        }
    }
}
```

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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 Scala

Note

For current examples, see [Examples](#).

Note

Starting from version 1.15 Flink is Scala free. Applications can now use the Java API from any Scala version. Flink still uses Scala in a few key components internally but doesn't

expose Scala into the user code classloader. Because of that, users need to add Scala dependencies into their jar-archives.

For more information about Scala changes in Flink 1.15, see [Scala Free in One Fifteen](#).

In this exercise, you will create a simple streaming application which uses Scala 3.2.0 and Flink's Java DataStream API. The application reads data from Kinesis stream, aggregates it using sliding windows and writes results to S3.

Note

To set up required prerequisites for this exercise, first complete the [Getting Started \(Scala\)](#) exercise. You only need to create an additional folder **data/** in the Amazon S3 bucket *ka-app-code-`<username>`*.

This topic contains the following sections:

- [Download and examine the application code](#)
- [Compile and upload the application code](#)
- [Create and run the application \(console\)](#)
- [Create and run the application \(CLI\)](#)
- [Update the application code](#)
- [Clean up AWS resources](#)

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-examples.git
```

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

Note the following about the application code:

- A `build.sbt` file contains information about the application's configuration and dependencies, including the Managed Service for Apache Flink libraries.
- The `BasicStreamingJob.scala` 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:

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
    defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
}
```

The application also uses a `StreamingFileSink` to write to an Amazon S3 bucket:

```
def createSink: StreamingFileSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val s3SinkPath =
    applicationProperties.get("ProducerConfigProperties").getProperty("s3.sink.path")

  StreamingFileSink
    .forRowFormat(new Path(s3SinkPath), new SimpleStringEncoder[String]("UTF-8"))
    .build()
}
```

- The application creates source and sink connectors to access external resources using a `StreamExecutionEnvironment` object.
- The application creates source and sink connectors using dynamic application properties. Runtime application's properties are read to configure the connectors. For more information about runtime properties, see [Runtime Properties](#).

Compile and upload the application code

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

Compile the Application Code

Use the [SBT](#) build tool to build the Scala code for the application. To install SBT, see [Install sbt with cs setup](#). You also need to install the Java Development Kit (JDK). See [Prerequisites for Completing the Exercises](#).

1. To use your application code, you compile and package it into a JAR file. You can compile and package your code with SBT:

```
sbt assembly
```

2. If the application compiles successfully, the following file is created:

```
target/scala-3.2.0/s3-sink-scala-1.0.jar
```

Upload the Apache Flink Streaming Scala Code

In this section, you create an Amazon S3 bucket and upload your 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 **Configure options**, keep the settings as they are, and choose **Next**.
5. In **Set permissions**, keep the settings as they are, and choose **Next**.
6. Choose **Create bucket**.
7. Choose the `ka-app-code-<username>` bucket, and then choose **Upload**.
8. In the **Select files** step, choose **Add files**. Navigate to the `s3-sink-scala-1.0.jar` file that you created in the previous step.
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 application (console)

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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 1.15.2 (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 Managed Service 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: `kinesisanalytics-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 `s3-sink-scala-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 **Add group**.
5. Enter the following:

Group ID	Key	Value
ConsumerConfigProperties	input.stream.name	ExampleInputStream
ConsumerConfigProperties	aws.region	us-west-2
ConsumerConfigProperties	flink.stream.initpos	LATEST

Choose **Save**.

6. Under **Properties**, choose **Add group**.
7. Enter the following:

Group ID	Key	Value
ProducerConfigProperties	s3.sink.path	s3a://ka-app-code- <i><user-name></i> /data

8. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
9. For **CloudWatch logging**, choose the **Enable** check box.
10. Choose **Update**.

Note

When you choose to enable Amazon CloudWatch logging, Managed Service for Apache Flink 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

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:Abort*",
        "s3:DeleteObject*",
        "s3:GetObject*",
        "s3:GetBucket*",
        "s3:List*",
        "s3:ListBucket",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::ka-app-code-<username>",
        "arn:aws:s3:::ka-app-code-<username>/*"
      ]
    },
    {
      "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"
  }
]
}

```

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.

Create and run the application (CLI)

In this section, you use the AWS Command Line Interface to create and run the Managed Service for Apache Flink application. Use the *kinesisanalyticsv2* AWS CLI command to create and interact with Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
```

```

        "s3:GetObjectVersion"
    ],
    "Resource": [
        "arn:aws:s3:::ka-app-code-username/getting-started-scala-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"
    }
  ]
}
```

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*.

Create an IAM role

In this section, you create an IAM role that the Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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 Managed Service for Apache Flink permission to assume the role, and the permissions policy determines what Managed Service for Apache Flink 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** and then **Create Role**.
3. Under **Select type of trusted identity**, choose **AWS Service**
4. Under **Choose the service that will use this role**, choose **Kinesis**.
5. Under **Select your use case**, choose **Managed Service for Apache Flink**.
6. Choose **Next: Permissions**.
7. On the **Attach permissions policies** page, choose **Next: Review**. You attach permissions policies after you create the role.
8. On the **Create role** page, enter **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-stream-rw-role`. Next, you update the trust and permissions policies for the role

9. Attach the permissions policy to the role.

Note

For this exercise, Managed Service for Apache Flink 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, [Create a Permissions Policy](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the **AKReadSourceStreamWriteSinkStream** 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 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": "s3_sink",
  "ApplicationDescription": "Scala tumbling window application",
  "RuntimeEnvironment": "FLINK-1_15",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
```

```

        "FileKey": "s3-sink-scala-1.0.jar"
      }
    },
    "CodeContentType": "ZIPFILE"
  },
  "EnvironmentProperties": {
    "PropertyGroups": [
      {
        "PropertyGroupId": "ConsumerConfigProperties",
        "PropertyMap" : {
          "aws.region" : "us-west-2",
          "stream.name" : "ExampleInputStream",
          "flink.stream.initpos" : "LATEST"
        }
      },
      {
        "PropertyGroupId": "ProducerConfigProperties",
        "PropertyMap" : {
          "s3.sink.path" : "s3a://ka-app-code-<username>/data"
        }
      }
    ]
  }
},
"CloudWatchLoggingOptions": [
  {
    "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
  }
]
}

```

Execute the [CreateApplication](#) with the following 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": "s3_sink",
  "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 Managed Service for Apache Flink 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": "s3_sink"
}
```

2. Execute the `StopApplication` action with the preceding 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 [Setting Up Application Logging](#).

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": "s3_sink",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap": {
            "aws.region": "us-west-2",
            "stream.name": "ExampleInputStream",
            "flink.stream.initpos": "LATEST"
          }
        },
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap": {
            "s3.sink.path": "s3a://ka-app-code-<username>/data"
          }
        }
      ]
    }
  }
}
```

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](#) 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 [Create dependent resources](#) section.

```
{
  "ApplicationName": "s3_sink",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationCodeConfigurationUpdate": {
      "CodeContentUpdate": {
        "S3ContentLocationUpdate": {
          "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
          "FileKeyUpdate": "s3-sink-scala-1.0.jar",
          "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhyvDU"
        }
      }
    }
  }
}
```

```
}
```

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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.

Using a Studio notebook with Managed Service for Apache Flink

Studio notebooks for Managed Service for Apache Flink 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 Managed Service for Apache Flink and Apache Zeppelin, see [Creating a Studio notebook tutorial](#). 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, Managed Service for Apache Flink stream-processing application for your production workloads.

This topic contains the following sections:

- [Studio notebook Runtime versions](#)
- [Creating a Studio notebook](#)
- [Interactive analysis of streaming data](#)
- [Deploying as an application with durable state](#)

- [IAM permissions for Studio notebooks](#)
- [Using connectors and dependencies](#)
- [Implement user-defined functions](#)
- [Enabling checkpointing](#)
- [Upgrading Studio Runtime](#)
- [Working with AWS Glue](#)
- [Examples and tutorials](#)
- [Troubleshooting](#)
- [Appendix: Creating custom IAM policies](#)

Studio notebook Runtime versions

With Amazon Managed Service for Apache Flink Studio, you can query data streams in real time and build and run stream processing applications using standard SQL, Python, and Scala in an interactive notebook. Studio notebooks are powered by [Apache Zeppelin](#) and use [Apache Flink](#) as the stream processing engine.

Note

We will deprecate Studio Runtime with **Apache Flink version 1.11 on November 5, 2024**. Starting from this date, you will not be able to run new notebooks or create new applications using this version. We recommend that you upgrade to the latest runtime (Apache Flink 1.15 and Apache Zeppelin 0.10) before that time. For guidance on how to upgrade your notebook, see [Upgrading Studio Runtime](#).

Studio Runtime

Apache Flink version	Apache Zeppelin version	Python version	
1.15	0.10	3.8	Recommended
1.13	0.9	3.8	Supported

Apache Flink version	Apache Zeppelin version	Python version	
1.11	0.9	3.7	Deprecating on November 5, 2024

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](#).
- 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 Managed Service for Apache Flink for Apache Flink](#).

- **Logging and Monitoring:** For information about application logging and monitoring, see [Logging and Monitoring in Amazon Managed Service for Apache Flink 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 Managed Service for Apache Flink console choose the **Studio** tab, then choose **Create Studio notebook**.

For a tutorial, see [Event Detection with Managed Service for Apache Flink](#).

For an example of a more advanced Studio notebook solution, see [Apache Flink on Amazon Managed Service for Apache Flink Studio](#).

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](#) 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 Managed Service for Apache Flink uses to run your application by using an *interpreter*. You can use the following interpreters with Managed Service for Apache Flink:

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

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

If you are using `%flink.pyflink` or `%flink.ipynflink` as your interpreters, you will need to use the `ZeppelinContext` to visualize the results within the notebook.

For more PyFlink specific examples, see [Query your data streams interactively using Managed Service for Apache Flink Studio and Python](#).

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
<code>senv</code>	<code>StreamExecutionEnvironment</code>

Variable	Resource
<code>stenv</code>	<code>StreamTableEnvironment for blink planner</code>

You access Python table environment resources with the following variables:

Variable	Resource
<code>s_env</code>	<code>StreamExecutionEnvironment</code>
<code>st_env</code>	<code>StreamTableEnvironment for blink planner</code>

For more information about using table environments, see [Concepts and Common API](#) 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 Managed Service for Apache Flink: 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, Managed Service for Apache Flink 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.ipyflink`, `%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.ipyflink` in a note with `%flink` and `%flink.ssql` 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.ssql(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.
- If you are deploying as an application with durable state your VPC must have internet access. If your VPC does not have internet access, see [Deploying as an application with durable state in a VPC with no internet access](#).

Scala/Python criteria

- In your Scala or Python code, use the [Blink planner](#) (`send`, `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.

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.

For more information and an example, see [Translate, redact and analyze streaming data using SQL functions with Amazon Managed Service for Apache Flink, Amazon Translate, and Amazon Comprehend](#).

IAM permissions for Studio notebooks

Managed Service for Apache Flink 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

Service	Access
Amazon EC2	List
AWS Glue	Read, Write
Managed Service for Apache Flink	Read
Managed Service for Apache Flink V2	Read
Amazon S3	Read, Write

Using connectors and dependencies

Connectors enable you to read and write data across various technologies. Managed Service for Apache Flink 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, Managed Service for Apache Flink includes the following custom connectors by default: `flink-sql-connector-kinesis`, `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",  
    "Version": "1.15.4"  
  }  
}]
```

For authenticating with AWS MSK through AWS IAM:

```
"CustomArtifactsConfiguration": [{  
  "ArtifactType": "DEPENDENCY_JAR",  
  "MavenReference": {  
    "GroupId": "software.amazon.msk",  
    "ArtifactId": "aws-msk-iam-auth",  
    "Version": "1.1.6"  
  }  
}]
```

For the Apache Kafka connector:

```
"CustomArtifactsConfiguration": [{  
  "ArtifactType": "DEPENDENCY_JAR",  
  "MavenReference": {  
    "GroupId": "org.apache.flink",  
  
    "ArtifactId": "flink-connector-kafka",  
    "Version": "1.15.4"  
  }  
}]
```

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.

Adding 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.

Implement 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.

When implementing JARs that register abstract classes that subclass `UserDefinedFunction` (or your own abstract classes), use `provided` scope in Apache Maven, `compileOnly` dependency declarations in Gradle, `provided` scope in SBT, or an equivalent directive in your UDF project build configuration. This allows the UDF source code to compile against the Flink APIs, but the Flink API classes are not themselves included in the build artifacts. Refer to this [pom](#) from the UDF jar example which adheres to such prerequisite on a Maven project.

Note

For an example setup, see [Translate, redact and analyze streaming data using SQL functions with Amazon Managed Service for Apache Flink, Amazon Translate, and Amazon Comprehend](#) on the *AWS Machine Learning Blog*.

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.

Considerations with user-defined functions

- Managed Service for Apache Flink Studio uses the [Apache Zeppelin terminology](#) wherein a notebook is a Zeppelin instance that can contain multiple notes. Each note can then contain multiple paragraphs. With Managed Service for Apache Flink Studio the interpreter process is shared across all the notes in the notebook. So if you perform an explicit function registration using [createTemporarySystemFunction](#) in one note, the same can be referenced as-is in another note of same notebook.

The *Deploy as application* operation however works on an *individual* note and not all notes in the notebook. When you perform deploy as application, only active note's contents are used to generate the application. Any explicit function registration performed in other notebooks are not part of the generated application dependencies. Additionally, during Deploy as application option an implicit function registration occurs by converting the main class name of JAR to a lowercase string.

For example, if `TextAnalyticsUDF` is the main class for UDF JAR, then an implicit registration will result in function name `textanalyticsudf`. So if an explicit function registration in note 1 of Studio occurs like the following, then all other notes in that notebook (say note 2) can refer the function by name `myNewFuncNameForClass` because of the shared interpreter:

```
stenv.createTemporarySystemFunction("myNewFuncNameForClass", new
TextAnalyticsUDF())
```

However during deploy as application operation on note 2, this explicit registration *will not be included* in the dependencies and hence the deployed application will not perform as expected. Because of the implicit registration, by default all references to this function is expected to be with `textanalyticsudf` and not `myNewFuncNameForClass`.

If there is a need for custom function name registration then note 2 itself is expected to contain another paragraph to perform another explicit registration as follows:

```
%flink(parallelism=1)
import com.amazonaws.kinesis.udf.textanalytics.TextAnalyticsUDF
# re-register the JAR for UDF with custom name
stenv.createTemporarySystemFunction("myNewFuncNameForClass", new TextAnalyticsUDF())
```



```
%flink. ssl(type=update, parallelism=1)
INSERT INTO
    table2
SELECT
    myNewFuncNameForClass(column_name)
FROM
    table1
;
```

- If your UDF JAR includes Flink SDKs, then configure your Java project so that the UDF source code can compile against the Flink SDKs, but the Flink SDK classes are not themselves included in the build artifact, for example the JAR.

You can use `provided` scope in Apache Maven, `compileOnly` dependency declarations in Gradle, `provided` scope in SBT, or equivalent directive in their UDF project build configuration. You can refer to this [pom](#) from the UDF jar example, which adheres to such a prerequisite on a maven project. For a complete step-by-step tutorial, see this [Translate, redact and analyze streaming data using SQL functions with Amazon Managed Service for Apache Flink, Amazon Translate, and Amazon Comprehend](#).

Enabling checkpointing

You enable checkpointing by using environment settings. For information about checkpointing, see [Fault Tolerance](#) in the [Managed Service for Apache Flink 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"
)
```

Upgrading Studio Runtime

This section contains information about how to upgrade your Studio notebook Runtime. We recommend that you always upgrade to the latest supported Studio Runtime.

Upgrading your notebook to a new Studio Runtime

Depending on how you use Studio, the steps to upgrade your Runtime differ. Select the option that fits your use case.

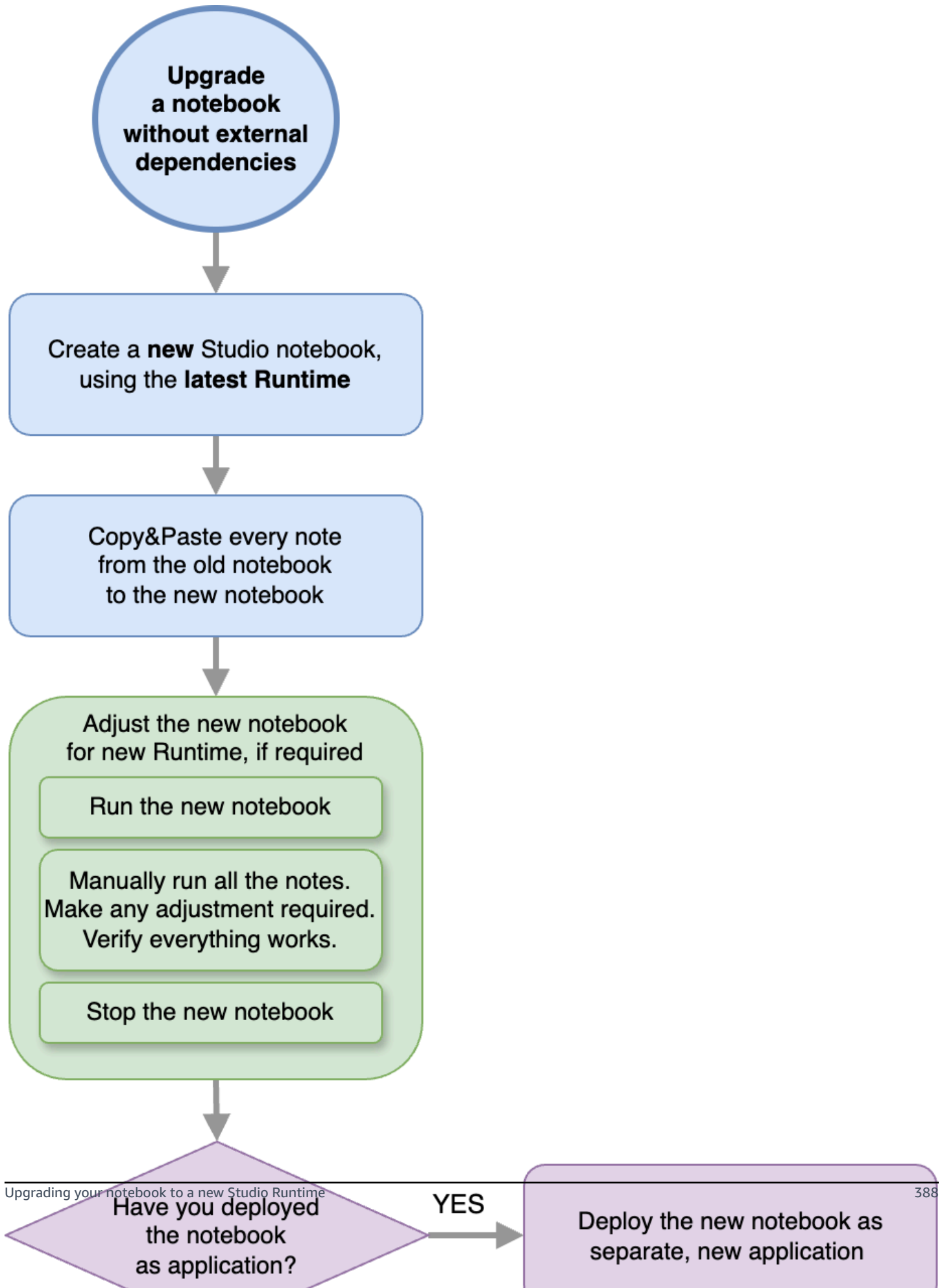
SQL queries or Python code with no external dependencies

If you are using SQL or Python without any external dependencies, use the following Runtime upgrade process. We recommend that you upgrade to the latest Runtime version. The upgrade process is the same, regardless of the Runtime version you are upgrading from.

1. Create a new Studio notebook using the latest Runtime.
2. Copy and paste the code of every note from the old notebook to the new notebook.
3. In the new notebook, adjust the code to make it compatible with any Apache Flink feature that has changed from the previous version.
 - Run the new notebook. Open the notebook and run it note by note, in sequence, and test if it works.

- Make any required changes to the code.
 - Stop the new notebook.
4. If you had deployed the old notebook as application:
 - Deploy the new notebook as a separate, new application.
 - Stop the old application.
 - Run the new application without snapshot.
 5. Stop the old notebook if it's running. Start the new notebook, as required, for interactive use.

Process flow for upgrading without external dependencies

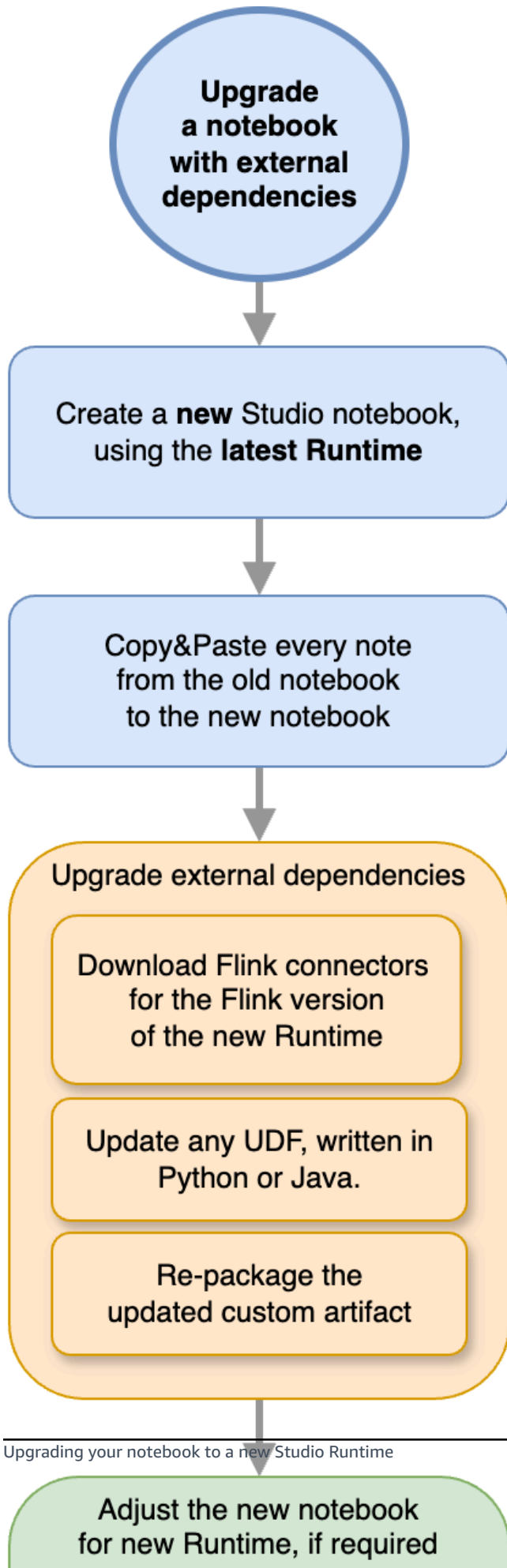


SQL queries or Python code with external dependencies

Follow this process if you are using SQL or Python and using external dependencies such as connectors or custom artifacts, like user-defined functions implemented in Python or Java. We recommend that you upgrade to the latest Runtime. The process is the same, regardless of the Runtime version that you are upgrading from.

1. Create a new Studio notebook using the latest Runtime.
2. Copy and paste the code of every note from the old notebook to the new notebook.
3. Update the external dependencies and custom artifacts.
 - Look for new connectors compatible with the Apache Flink version of the new Runtime. Refer to [Table & SQL Connectors](#) in the Apache Flink documentation to find the correct connectors for the Flink version.
 - Update the code of user-defined functions to match changes in the Apache Flink API, and any Python or JAR dependencies used by the user-defined functions. Re-package your updated custom artifact.
 - Add these new connectors and artifacts to the new notebook.
4. In the new notebook, adjust the code to make it compatible with any Apache Flink feature that has changed from the previous version.
 - Run the new notebook. Open the notebook and run it note by note, in sequence, and test if it works.
 - Make any required changes to the code.
 - Stop the new notebook.
5. If you had deployed the old notebook as application:
 - Deploy the new notebook as a separate, new application.
 - Stop the old application.
 - Run the new application without snapshot.
6. Stop the old notebook if it's running. Start the new notebook, as required, for interactive use.

Process flow for upgrading with external dependencies



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](#).

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](#) in the [Creating a Studio notebook tutorial](#) tutorial.

Table properties

In addition to data fields, your AWS Glue tables provide other information to your Studio notebook using table properties. Managed Service for Apache Flink uses the following AWS Glue table properties:

- [Using Apache Flink time values](#): These properties define how Managed Service for Apache Flink emits Apache Flink internal data processing time values.
- [Using Flink connector and format properties](#): 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 **managed-flink.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 Managed Service for Apache Flink 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
Processing Time	managed-flink.proctime	The column name that AWS Glue will use to expose the value. This column name does not correspond to an existing table column.
Event Time	managed-flink.rowtime	The column name that AWS Glue will use to expose the value. This column name corresponds to an existing table column.
	managed-flink.watermark. <i>column_name</i> .milliseconds	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 Managed Service for Apache Flink uses for connectors are as follows:

Connector Type	Key	Value
Kafka	format	The format used to deserialize and serialize Kafka messages, e.g. json or csv.
	scan.startup.mode	The startup mode for the Kafka consumer, e.g.

Connector Type	Key	Value
Kinesis	<code>format</code>	<code>earliest-offset</code> or <code>timestamp</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 Managed Service for Apache Flink](#)
- [Tutorial: Deploying as an application with durable state](#)
- [Examples](#)

Tutorial: Creating a Studio notebook in Managed Service for Apache Flink

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](#)
- [Create an AWS Glue database](#)
- [Next steps](#)
- [Creating a Studio notebook with Kinesis Data Streams](#)
- [Creating a Studio notebook with Amazon MSK](#)
- [Cleaning up your application and dependent resources](#)

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](#) : 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](#) : 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](#)
- [Create an AWS Glue table](#)
- [Create a Studio notebook with Kinesis Data Streams](#)
- [Send data to your Kinesis data stream](#)
- [Test your Studio notebook](#)

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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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
ticker	string
price	double

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 `managed-flink.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](#)
- [Create a Studio notebook using the AWS CLI](#)

Create a Studio notebook using the AWS Management Console

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard>.
2. In the **Managed Service for Apache Flink 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-3_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:kinesisanalyticsus-
east-1:012345678901:application/MyNotebook",
    "ApplicationName": "MyNotebook",
    "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
    "ApplicationMode": "INTERACTIVE",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/ZeppeleinRole",
    ...
  }
}
```

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

```
aws kinesisanalyticsv2 start-application --application-arn
arn:aws:kinesisanalyticsus-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:

Region	us-east-1
Stream/Firehose stream	ExampleInputStream
Records per second	1

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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard>.
2. On the **Managed Service for Apache Flink 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 [Managed Service for Apache Flink 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](#)
- [Add a NAT gateway to your VPC](#)
- [Create an AWS Glue connection and table](#)
- [Create a Studio notebook with Amazon MSK](#)
- [Send data to your Amazon MSK cluster](#)
- [Test your Studio notebook](#)

Setup

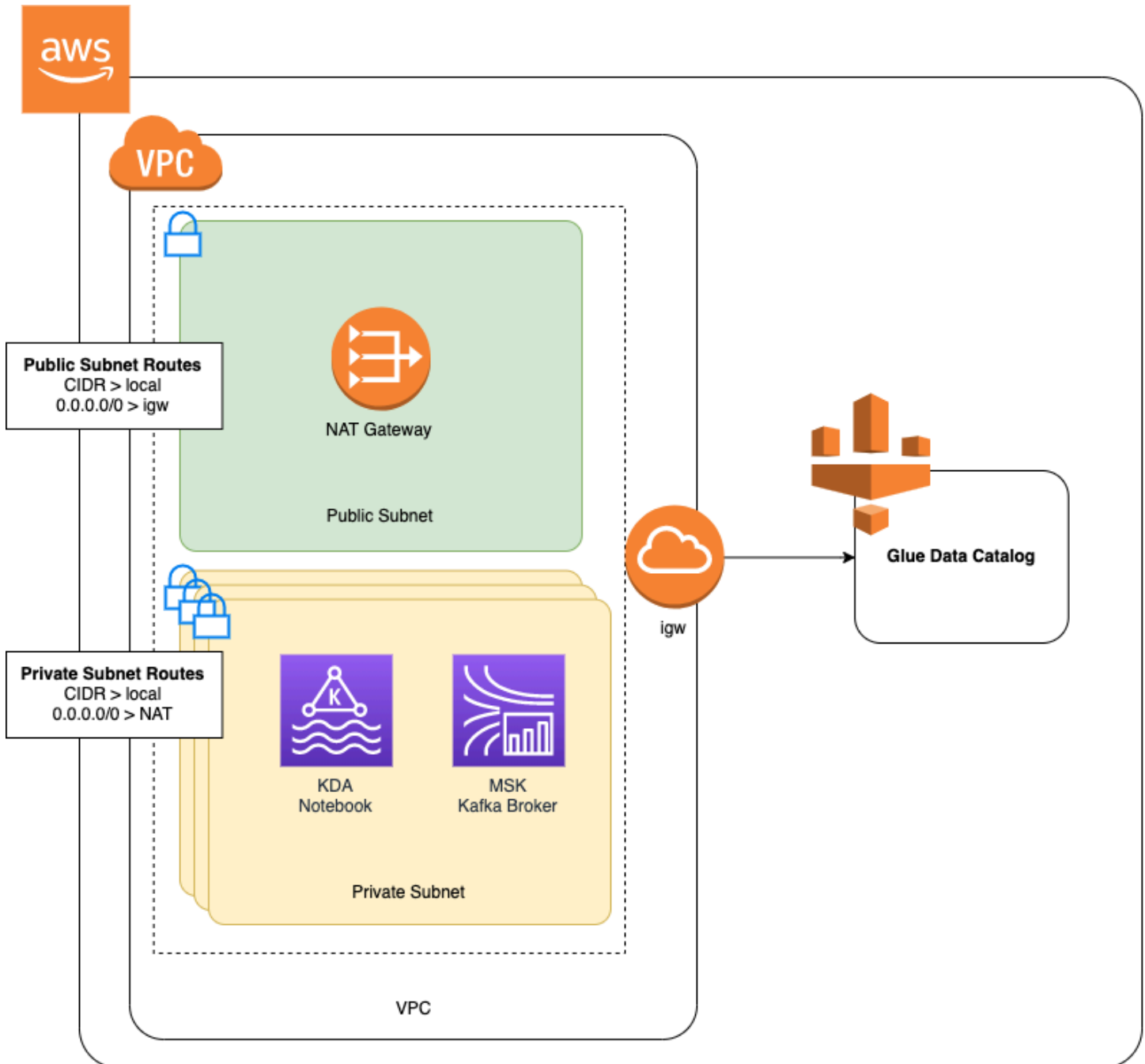
For this tutorial, you need an Amazon MSK cluster that allows plaintext access. If you don't have an Amazon MSK cluster set up already, follow the [Getting Started Using Amazon MSK](#) tutorial to create an Amazon VPC, an Amazon MSK cluster, a topic, and an Amazon EC2 client instance.

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

Name - optional

ZeppelinGateway

Subnet

AWSKafkaTutorialSubnet1

Elastic IP allocation ID

Choose an available Elastic IP. If there are no Elastic IPs available, choose **Allocate Elastic IP**, and then choose the Elastic IP that the console creates.

Choose **Create NAT Gateway**.

5. On the left navigation bar, choose **Route Tables**.
6. Choose **Create Route Table**.
7. On the **Create route table** page, provide the following information:
 - **Name tag:** **ZeppelinRouteTable**
 - **VPC:** Choose your VPC (e.g. **AWSKafkaTutorialVPC**).

Choose **Create**.

8. In the list of route tables, choose **ZeppelinRouteTable**. Choose the **Routes** tab, and choose **Edit routes**.
9. In the **Edit Routes** page, choose **Add route**.
10. In the **For Destination**, enter **0.0.0.0/0**. For **Target**, choose **NAT Gateway, ZeppelinGateway**. Choose **Save Routes**. Choose **Close**.
11. On the Route Tables page, with **ZeppelinRouteTable** selected, choose the **Subnet associations** tab. Choose **Edit subnet associations**.
12. 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

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. Choose **Connections** from the left navigation bar. Choose **Add Connection**.
4. 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 Managed Service for Apache Flink 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
ticker	string
price	double

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 `managed-flink.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](#)
- [Create a Studio notebook using the AWS CLI](#)

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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard>.
2. In the **Managed Service for Apache Flink 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-3_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 kinesisanalyticstv2 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:kinesisanalyticstv2:us-east-1:012345678901:application/MyNotebook",
    "ApplicationName": "MyNotebook",
    "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
    "ApplicationMode": "INTERACTIVE",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/ZeppeleinRole",
    ...
  }
}

```

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

```
aws kinesisanalyticstv2 start-application --application-arn
arn:aws:kinesisanalyticstv2: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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard>.
2. On the **Managed Service for Apache Flink 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 [Managed Service for Apache Flink 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 Managed Service for Apache Flink 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/KinesisAnalytics/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 Managed Service for Apache Flink application with durable state.

This tutorial contains the following sections:

- [Setup](#)
- [Deploy an application with durable state using the AWS Management Console](#)
- [Deploy an application with durable state using the AWS CLI](#)

Setup

Create a new Studio notebook by following the [Creating a Studio notebook tutorial](#), 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 Managed Service for Apache Flink application:
 - AmazonS3FullAccess

- Amazonmanaged-flinkFullAccess
- Access to your sources, destinations, and VPCs as applicable. For more information, see [IAM permissions for Studio notebooks](#).

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 Managed Service for Apache Flink 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](#).

The following example code creates a new Studio notebook:

```
aws kinesisanalyticsv2 create-application \  
  --application-name <app-name> \  
  --runtime-environment ZEPPELIN-FLINK-3_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 kinesisanalyticstv2 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 kinesisanalyticstv2 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.

- [Creating tables with Amazon MSK/Apache Kafka](#)
- [Creating tables with Kinesis](#)
- [Tumbling window](#)
- [Sliding window](#)
- [Interactive SQL](#)
- [BlackHole SQL connector](#)
- [Data generator](#)
- [Interactive Scala](#)

- [Interactive Python](#)
- [Interactive Python, SQL, and Scala](#)
- [Cross-account Kinesis data stream](#)

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](#).

Creating tables with Amazon MSK/Apache Kafka

You can use the Amazon MSK Flink connector with Managed Service for Apache Flink Studio to authenticate your connection with Plaintext, SSL, or IAM authentication. Create your tables using the specific properties per your requirements.

```
-- Plaintext connection

CREATE TABLE your_table (
  `column1` STRING,
  `column2` BIGINT
) WITH (
  'connector' = 'kafka',
  'topic' = 'your_topic',
  'properties.bootstrap.servers' = '<bootstrap servers>',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);

-- SSL connection

CREATE TABLE your_table (
  `column1` STRING,
  `column2` BIGINT
) WITH (
  'connector' = 'kafka',
```

```
'topic' = 'your_topic',
'properties.bootstrap.servers' = '<bootstrap servers>',
'properties.security.protocol' = 'SSL',
'properties.ssl.truststore.location' = '/usr/lib/jvm/java-11-amazon-corretto/lib/
security/cacerts',
'properties.ssl.truststore.password' = 'changeit',
'properties.group.id' = 'myGroup',
'scan.startup.mode' = 'earliest-offset',
'format' = 'json'
);

-- IAM connection (or for MSK Serverless)

CREATE TABLE your_table (
  `column1` STRING,
  `column2` BIGINT
) WITH (
  'connector' = 'kafka',
  'topic' = 'your_topic',
  'properties.bootstrap.servers' = '<bootstrap servers>',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.mechanism' = 'AWS_MSK_IAM',
  'properties.sasl.jaas.config' = 'software.amazon.msk.auth.iam.IAMLoginModule
required;',
  'properties.sasl.client.callback.handler.class' =
'software.amazon.msk.auth.iam.IAMClientCallbackHandler',
  'properties.group.id' = 'myGroup',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);
```

You can combine these with other properties at [Apache Kafka SQL Connector](#).

Creating tables with Kinesis

In the following example, you create a table using Kinesis:

```
CREATE TABLE KinesisTable (
  `column1` BIGINT,
  `column2` BIGINT,
  `column3` BIGINT,
  `column4` STRING,
  `ts` TIMESTAMP(3)
```

```
)  
PARTITIONED BY (column1, column2)  
WITH (  
  'connector' = 'kinesis',  
  'stream' = 'test_stream',  
  'aws.region' = '<region>',  
  'scan.stream.initpos' = 'LATEST',  
  'format' = 'csv'  
);
```

For more information on other properties you can use, see [Amazon Kinesis Data Streams SQL Connector](#).

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”](#) 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”](#). 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.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(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.ssql(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”](#). 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.ssql(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.sql
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'
)
```

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 [Managed Service for Apache Flink Developer Guide](#).

Deploying as an application with durable state in a VPC with no internet access

The Managed Service for Apache Flink Studio deploy-as-application function does not support VPC applications without internet access. We recommend that you build your application in Studio, and

then use Managed Service for Apache Flink to manually create a Flink application and select the zip file you built in your Notebook.

The following steps outline this approach:

1. Build and export your Studio application to Amazon S3. This should be a zip file.
2. Create a Managed Service for Apache Flink application manually with code path referencing the zip file location in Amazon S3. In addition, you will need to configure the application with the following env variables (2 `groupID`, 3 `var` in total):
3. `kinesis.analytics.flink.run.options`
 - a. `python: source/note.py`
 - b. `jarfile: lib/PythonApplicationDependencies.jar`
4. `managed.deploy_as_app.options`
 - DatabaseARN: *<glue database ARN (Amazon Resource Name)>*
5. You may need to give permissions to the Managed Service for Apache Flink Studio and Managed Service for Apache Flink IAM roles for the services your application uses. You can use the same IAM role for both apps.

Deploy-as-app size and build time reduction

Studio `deploy-as-app` for Python applications packages everything available in the Python environment because we cannot determine which libraries you need. This may result in a larger-than necessary `deploy-as-app` size. The following procedure demonstrates how to reduce the size of the `deploy-as-app` Python application size by uninstalling dependencies.

If you're building a Python application with `deploy-as-app` feature from Studio, you might consider removing pre-installed Python packages from the system if your applications are not depending on. This will not only help to reduce the final artifact size to avoid breaching the service limit for application size, but also improve the build time of applications with the `deploy-as-app` feature.

You can execute following command to list out all installed Python packages with their respective installed size and selectively remove packages with significant size.

```
%flink.pyflink
```

```
!pip list --format freeze | awk -F = {'print $1'} | xargs pip show | grep -E  
'Location:|Name:' | cut -d ' ' -f 2 | paste -d ' ' - - | awk '{gsub("-", "_", $1); print  
$2 "/" tolower($1)}' | xargs du -sh 2> /dev/null | sort -hr
```

Note

apache-beam is required by Flink Python to operate. You should never remove this package and its dependencies.

Following is the list of pre-install Python packages in Studio V2 which can be considered for removal:

```
scipy  
statsmodels  
plotnine  
seaborn  
llvmlite  
bokeh  
pandas  
matplotlib  
botocore  
boto3  
numba
```

To remove a Python package from Zeppelin notebook:

1. Check if your application depends on the package, or any of its consuming packages, before removing it. You can identify dependants of a package using [pipdeptree](#).
2. Executing following command to remove a package:

```
%flink.pyflink  
!pip uninstall -y <package-to-remove>
```

3. If you need to retrieve a package which you removed by mistake, executing the following command:

```
%flink.pyflink  
!pip install <package-to-install>
```

Example Example: Remove `scipy` package before deploying your Python application with `deploy-as-app` feature.

1. Use `pipdeptree` to discover all `scipy` consumers and verify if you can safely remove `scipy`.
 - Install the tool through notebook:

```
%flink.pyflink
!pip install pipdeptree
```

- Get reversed dependency tree of `scipy` by running:

```
%flink.pyflink
!pip -r -p scipy
```

You should see output similar to the following (condensed for brevity):

```
...
-----
scipy==1.8.0
### plotnine==0.5.1 [requires: scipy>=1.0.0]
### seaborn==0.9.0 [requires: scipy>=0.14.0]
### statsmodels==0.12.2 [requires: scipy>=1.1]
    ### plotnine==0.5.1 [requires: statsmodels>=0.8.0]
```

2. Carefully inspect the usage of `seaborn`, `statsmodels` and `plotnine` in your applications. If your applications do not depend on any of `scipy`, `seaborn`, `statemodels`, or `plotnine`, you can remove all of these packages, or only ones which your applications don't need.
3. Remove the package by running:

```
!pip uninstall -y scipy plotnine seaborn statemodels
```

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](#)
- [CloudWatch Logs](#)
- [Kinesis streams](#)
- [Amazon MSK clusters](#)

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 Managed Service for Apache Flink (DataStream API)

This section introduces you to the fundamental concepts of Managed Service 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

- [Review the components of the Managed Service for Apache Flink application](#)
- [Fulfill the prerequisites for completing the exercises](#)
- [Step 1: Set up an AWS account and create an administrator user](#)
- [Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)
- [Step 3: Create and run a Managed Service for Apache Flink application](#)
- [Step 4: Clean up AWS resources](#)
- [Step 5: Use resources to complete next steps](#)

Review the components of the Managed Service for Apache Flink application

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

An Managed Service 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](#).
- **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](#).

- **Sink:** The application produces data to external sources by using *sinks*. A sink connector writes data to a Kinesis data stream, a Firehose stream, an Amazon S3 bucket, etc. For more information, see [Sinks](#).

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 Managed Service for Apache Flink 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.

Fulfill the 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](#).

Step 1: Set up an AWS account and create an administrator user

Before you use Managed Service for Apache Flink for the first time, complete the following tasks:

Sign up for an AWS account

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

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.

2. Follow the online instructions.

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

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

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

Create a user with administrative access

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

Secure your AWS account root user

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

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

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

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

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

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

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

Sign in as the user with administrative access

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

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

Assign access to additional users

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

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

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

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use.

Which user needs programmatic access?	To	By
		<ul style="list-style-type: none">• For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the <i>AWS Command Line Interface User Guide</i>.• For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the <i>AWS SDKs and Tools Reference Guide</i>.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the <i>IAM User Guide</i> .

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. <ul style="list-style-type: none">• For the AWS CLI, see Authenticating using IAM user credentials in the <i>AWS Command Line Interface User Guide</i>.• For AWS SDKs and tools, see Authenticate using long-term credentials in the <i>AWS SDKs and Tools Reference Guide</i>.• For AWS APIs, see Managing access keys for IAM users in the <i>IAM User Guide</i>.

Next Step

[Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)

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

In this step, you download and configure the AWS CLI to use with Managed Service 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 Managed Service for Apache Flink application](#)

Step 3: Create and run a Managed Service for Apache Flink application

In this exercise, you create a Managed Service for Apache Flink application with data streams as a source and a sink.

This section contains the following steps:

- [Create two Amazon Kinesis data streams](#)
- [Write sample records to the input stream](#)
- [Download and examine the Apache Flink streaming Java code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Next step](#)

Create two Amazon Kinesis data streams

Before you create a Managed Service for Apache Flink 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', region_name='us-west-2'))
```

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-managed-service-for-apache-flink-examples.git
```

2. Navigate to the `amazon-managed-service-for-apache-flink-examples/tree/main/java/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 Managed Service 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](#).

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 [Fulfill the prerequisites for completing the exercises](#).

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=1.19.1
```

- 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/amazon-msf-java-stream-app-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 `amazon-msf-java-stream-app-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 Managed Service for Apache Flink application

You can create and run a Managed Service for Apache Flink 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 \(Console\)](#)
- [Create and run the Application \(AWS CLI\)](#)

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 Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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 1.19**.
4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
 5. Choose **Create application**.

Note

When you create a Managed Service 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: `kinesisanalytics-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/amazon-msf-java-stream-
app-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 **amazon-msf-java-stream-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 **ProducerConfigProperties**.
5. Enter the following application properties and values:

Group ID	Key	Value
FlinkApplicationPr operties	flink.inputstream. initpos	LATEST
FlinkApplicationPr operties	aws.region	us-west-2
FlinkApplicationPr operties	AggregationEnabled	false

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, Managed Service for Apache Flink 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 Managed Service for Apache Flink application. Managed Service for Apache Flink uses the `kinesisanalyticsv2` AWS CLI command to create and interact with Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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. Managed Service for Apache Flink 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 Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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 Managed Service for Apache Flink permission to assume the role, and the permissions policy determines what Managed Service for Apache Flink 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 **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-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, Managed Service for Apache Flink 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"](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the **AKReadSourceStreamWriteSinkStream** 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 Managed Service for Apache Flink 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_19",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "amazon-msf-java-stream-app-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 kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

The application is now running. You can check the Managed Service for Apache Flink 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”](#).

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"](#) section.

```
{
  "ApplicationName": "test",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationCodeConfigurationUpdate": {
      "CodeContentUpdate": {
        "S3ContentLocationUpdate": {
          "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
          "FileKeyUpdate": "amazon-msf-java-stream-app-1.0.jar",
          "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhyvpDU"
        }
      }
    }
  }
}
```

Next step

[Step 4: Clean up AWS resources](#)

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)
- [Next Step](#)

Delete your Managed Service for Apache Flink application

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. In the Managed Service for Apache Flink panel, choose **MyApplication**.
3. In the application's page, choose **Delete** and then confirm the deletion.

Delete your Kinesis data streams

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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: Use resources to complete next steps](#)

Step 5: Use resources to complete next steps

Now that you've created and run a basic Managed Service for Apache Flink application, see the following resources for more advanced Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink for Apache Flink applications for stream processing.
- [Amazon Managed Service for Apache Flink 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.
- [Managed Service for Apache Flink: Examples](#): This section of this Developer Guide provides examples of creating and working with applications in Managed Service for Apache Flink. They include example code and step-by-step instructions to help you create Managed Service for Apache Flink 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.

Getting started with Amazon Managed Service for Apache Flink (Table API)

This section introduces you to the fundamental concepts of Managed Service 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 the Managed Service for Apache Flink application](#)
- [Prerequisites](#)
- [Create and run a Managed Service for Apache Flink application](#)
- [Clean up AWS resources](#)
- [Next steps](#)

Components of the Managed Service for Apache Flink application

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

Managed Service for Apache Flink 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](#).
- **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 Firehose Firehose stream, an Amazon MSK topic, an Amazon S3 bucket, and so on. For more information, see [Table API sinks](#).

After you create, compile, and package your application code, you upload the code package to an Amazon S3 bucket. You then create a Managed Service for Apache Flink 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 Managed Service for Apache Flink \(DataStream API\)](#):

- [Step 1: Set up an AWS account and create an administrator user](#)
- [Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)

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

Create and run a Managed Service for Apache Flink application

In this exercise, you create a Managed Service for Apache Flink 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](#)
- [Write samplerRecords to the input stream](#)
- [Download and examine the Apache Flink streaming Java code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Next step](#)

Create dependent resources

Before you create a Managed Service for Apache Flink 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-code-*<username>*)

Create a VPC and an Amazon MSK cluster

To create a VPC and Amazon MSK cluster to access from your Managed Service for Apache Flink 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-code-*<username>***.

Other resources

When you create your application, Managed Service for Apache Flink creates the following Amazon CloudWatch resources if they don't already exist:

- A log group called `/AWS/KinesisAnalytics-java/MyApplication`.

- A log stream called `kinesis-analytics-log-stream`.

Write `samplerRecords` 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 = "b-1.stocks.8e6izk.c12.kafka.us-
east-1.amazonaws.com:9092,b-2.stocks.8e6izk.c12.kafka.us-east-1.amazonaws.com:9092"
BROKERS = "localhost:9092"
producer = KafkaProducer(
    bootstrap_servers=BROKERS,
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    key_serializer=str.encode,
    retry_backoff_ms=500,
    request_timeout_ms=20000,
    security_protocol='PLAINTEXT')

def getReferrer():
    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 =getReferrer()
    # print(data)
    try:
        future = producer.send("stocktopic", value=data,key=data['ticker'])
        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-examples.git
```

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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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](#).

Note

When building your application, we strongly advise creating and running the Managed Service for Apache Flink application in the same Region as the Amazon MSK cluster. This

is because the Flink Kafka connector is by default optimized for low latency environment. If you need to consume from a cross Region Kafka cluster, consider increasing the configuration value for `receive.buffer.byte`, such as 2097152. For more information, see [Custom MSK configurations](#).

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 [Fulfill the prerequisites for completing the exercises](#).

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=1.19.1
```

- 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 Managed Service for Apache Flink application

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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**.

- Keep the version as **Apache Flink version 1.19.1**.
4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
 5. Choose **Create application**.

Note

When you create a Managed Service 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: `kinesisanalytics-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": "S3",
      "Effect": "Allow",
      "Action": [
        "s3:Abort*",
        "s3:DeleteObject*",
        "s3:GetObject*",
```

```

        "s3:GetBucket*",
        "s3:List*",
        "s3:ListBucket",
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::ka-app-code-<username>",
        "arn:aws:s3:::ka-app-code-<username>/*"
    ]
},
{
    "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"
    ]
}
]

```

}

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**.
5. Enter the following:

Group ID	Key	Value
FlinkApplicationPr operties	kafka-topic	AWSKafkaTutorialTo pic
FlinkApplicationPr operties	brokers	<i>Your Amazon MSK cluster's Bootstrap Brokers list</i>
FlinkApplicationPr operties	s3Path	ka-app-co de- <i><username></i>
FlinkApplicationPr operties	security.protocol	SSL
FlinkApplicationPr operties	ssl.truststore.loc ation	/usr/lib/jvm/java- 11-amazon-corretto /lib/security/cace rts

Group ID	Key	Value
FlinkApplicationPr operties	ssl.truststore.pas sword	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, Managed Service for Apache Flink 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](#)

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 Managed Service for Apache Flink application](#)
- [Delete your Amazon MSK cluster](#)
- [Delete your VPC](#)
- [Delete your Amazon S3 objects and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)
- [Next step](#)

Delete your Managed Service for Apache Flink 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 Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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](#)

Next steps

Now that you've created and run a Managed Service for Apache Flink application that uses the Table API, see [Step 5: Use resources to complete next steps](#) in the [Getting started with Amazon Managed Service for Apache Flink \(DataStream API\)](#).

Getting started with Amazon Managed Service for Apache Flink for Python

This section introduces you to the fundamental concepts of a Managed Service 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

- [Getting started with Pyflink - The Python Interpreter for Apache | Amazon Web Services](#)
- [Components of a Managed Service for Apache Flink application](#)
- [Prerequisites](#)
- [Create and run a Managed Service for Apache Flink for Python application](#)
- [Clean up AWS resources](#)

Getting started with Pyflink - The Python Interpreter for Apache | Amazon Web Services

Before you begin, we encourage you to watch the following video:

[Getting started with Pyflink - The Python Interpreter for Apache | Amazon Web Services](#)

Components of a Managed Service for Apache Flink application

To process data, your Managed Service for Apache Flink application uses a Python application that processes input and produces output using the Apache Flink runtime.

Managed Service for Apache Flink 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](#).

- **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 Firehose Firehose stream, an Amazon MSK topic, an Amazon S3 bucket, and so on. For more information, see [Table API sinks](#).

After you create and package your application code, you upload the code package to an Amazon S3 bucket. You then create a Managed Service for Apache Flink 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 Managed Service for Apache Flink \(DataStream API\)](#):

- [Step 1: Set up an AWS account and create an administrator user](#)
- [Step 2: Set up the AWS Command Line Interface \(AWS CLI\)](#)

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

Create and run a Managed Service for Apache Flink for Python application

In this exercise, you create a Managed Service for Apache Flink application for Python application with a Kinesis stream as a source and a sink.

This section contains the following steps.

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Create and examine the Apache Flink streaming Python code](#)
- [Adding third-party dependencies to Python apps](#)
- [Upload the Apache Flink streaming Python code](#)
- [Create and run the Managed Service for Apache Flink application](#)

- [Next step](#)

Create dependent resources

Before you create a Managed Service for Apache Flink 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-code-*<username>*)

Create two Kinesis streams

Before you create a Managed Service for Apache Flink 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-code-*<username>***.

Other resources

When you create your application, Managed Service for Apache Flink creates the following Amazon CloudWatch resources if they don't already exist:

- A log group called `/AWS/KinesisAnalytics-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

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', region_name='us-west-2'))
```

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-examples.git
```

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(output_table_name, output_stream, output_region)
```

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 = None):  
    init_pos = "\n'scan.stream.initpos' = '{0}',".format(stream_initpos) if  
    stream_initpos is not None else ''  
  
    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}',{3}  
        'format' = 'json',  
        'json.timestamp-format.standard' = 'ISO-8601'  
    ) """.format(table_name, stream_name, region, init_pos)  
}
```

- 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)
)

# 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_2.12/1.15.2](#) file.

Adding third-party dependencies to Python apps

When using third-party python packages (such as [boto3](#)), you will need to add their transitive dependencies and the properties required to target these dependencies. At a high level, for PyPi dependencies, you can copy the files and folders that are located within your python environments `site-packages` folder to a create a directory structure like below:

```
PythonPackages
#  README.md
#  python-packages.py
#
####my_deps
    ####boto3
    #  #  session.py
    #  #  utils.py
    #  #  ...
    #
    ####botocore
    #  #  args.py
    #  #  auth.py
    #  ...
    ####mynonpypimodule
    #  #  mymodulefile1.py
```

```
# # mymodulefile2.py
# # ...
####lib
# # flink-sql-connector-kinesis-4.2.0-1.18.jar
# # ...
# # ...
```

To add the the boto3 as a third-party dependency:

1. Create a standalone Python environment (conda or similar) on your local machine with the required dependencies.
2. Note the initial list of packages in that environment's `site_packages` folder.
3. `pip-install` all required dependencies for your app.
4. Note the packages that were added to the `site_packages` folder after step 3 above. These are the folders you need to include in your package (under the `my_deps` folder), organized as shown above. This will allow you to capture a *diff* of the packages between steps 2 and 3 to identify the right package dependencies for your application.
5. Supply `my_deps/` as an argument for the `pyFiles` property in the `kinesis.analytics.flink.run.options` property group as described below for the `jarfiles` property. Flink also allows you to specify Python dependencies using the [add_python_file](#) function, but it's important to keep in mind that you only need to specify one or the other – not both.

Note

You don't have to name the folder `my_deps`. The important part is registering the dependencies using either `pyFiles` or `add_python_file`. An example can be found at [How to use boto3 within pyFlink](#).

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 `getting-started.py` and [Flink SQL connector](#) 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:

Note

Do not use the compress features in Finder (macOS) or Windows Explorer (Windows) to create the `myapp.zip` archive. This may result in invalid application code.

1. Use your preferred compression application to compress the `streaming-file-sink.py` and [Flink SQL connector](#) files.

Note

Do not use the compress features in Finder (macOS) or Windows Explorer (Windows) to create the `myapp.zip` archive. This may result in invalid application code.

2. Use your preferred compression application to compress the `getting-started.py` and [Flink SQL connector](#) 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.

3. 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 Managed Service for Apache Flink application

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

Create the application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - 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 1.18**.
4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
5. Choose **Create application**.

Note

When you create a Managed Service 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: `kinesisanalytics-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**.
5. Enter the following:

Group ID	Key	Value
<code>consumer.config.0</code>	<code>input.stream.name</code>	<code>ExampleInputStream</code>
<code>consumer.config.0</code>	<code>aws.region</code>	<code>us-west-2</code>
<code>consumer.config.0</code>	<code>scan.stream.initpos</code>	<code>LATEST</code>

Choose **Save**.

6. Under **Properties**, choose **Add group** again.
7. Enter the following:

Group ID	Key	Value
<code>producer.config.0</code>	<code>output.stream.name</code>	<code>ExampleOutputStream</code>
<code>producer.config.0</code>	<code>aws.region</code>	<code>us-west-2</code>

Group ID	Key	Value
<code>producer.config.0</code>	<code>shard.count</code>	<code>1</code>

- Under **Properties**, choose **Add group** again. For **Group ID**, enter `flink.sql.connector.kinesis.options`. This special property group tells your application where to find its code resources. For more information, see [Specifying your code files](#).
- Enter the following:

Group ID	Key	Value
<code>kinesis.analytics.flink.run.options</code>	<code>python</code>	<code>getting-started.py</code>
<code>kinesis.analytics.flink.run.options</code>	<code>jarfile</code>	<code>flink-sql-connector-kinesis-4.2.0-1.18.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, Managed Service for Apache Flink 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

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/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](#)

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 objects and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink 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 Managed Service for Apache Flink panel, choose **MyApplication**.
3. On the application page, choose **Delete** and then confirm the deletion.

Delete your Kinesis data streams

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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.

Getting started (Scala)

Note

Starting from version 1.15, Flink is Scala free. Applications can now use the Java API from any Scala version. Flink still uses Scala in a few key components internally, but doesn't expose Scala into the user code classloader. Because of that, you must add Scala dependencies into your JAR-archives.

For more information about Scala changes in Flink 1.15, see [Scala Free in One Fifteen](#).

In this exercise, you create a Managed Service for Apache Flink application application for Scala with a Kinesis stream as a source and a sink.

This topic contains the following sections:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the application code](#)
- [Compile and upload the application code](#)
- [Create and run the application \(console\)](#)
- [Create and run the application \(CLI\)](#)
- [Clean up AWS resources](#)

Create dependent resources

Before you create a Managed Service 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 (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**.

To create the data streams (AWS CLI)

- 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
```

- 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
```

- [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>***.

Other resources

When you create your application, Managed Service for Apache Flink creates the following Amazon CloudWatch resources if they don't already exist:

- A log group called `/AWS/KinesisAnalytics-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

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', region_name='us-west-2'))
```

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-examples.git
```

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

Note the following about the application code:

- A `build.sbt` file contains information about the application's configuration and dependencies, including the Managed Service for Apache Flink libraries.
- The `BasicStreamingJob.scala` 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:

```
private def createSource: FlinkKinesisConsumer[String] = {  
    val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties  
    val inputProperties = applicationProperties.get("ConsumerConfigProperties")  
  
    new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,  
    defaultInputStreamName),  
    new SimpleStringSchema, inputProperties)
```

```
}
```

The application also uses a Kinesis sink to write into the result stream. The following snippet creates the Kinesis sink:

```
private def createSink: KinesisStreamsSink[String] = {  
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties  
  val outputProperties = applicationProperties.get("ProducerConfigProperties")  
  
  KinesisStreamsSink.builder[String]  
    .setKinesisClientProperties(outputProperties)  
    .setSerializationSchema(new SimpleStringSchema)  
    .setStreamName(outputProperties.getProperty(streamNameKey,  
defaultOutputStreamName))  
    .setPartitionKeyGenerator((element: String) => String.valueOf(element.hashCode))  
    .build  
}
```

- The application creates source and sink connectors to access external resources using a `StreamExecutionEnvironment` object.
- The application creates source and sink connectors using dynamic application properties. Runtime application's properties are read to configure the connectors. For more information about runtime properties, see [Runtime Properties](#).

Compile and upload the application code

In this section, you compile and upload your application code to the Amazon S3 bucket you created in the [Create dependent resources](#) section.

Compile the Application Code

In this section, you use the [SBT](#) build tool to build the Scala code for the application. To install SBT, see [Install sbt with cs setup](#). You also need to install the Java Development Kit (JDK). See [Prerequisites for Completing the Exercises](#).

1. To use your application code, you compile and package it into a JAR file. You can compile and package your code with SBT:

```
sbt assembly
```

2. If the application compiles successfully, the following file is created:

```
target/scala-3.2.0/getting-started-scala-1.0.jar
```

Upload the Apache Flink Streaming Scala Code

In this section, you create an Amazon S3 bucket and upload your 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 **Configure options**, keep the settings as they are, and choose **Next**.
5. In **Set permissions**, keep the settings as they are, and choose **Next**.
6. Choose **Create bucket**.
7. Choose the `ka-app-code-<username>` bucket, and then choose **Upload**.
8. In the **Select files** step, choose **Add files**. Navigate to the `getting-started-scala-1.0.jar` file that you created in the previous step.
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 application (console)

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

Create the Application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.

- For **Description**, enter **My scala test app**.
 - For **Runtime**, choose **Apache Flink**.
 - Keep the version as **Apache Flink version 1.19.1**.
4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
 5. Choose **Create application**.

Note

When you create a Managed Service 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: `kinesisanalytics-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 `getting-started-scala-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 **Add group**.
5. Enter the following:

Group ID	Key	Value
ConsumerConfigProperties	input.stream.name	ExampleInputStream
ConsumerConfigProperties	aws.region	us-west-2
ConsumerConfigProperties	flink.stream.initpos	LATEST

Choose **Save**.

- Under **Properties**, choose **Add group** again.
- Enter the following:

Group ID	Key	Value
ProducerConfigProperties	output.stream.name	ExampleOutputStream
ProducerConfigProperties	aws.region	us-west-2

- 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, Managed Service for Apache Flink 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

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/getting-started-scala-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"
    }
]
}

```

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.

Create and run the application (CLI)

In this section, you use the AWS Command Line Interface to create and run the Managed Service for Apache Flink application. Use the *kinesisanalyticsv2* AWS CLI command to create and interact with Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 `AKReadSourceStreamWriteSinkStream` 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": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
```

```

        "arn:aws:s3:::ka-app-code-username/getting-started-scala-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"
    }
  ]
}
```

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*.

Create an IAM policy

In this section, you create an IAM role that the Managed Service for Apache Flink application can assume to read a source stream and write to the sink stream.

Managed Service for Apache Flink 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 Managed Service for Apache Flink permission to assume the role, and the permissions policy determines what Managed Service for Apache Flink 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** and then **Create Role**.
3. Under **Select type of trusted identity**, choose **AWS Service**
4. Under **Choose the service that will use this role**, choose **Kinesis**.
5. Under **Select your use case**, choose **Managed Service for Apache Flink**.
6. Choose **Next: Permissions**.
7. On the **Attach permissions policies** page, choose **Next: Review**. You attach permissions policies after you create the role.
8. On the **Create role** page, enter **MF-stream-rw-role** for the **Role name**. Choose **Create role**.

Now you have created a new IAM role called `MF-stream-rw-role`. Next, you update the trust and permissions policies for the role

9. Attach the permissions policy to the role.

Note

For this exercise, Managed Service for Apache Flink 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, [Create a Permissions Policy](#).

- a. On the **Summary** page, choose the **Permissions** tab.
- b. Choose **Attach Policies**.
- c. In the search box, enter **AKReadSourceStreamWriteSinkStream** (the policy that you created in the previous section).
- d. Choose the **AKReadSourceStreamWriteSinkStream** 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 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": "getting_started",
  "ApplicationDescription": "Scala getting started application",
  "RuntimeEnvironment": "FLINK-1_19",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
```

```

        "FileKey": "getting-started-scala-1.0.jar"
      }
    },
    "CodeContentType": "ZIPFILE"
  },
  "EnvironmentProperties": {
    "PropertyGroups": [
      {
        "PropertyGroupId": "ConsumerConfigProperties",
        "PropertyMap" : {
          "aws.region" : "us-west-2",
          "stream.name" : "ExampleInputStream",
          "flink.stream.initpos" : "LATEST"
        }
      },
      {
        "PropertyGroupId": "ProducerConfigProperties",
        "PropertyMap" : {
          "aws.region" : "us-west-2",
          "stream.name" : "ExampleOutputStream"
        }
      }
    ]
  }
},
"CloudWatchLoggingOptions": [
  {
    "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
  }
]
}

```

Execute the [CreateApplication](#) with the following 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": "getting_started",
  "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 Managed Service for Apache Flink 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": "s3_sink"
}
```

2. Execute the `StopApplication` action with the preceding 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 [Setting Up Application Logging](#).

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": "getting_started",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
            "stream.name" : "ExampleInputStream",
            "flink.stream.initpos" : "LATEST"
          }
        },
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2",
            "stream.name" : "ExampleOutputStream"
          }
        }
      ]
    }
  }
}
```


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](#) 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 [Create dependent resources](#) section.

```
{
  "ApplicationName": "getting_started",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationCodeConfigurationUpdate": {
      "CodeContentUpdate": {
        "S3ContentLocationUpdate": {
          "BucketARNUpdate": "arn:aws:s3:::ka-app-code-<username>",
          "FileKeyUpdate": "getting-started-scala-1.0.jar",
          "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhyvDU"
        }
      }
    }
  }
}
```

```
}  
  }  
}
```

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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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.

Creating Managed Service for Apache Flink applications with Apache Beam

Note

Apache Beam is not supported in Apache Flink version 1.19. As of June 27, 2024, there is no compatible Apache Flink Runner for Flink 1.18. For more information, see [Flink Version Compatibility](#) in the Apache Beam Documentation.>

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

For a tutorial about how to use Apache Beam in a Managed Service for Apache Flink application, see [Using CloudFormation with Managed Service for Apache Flink](#).

This topic contains the following sections:

- [Using Apache Beam with Managed Service for Apache Flink](#)
- [Beam capabilities](#)
- [Creating an application using Apache Beam](#)

Using Apache Beam with Managed Service for Apache Flink

Note the following about using the Apache Flink runner with Managed Service for Apache Flink:

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

Beam capabilities

Managed Service for Apache Flink 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 Managed Service for Apache Flink service to verify that we support all the features that your application needs.

Creating an application using Apache Beam

In this exercise, you create a Managed Service for Apache Flink application that transforms data using [Apache Beam](#). Apache Beam is a programming model for processing streaming data. For information about using Apache Beam with Managed Service for Apache Flink, see [Using Apache Beam](#).

Note

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

This topic contains the following sections:

- [Create dependent resources](#)
- [Write sample records to the input stream](#)
- [Download and examine the application code](#)
- [Compile the application code](#)
- [Upload the Apache Flink streaming Java code](#)
- [Create and run the Managed Service for Apache Flink application](#)
- [Clean up AWS resources](#)
- [Next steps](#)

Create dependent resources

Before you create a Managed Service for Apache Flink 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-examples.git
```

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()))
```

- Managed Service for Apache Flink 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](#) in the [Getting started \(DataStream API\)](#) tutorial.
2. Compile the application with the following command:

```
mvn package -Dflink.version=1.15.2 -Dflink.version.minor=1.8
```

Note

The provided source code relies on libraries from Java 11.

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](#) 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 Managed Service for Apache Flink application

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

Create the Application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. On the Managed Service for Apache Flink dashboard, choose **Create analytics application**.
3. On the **Managed Service for Apache Flink - Create application** page, provide the application details as follows:
 - For **Application name**, enter **MyApplication**.
 - For **Runtime**, choose **Apache Flink**.

Note

Apache Beam is not presently compatible with Apache Flink version 1.19 or later.

- Select **Apache Flink version 1.15** from the version pulldown.
4. For **Access permissions**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-west-2**.
 5. Choose **Create application**.

Note

When you create a Managed Service 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 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. Enter the following:

Group ID	Key	Value
BeamApplicationProperties	InputStreamName	ExampleInputStream
BeamApplicationProperties	OutputStreamName	ExampleOutputStream
BeamApplicationProperties	AwsRegion	us-west-2

5. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
6. For **CloudWatch logging**, select the **Enable** check box.
7. Choose **Update**.

Note

When you choose to enable CloudWatch logging, Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application](#)
- [Delete your Kinesis data streams](#)
- [Delete your Amazon S3 object and bucket](#)
- [Delete your IAM resources](#)
- [Delete your CloudWatch resources](#)

Delete your Managed Service for Apache Flink application

1. Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
2. in the Managed Service for Apache Flink 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-us-west-2** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-west-2** 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 Managed Service for Apache Flink application that transforms data using Apache Beam, see the following application for an example of a more advanced Managed Service for Apache Flink solution.

- [Beam on Managed Service for Apache Flink Streaming Workshop](#): In this workshop, we explore an end to end example that combines batch and streaming aspects in one uniform Apache Beam pipeline.

Training workshops, labs, and solution implementations

The following end-to-end examples demonstrate advanced Managed Service for Apache Flink solutions.

Topics

- [Deploy, operate, and scale applications with Amazon Managed Service for Apache Flink](#)
- [Develop Apache Flink applications locally before deploying to Managed Service for Apache Flink](#)
- [Use event detection with Managed Service for Apache Flink Studio](#)
- [Use the AWS Streaming data solution for Amazon Kinesis](#)
- [Practice using a Clickstream lab with Apache Flink and Apache Kafka](#)
- [Set up custom scaling using Application Auto Scaling](#)
- [View a sample Amazon CloudWatch dashboard](#)
- [Use templates for AWS Streaming data solution for Amazon MSK](#)
- [Explore more Managed Service for Apache Flink solutions on GitHub](#)

Deploy, operate, and scale applications with Amazon Managed Service for Apache Flink

This workshop covers the development an Apache Flink application in Java, how to run and debug in your IDE, and how to package, deploy and run on Amazon Managed Service for Apache Flink. You will also learn how to scale, monitor, and troubleshoot your application.

The workshop can be found here: [Amazon Managed Service for Apache Flink workshop](#).

Develop Apache Flink applications locally before deploying to Managed Service for Apache Flink

This workshop will show you the basics of getting up and started developing Apache Flink applications locally with the long term goal of deploying to Managed Service for Apache Flink for Apache Flink.

The solution can be found here: [Starters Guide to Local Development with Apache Flink](#)

Use event detection with Managed Service for Apache Flink Studio

This workshop describes event detection with Managed Service for Apache Flink Studio and deploying it as a Managed Service for Apache Flink application

The solution can be found here: [Event Detection with Managed Service for Apache Flink for Apache Flink](#)

Use 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 Managed Service for Apache Flink 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:

- An 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](#)

Practice using a 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 Managed Service for Apache Flink for Apache Flink applications for stream processing.

The solution can be found here: [Clickstream Lab](#)

Set up custom scaling using Application Auto Scaling

A sample that helps users automatically scale their Managed Service for Apache Flink applications using Application Auto Scaling. This enables users to set up custom scaling policies and custom scaling attributes.

The solutions can be found here:

- [Managed Service for Apache Flink App Autoscaling](#)
- [Scheduled Scaling](#)

For more information on you can perform custom scaling, see [Enable metric-based and scheduled scaling for Amazon Managed Service for Apache Flink](#).

View a sample Amazon CloudWatch dashboard

A sample CloudWatch dashboard for monitoring Managed Service for Apache Flink applications. The sample dashboard also includes a [demo application](#) to help with demonstrating the functionality of the dashboard.

The solution can be found here: [Managed Service for Apache Flink Metrics Dashboard](#)

Use templates for 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](#)

Explore more Managed Service for Apache Flink solutions on GitHub

The following end-to-end examples demonstrate advanced Managed Service for Apache Flink solutions and are available on GitHub:

- [Amazon Managed Service for Apache Flink Flink – Benchmarking Utility](#)

- [Snapshot Manager – Amazon Managed Service for Apache Flink for Apache Flink](#)
- [Streaming ETL with Apache Flink and Amazon Managed Service for Apache Flink](#)
- [Real-time sentiment analysis on customer feedback](#)

Utilities

The following utilities can make using the Managed Service for Apache Flink service easier to use:

Topics

- [Snapshot manager](#)
- [Benchmarking](#)

Snapshot manager

It's a best practice for Flink Applications to regularly trigger savepoints/snapshots to allow for more seamless failure recovery. Snapshot manager automates this task and offers the following benefits:

- takes a new snapshot of a running Managed Service for Apache Flink for Apache Flink Application
- gets a count of application snapshots
- checks if the count is more than the required number of snapshots
- deletes older snapshots that are older than the required number

For an example, see [Snapshot manager](#).

Benchmarking

Managed Service for Apache Flink Flink Benchmarking Utility helps with capacity planning, integration testing, and benchmarking of Managed Service for Apache Flink for Apache Flink applications.

For an example, see [Benchmarking](#)

Managed Service for Apache Flink: Examples

This section provides examples of creating and working with applications in Managed Service for Apache Flink. They include example code and step-by-step instructions to help you create Managed Service for Apache Flink applications and test your results.

Before you explore these examples, we recommend that you first review the following:

- [How it works](#)
- [Getting started \(DataStream API\)](#)

Note

These examples assume that you are using the US East (N. Virginia) Region (us-east-1). If you are using a different Region, update your application code, commands, and IAM roles appropriately.

Topics

- [Java examples](#)
- [Python examples](#)
- [Scala examples](#)

Java examples

The following examples demonstrate how to create applications written in Java.

Note

Most of the examples are designed to run both locally, on your development machine and your IDE of choice, and on Amazon Managed Service for Apache Flink. They demonstrate the mechanisms that you can use to pass application parameters, and how to set the dependency correctly to run the application in both environments with no changes.

Getting started with the DataStream API

This example shows a simple application, reading from a Kinesis data stream and writing to another Kinesis data stream, using the `DataStream` API. The example demonstrates how to set up the file with the correct dependencies, build the uber-JAR, and then parse the configuration parameters, so you can run the application both locally, in your IDE, and on Amazon Managed Service for Apache Flink.

Code example: [GettingStarted](#)

Getting started with the Table API and SQL

This example shows a simple application using the `Table` API and SQL. It demonstrates how to integrate the `DataStream` API with the `Table` API or SQL in the same Java application. It also demonstrates how to use the `DataGen` connector to generate random test data from within the Flink application itself, not requiring an external data generator.

Complete example: [GettingStartedTable](#)

Using S3Sink (DataStream API)

This example demonstrates how to use the `DataStream` API's `FileSink` to write JSON files to an S3 bucket.

Code example: [S3Sink](#)

Using a Kinesis source, standard or EFO consumers, and sink (DataStream API)

This example demonstrates how to configure a source consuming from a Kinesis data stream, either using the standard consumer or EFO, and how to set up a sink to the Kinesis data stream.

Code example: [KinesisConnectors](#)

Using an Amazon Data Firehose sink (DataStream API)

This example shows how to send data to Amazon Data Firehose (formerly known as Kinesis Data Firehose).

Code example: [KinesisFirehoseSink](#)

Using windowing aggregations (DataStream API)

This example demonstrates four types of the windowing aggregation in the `DataStream` API.

1. Sliding Window based on processing time
2. Sliding Window based on event time
3. Tumbling Window based on processing time
4. Tumbling Window based on event time

Code example: [Windowing](#)

Using custom metrics

This example shows how to add custom metrics to your Flink application and send them to CloudWatch metrics.

Code examples: [CustomMetrics](#)

Python examples

The following examples demonstrate how to create applications written in Python.

Note

Most of the examples are designed to run both locally, on your development machine and your IDE of choice, and on Amazon Managed Service for Apache Flink. They demonstrate the simple mechanism that you can use to pass application parameters, and how to set the dependency correctly to run the application in both environments with no changes.

Project dependencies

Most PyFlink examples require one or more dependencies in the form of JAR files, for example for Flink connectors. These dependencies must then be packaged with the application when deployed on Amazon Managed Service for Apache Flink.

The following examples already include the tooling that lets you run the application locally for development and testing, and to package the required dependencies correctly. This tooling requires using Java JDK11 and Apache Maven. Refer to the README contained in each example for the specific instructions.

Examples

Getting started with PyFlink

This example demonstrates the basic structure of a PyFlink application using SQL embedded in Python code. This project also provides a skeleton for any PyFlink application that includes JAR dependencies such as connectors. The README section provides detailed guidance about how to run your Python application locally for development. The example also shows how to include a single JAR dependency, the Kinesis SQL connector in this example, in your PyFlink application.

Code example: [GettingStarted](#)

Using windowing aggregations (DataStream API)

This example demonstrates four types of the windowing aggregation in SQL embedded in a Python application.

1. Sliding Window based on processing time
2. Sliding Window based on event time
3. Tumbling Window based on processing time
4. Tumbling Window based on event time

Code example: [Windowing](#)

Using an S3 sink

This example shows how to write your output to Amazon S3 as JSON files, using SQL embedded in a Python application. You must enable checkpointing for the S3 sink to write and rotate files to Amazon S3.

Code example: [S3Sink](#)

Using a User Defined Function (UDF)

This example demonstrates how to define a User Defined Function, implement it in Python, and use it in SQL code that runs in a Python application.

Code example: [UDF](#)

Using an Amazon Data Firehose sink

This example demonstrates how to send data to Amazon Data Firehose using SQL.

Code example: [FirehoseSink](#)

Scala examples

The following examples demonstrate how to create applications using Scala with Apache Flink.

Multi-step application

This example shows how to set up a Flink application in Scala. It demonstrates how to configure the SBT project to include dependencies and build the uber-JAR.

Code example: [GettingStarted](#)

Security in Amazon Managed Service for Apache Flink

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 Managed Service for Apache Flink, 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 Managed Service for Apache Flink. The following topics show you how to configure Managed Service for Apache Flink 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 Managed Service for Apache Flink resources.

Topics

- [Data protection in Amazon Managed Service for Apache Flink](#)
- [Identity and Access Management for Amazon Managed Service for Apache Flink](#)
- [Monitoring Managed Service for Apache Flink](#)
- [Compliance validation for Amazon Managed Service for Apache Flink](#)
- [Resilience in Amazon Managed Service for Apache Flink](#)
- [Infrastructure security in Managed Service for Apache Flink](#)
- [Security best practices for Managed Service for Apache Flink](#)

Data protection in Amazon Managed Service for Apache Flink

You can protect your data using tools that are provided by AWS. Managed Service for Apache Flink can work with services that support encrypting data, including Firehose, and Amazon S3.

Data encryption in Managed Service for Apache Flink

Encryption at rest

Note the following about encrypting data at rest with Managed Service 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 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 \(SSE-KMS\)](#).
- Managed Service for Apache Flink 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

Managed Service for Apache Flink encrypts all data in transit. Encryption in transit is enabled for all Managed Service for Apache Flink applications and cannot be disabled.

Managed Service for Apache Flink encrypts data in transit in the following scenarios:

- Data in transit from Kinesis Data Streams to Managed Service for Apache Flink.
- Data in transit between internal components within Managed Service for Apache Flink.
- Data in transit between Managed Service for Apache Flink and Firehose.

Key management

Data encryption in Managed Service for Apache Flink uses service-managed keys. Customer-managed keys are not supported.

Identity and Access Management for Amazon Managed Service for Apache Flink

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Managed Service for Apache Flink resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How Amazon Managed Service for Apache Flink works with IAM](#)
- [Identity-based policy examples for Amazon Managed Service for Apache Flink](#)
- [Troubleshooting Amazon Managed Service for Apache Flink identity and access](#)
- [Cross-service confused deputy prevention](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Managed Service for Apache Flink.

Service user – If you use the Managed Service for Apache Flink service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Managed Service for Apache Flink features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Managed Service for Apache Flink, see [Troubleshooting Amazon Managed Service for Apache Flink identity and access](#).

Service administrator – If you're in charge of Managed Service for Apache Flink resources at your company, you probably have full access to Managed Service for Apache Flink. It's your job to determine which Managed Service for Apache Flink features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Managed Service for Apache Flink, see [How Amazon Managed Service for Apache Flink works with IAM](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Managed Service for Apache Flink. To view example Managed Service for Apache Flink identity-based policies that you can use in IAM, see [Identity-based policy examples for Amazon Managed Service for Apache Flink](#).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [Signing AWS API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Multi-factor authentication](#) in the

AWS IAM Identity Center User Guide and [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An [IAM group](#) is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier

to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Creating a role for a third-party Identity Provider](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permission sets, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).
- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How Amazon Managed Service for Apache Flink works with IAM

Before you use IAM to manage access to Managed Service for Apache Flink, learn what IAM features are available to use with Managed Service for Apache Flink.

IAM features you can use with Amazon Managed Service for Apache Flink

IAM feature	Managed Service for Apache Flink support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	No
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	No

To get a high-level view of how Managed Service for Apache Flink and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Identity-based policies for Managed Service for Apache Flink

Supports identity-based policies	Yes
----------------------------------	-----

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Identity-based policy examples for Managed Service for Apache Flink

To view examples of Managed Service for Apache Flink identity-based policies, see [Identity-based policy examples for Amazon Managed Service for Apache Flink](#).

Resource-based policies within Managed Service for Apache Flink

Supports resource-based policies	Yes
----------------------------------	-----

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant

the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Policy actions for Managed Service for Apache Flink

Supports policy actions	Yes
-------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Managed Service for Apache Flink actions, see [Actions Defined by Amazon Managed Service for Apache Flink](#) in the *Service Authorization Reference*.

Policy actions in Managed Service for Apache Flink use the following prefix before the action:

```
Kinesis Analytics
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
  "Kinesis Analytics:action1",  
  "Kinesis Analytics:action2"  
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "Kinesis Analytics:Describe*"
```

To view examples of Managed Service for Apache Flink identity-based policies, see [Identity-based policy examples for Amazon Managed Service for Apache Flink](#).

Policy resources for Managed Service for Apache Flink

Supports policy resources	Yes
---------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Managed Service for Apache Flink resource types and their ARNs, see [Resources Defined by Amazon Managed Service for Apache Flink](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by Amazon Managed Service for Apache Flink](#).

To view examples of Managed Service for Apache Flink identity-based policies, see [Identity-based policy examples for Amazon Managed Service for Apache Flink](#).

Policy condition keys for Managed Service for Apache Flink

Supports service-specific policy condition keys	Yes
---	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or *Condition block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

To see a list of Managed Service for Apache Flink condition keys, see [Condition Keys for Amazon Managed Service for Apache Flink](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions Defined by Amazon Managed Service for Apache Flink](#).

To view examples of Managed Service for Apache Flink identity-based policies, see [Identity-based policy examples for Amazon Managed Service for Apache Flink](#).

Access control lists (ACLs) in Managed Service for Apache Flink

Supports ACLs

No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with Managed Service for Apache Flink

Supports ABAC (tags in policies)

Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [What is ABAC?](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

Using Temporary credentials with Managed Service for Apache Flink

Supports temporary credentials	Yes
--------------------------------	-----

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

Cross-service principal permissions for Managed Service for Apache Flink

Supports forward access sessions (FAS)	Yes
--	-----

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles for Managed Service for Apache Flink

Supports service roles	Yes
------------------------	-----

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Warning

Changing the permissions for a service role might break Managed Service for Apache Flink functionality. Edit service roles only when Managed Service for Apache Flink provides guidance to do so.

Service-linked roles for Managed Service for Apache Flink

Supports service-linked roles	Yes
-------------------------------	-----

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see [AWS services that work with IAM](#). Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon Managed Service for Apache Flink

By default, users and roles don't have permission to create or modify Managed Service for Apache Flink resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Creating IAM policies](#) in the *IAM User Guide*.

For details about actions and resource types defined by Managed Service for Apache Flink, including the format of the ARNs for each of the resource types, see [Actions, Resources, and Condition Keys for Amazon Managed Service for Apache Flink](#) in the *Service Authorization Reference*.

Topics

- [Policy best practices](#)
- [Using the Managed Service for Apache Flink console](#)
- [Allow users to view their own permissions](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Managed Service for Apache Flink resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Configuring MFA-protected API access](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the Managed Service for Apache Flink console

To access the Amazon Managed Service for Apache Flink console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Managed Service for Apache Flink resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the Managed Service for Apache Flink console, also attach the Managed Service for Apache Flink ConsoleAccess or ReadOnly AWS managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

Troubleshooting Amazon Managed Service for Apache Flink identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Managed Service for Apache Flink and IAM.

Topics

- [I am not authorized to perform an action in Managed Service for Apache Flink](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my Managed Service for Apache Flink resources](#)

I am not authorized to perform an action in Managed Service for Apache Flink

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` user tries to use the console to view details about a fictional `my-example-widget` resource but does not have the fictional Kinesis Analytics:`GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: Kinesis Analytics:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-example-widget` resource using the Kinesis Analytics:`GetWidget` action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Managed Service for Apache Flink.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Managed Service for Apache Flink. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Managed Service for Apache Flink resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Managed Service for Apache Flink supports these features, see [How Amazon Managed Service for Apache Flink works with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Cross-service confused deputy prevention

In AWS, cross-service impersonation can occur when one service (the calling service) calls another service (the called service). The calling service can be manipulated to act on another customer's

resources even though it shouldn't have the proper permissions, resulting in the confused deputy problem.

To prevent confused deputies, AWS provides tools that help you protect your data for all services using service principals that have been given access to resources in your account. This section focuses on cross-service confused deputy prevention specific to Managed Service for Apache Flink however, you can learn more about this topic at [The confused deputy problem](#) section of the *IAM User Guide*.

In the context of Managed Service for Apache Flink, we recommend using the [aws:SourceArn](#) and [aws:SourceAccount](#) global condition context keys in your role trust policy to limit access to the role to only those requests that are generated by expected resources.

Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must be the ARN of the resource used by Managed Service for Apache Flink, which is specified with the following format:
`arn:aws:kinesisanalytics:region:account:resource`.

The recommended approach to the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full resource ARN.

If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` key with wildcard characters (*) for the unknown portions of the ARN. For example: `arn:aws:kinesisanalytics::111122223333:*`.

Policies of roles that you provide to Managed Service for Apache Flink as well as trust policies of roles generated for you can make use of these keys.

In order to protect against the confused deputy problem, carry out the following steps:

To protect against the confused deputy problem

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Roles** and then choose the role you want to modify.
3. Choose **Edit trust policy**.

4. On the **Edit trust policy** page, replace the default JSON policy with a policy that uses one or both of the `aws:SourceArn` and `aws:SourceAccount` global condition context keys. See the following example policy:
5. Choose **Update policy**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "kinesisanalytics.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "Account ID"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/my-app"
        }
      }
    }
  ]
}
```

Monitoring Managed Service for Apache Flink

Managed Service for Apache Flink provides monitoring functionality for your applications. For more information, see [Logging and monitoring](#).

Compliance validation for Amazon Managed Service for Apache Flink

Third-party auditors assess the security and compliance of Amazon Managed Service 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 . 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 Managed Service 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 Managed Service 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 on Amazon Web Services](#). 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.

FedRAMP

The AWS FedRAMP Compliance program includes Managed Service for Apache Flink as a FedRAMP-authorized service. If you are a federal or commercial customer, you can use the service to process and store sensitive workloads in the AWS GovCloud (US) Region's authorization boundary with data up to the high impact level, as well as US East (N. Virginia), US East (Ohio), US West (N. California), US West (Oregon) Regions with data up to a moderate level.

You can request access to the AWS FedRAMP Security Packages through the FedRAMP PMO, your AWS Sales Account Manager, or you can download them through AWS Artifact at [AWS Artifact](#).

For more information, see [FedRAMP](#).

Resilience in Amazon Managed Service 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, a Managed Service for Apache Flink offers several features to help support your data resiliency and backup needs.

Disaster recovery

Managed Service for Apache Flink runs in a serverless mode, and takes care of host degradations, Availability Zone availability, and other infrastructure related issues by performing automatic migration. Managed Service for Apache Flink achieves this through multiple, redundant mechanisms. Each Managed Service for Apache Flink application 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. Managed Service for Apache Flink 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, Managed Service for Apache Flink first tries to recover the application within the running Apache Flink cluster using your application's checkpoints, if available.

Managed Service for Apache Flink backs up application state using *Checkpoints* and *Snapshots*:

- *Checkpoints* are backups of application state that Managed Service for Apache Flink 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](#).

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.

Managed Service for Apache Flink encrypts data stored in checkpoints and savepoints.

Infrastructure security in Managed Service for Apache Flink

As a managed service, Managed Service for Apache Flink 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 Managed Service for Apache Flink through the network. All API calls to Managed Service for Apache Flink are secured via Transport Layer Security (TLS) and authenticated via IAM. Clients must support 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 Managed Service for Apache Flink

Amazon Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application must have valid credentials to access resources in other services, such as Kinesis data streams, Firehose streams, or Amazon S3 buckets. You should not store AWS credentials directly in the application or in an Amazon S3 bucket. These are long-term credentials that are not automatically rotated and could have a significant business impact if they are compromised.

Instead, you should use an IAM role to manage temporary credentials for your application to access other resources. When you use a role, you don't have to use long-term credentials to access other resources.

For more information, see the following topics in the *IAM User Guide*:

- [IAM Roles](#)
- [Common Scenarios for Roles: Users, Applications, and Services](#)

Implement server-side encryption in dependent resources

Data at rest and data in transit is encrypted in Managed Service for Apache Flink, and this encryption cannot be disabled. You should implement server-side encryption in your dependent resources, such as Kinesis data streams, Firehose streams, and Amazon S3 buckets. For more information on implementing server-side encryption in dependent resources, see [Data protection](#).

Use CloudTrail to monitor API calls

Managed Service for Apache Flink is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an Amazon service in Managed Service for Apache Flink.

Using the information collected by CloudTrail, you can determine the request that was made to Managed Service for Apache Flink, 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"](#).

Logging and monitoring in Amazon Managed Service for Apache Flink

Monitoring is an important part of maintaining the reliability, availability, and performance of Managed Service for Apache Flink 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 Managed Service for Apache Flink, 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 Managed Service for Apache Flink performance in your environment. You do this by measuring performance at various times and under different load conditions. As you monitor Managed Service for Apache Flink, 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

- [Logging](#)
- [Monitoring](#)
- [Setting up application logging](#)
- [Analyzing logs with CloudWatch Logs Insights](#)
- [Viewing metrics and dimensions in Managed Service for Apache Flink](#)
- [Writing custom messages to CloudWatch Logs](#)
- [Logging Managed Service for Apache Flink API calls with AWS CloudTrail](#)

Logging

Logging is important for production applications to understand errors and failures. However, the logging subsystem needs to collect and forward log entries to CloudWatch Logs. While some logging is fine and desirable, extensive logging can overload the service and cause the Flink application to fall behind. Logging exceptions and warnings is certainly a good idea. But you cannot generate a log message for each and every message that is processed by the Flink application. Flink is optimized for high throughput and low latency, the logging subsystem is not. In case it is really required to generate log output for every processed message, use an additional `DataStream` inside the Flink application and a proper sink to send the data to Amazon S3 or CloudWatch. Do not use the Java logging system for this purpose. Moreover, Managed Service for Apache Flink' `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.

Querying logs with CloudWatch Logs Insights

CloudWatch Logs Insights is a powerful service to query log at scale. Customers should leverage its capabilities to quickly search through logs to identify and mitigate errors during operational events.

The following query looks for exceptions in all task manager logs and orders them according to the time they occurred.

```
fields @timestamp, @message
| filter isPresent(throwableInformation.0) or isPresent(throwableInformation) or
  @message like /(Error|Exception)/
| sort @timestamp desc
```

For other useful queries, see [Example Queries](#).

Monitoring

When running streaming applications in production, you set out to execute the application continuously and indefinitely. It is crucial to implement monitoring and proper alarming of all components not only the Flink application. Otherwise you risk to miss emerging problems early on and only realize an operational event once it is fully unravelling and much harder to mitigate. General things to monitor include:

- Is the source ingesting data?
- Is data read from the source (from the perspective of the source)?
- Is the Flink application receiving data?
- Is the Flink application able to keep up or is it falling behind?
- Is the Flink application persisting data into the sink (from the application perspective)?
- Is the sink receiving data?

More specific metrics should then be considered for the Flink application. This [CloudWatch dashboard](#) provides a good starting point. For more information on what metrics to monitor for production applications, see [Using CloudWatch Alarms with Amazon Managed Service for Apache Flink](#). These metrics include:

- **records_lag_max** and **millisbehindLatest** – If the application is consuming from Kinesis or Kafka, these metrics indicate if the application is falling behind and needs to be scaled in order to keep up with the current load. This is a good generic metric that is easy to track for all kinds of applications. But it can only be used for reactive scaling, i.e., when the application has already fallen behind.
- **cpuUtilization** and **heapMemoryUtilization** – These metrics give a good indication of the overall resource utilization of the application and can be used for proactive scaling unless the application is I/O bound.
- **downtime** – A downtime greater than zero indicates that the application has failed. If the value is larger than 0, the application is not processing any data.
- **lastCheckpointSize** and *lastCheckpointDuration* – These metrics monitor how much data is stored in state and how long it takes to take a checkpoint. If checkpoints grow or take long, the application is continuously spending time on checkpointing and has less cycles for actual processing. At some points, checkpoints may grow too large or take so long that they fail. In addition to monitoring absolute values, customers should also consider monitoring the change rate with `RATE(lastCheckpointSize)` and `RATE(lastCheckpointDuration)`.
- **numberOfFailedCheckpoints** – This metric counts the number of failed checkpoints since the application started. Depending on the application, it can be tolerable if checkpoints fail occasionally. But if checkpoints are regularly failing, the application is likely unhealthy and needs further attention. We recommend monitoring `RATE(numberOfFailedCheckpoints)` to alarm on the gradient and not on absolute values.

Setting up application logging

By adding an Amazon CloudWatch logging option to your Managed Service for Apache Flink 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, Managed Service for Apache Flink 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](#)
- [Setting up CloudWatch logging using the CLI](#)
- [Application monitoring levels](#)
- [Logging best practices](#)
- [Logging troubleshooting](#)
- [Next step](#)

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.

Managed Service for Apache Flink creates a log group named using the following convention, where *ApplicationName* is your application's name.

```
/AWS/KinesisAnalytics/ApplicationName
```

Managed Service for Apache Flink 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”](#).

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](#)
- [Working with application CloudWatch logging options](#)
- [Adding permissions to write to the CloudWatch log stream](#)

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 [Managed Service for Apache Flink API example code](#).

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_15",
  "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"](#).

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_15",
    "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

Managed Service for Apache Flink needs permissions to write misconfiguration errors to CloudWatch. You can add these permissions to the AWS Identity and Access Management (IAM) role that Managed Service for Apache Flink assumes.

For more information about using an IAM role for Managed Service for Apache Flink, see [Identity and Access Management for Amazon Managed Service for Apache Flink](#).

Trust policy

To grant Managed Service for Apache Flink 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 Managed Service for Apache Flink 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”](#).
- **Operator:** Metrics are scoped to each operator. For information about operators, see [the section called “DataStream API operators”](#).
- **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”](#).

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”](#).

- 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 in Managed Service for Apache Flink](#).

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”](#).

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 Managed Service for Apache Flink 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](#)
- [Analyze operations: Change in parallelism](#)
- [Analyze errors: Access denied](#)
- [Analyze errors: Source or sink not found](#)
- [Analyze errors: Application task-related failures](#)

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](#).

```
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"](#).

```
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:kinesisanalyticsus-
west-2:012345678901:application\YourApplication/
| filter @message like /AccessDenied/
| sort @timestamp desc
```

Analyze errors: Source or sink not found

The following CloudWatch Logs Insights query returns ResourceNotFound logs. ResourceNotFound logs result if a Kinesis source or sink is not found.

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalyticsus-
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:kinesisanalyticsus-
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:kinesisanalyticsus-
west-2:012345678901:application\YourApplication/
| filter @message like /switched from RUNNING to FAILED/
| sort @timestamp desc
```

Viewing metrics and dimensions in Managed Service for Apache Flink

This topic contains the following sections:

- [Application metrics](#)
- [Kinesis Data Streams connector metrics](#)
- [Amazon MSK connector metrics](#)
- [Apache Zeppelin metrics](#)
- [Viewing CloudWatch metrics](#)
- [Setting CloudWatch metrics reporting levels](#)
- [Using custom metrics with Amazon Managed Service for Apache Flink](#)
- [Using CloudWatch Alarms with Amazon Managed Service for Apache Flink](#)

When your Managed Service for Apache Flink processes a data source, Managed Service for Apache Flink reports the following metrics and dimensions to Amazon CloudWatch.

Application metrics

Metric	Unit	Description	Level	Usage Notes
backPressuredTimeMsPerSecond*	Milliseconds	The time (in milliseconds) this task or operator is back pressured per second.	Task, Operator, Parallelism	<p>*Available for Managed Service for Apache Flink applications running Flink version 1.13 only.</p> <p>These metrics can be useful in identifying bottlenecks in an application.</p>
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	<p>*Available for Managed Service for Apache Flink applications running Flink version 1.13 only.</p> <p>These metrics can be useful in identifying bottlenecks in an application.</p>
cpuUtilization	Percentage	Overall percentage of CPU utilization across task	Application	You can use this metric to monitor minimum,

Metric	Unit	Description	Level	Usage Notes	
		managers. For example, if there are five task managers, Managed Service for Apache Flink publishes five samples of this metric per reporting interval.		average, and maximum CPU utilization in your application. The CPUUtilization metric only accounts for CPU usage of the TaskManager JVM process running inside the container.	

Metric	Unit	Description	Level	Usage Notes
container CPUUtilization	Percentage	Overall percentage of CPU utilization across task manager containers in Flink application cluster. For example, if there are five task managers, correspondingly there are five TaskManager containers and Managed Service for Apache Flink publishes 2 * five samples of this metric per 1 minute reporting interval.	Application	<p>It is calculated per container as:</p> $\frac{\text{Total CPU time (in seconds) consumed by container} * 100}{\text{Container CPU limit (in CPUs/seconds)}}$ <p>The CPUUtilization metric only accounts for CPU usage of the TaskManager JVM process running inside the container. There are other components running outside the JVM within the same container. The container CPUUtilization metric gives you a</p>

Metric	Unit	Description	Level	Usage Notes	
				more complete picture, including all processes in terms of CPU exhaustion at the container and failures resulting from that.	

Metric	Unit	Description	Level	Usage Notes
containerMemoryUtilization	Percentage	Overall percentage of memory utilization across task manager containers in Flink application cluster. For example, if there are five task managers, correspondingly there are five TaskManager containers and Managed Service for Apache Flink publishes 2 * five samples of this metric per 1 minute reporting interval.	Application	<p>It is calculated per container as:</p> $\frac{\text{Container memory usage (bytes)} * 100}{\text{Container memory limit as per pod deployment spec (in bytes)}}$ <p>The HeapMemoryUtilization and ManagedMemoryUtilizations metrics only account for specific memory metrics like Heap Memory Usage of TaskManager JVM or Managed Memory (memory usage outside JVM for native processes like</p>

Metric	Unit	Description	Level	Usage Notes
				RocksDB State Backend). The container MemoryUtilization metric gives you a more complete picture by including the working set memory, which is a better tracker of total memory exhaustion. Upon its exhaustion, it will result in Out of Memory Error for the TaskManager pod.

Metric	Unit	Description	Level	Usage Notes
container DiskUtili zation	Percentage	Overall percentage of disk utilization across task manager containers in Flink application cluster. For example, if there are five task managers, correspondingly there are five TaskManager containers and Managed Service for Apache Flink publishes 2 * five samples of this metric per 1 minute reporting interval.	Application	<p>It is calculated per container as:</p> <p><i>Disk usage in bytes * 100 / Disk Limit for container in bytes</i></p> <p>For containers, it represents utilization of the filesystem on which root volume of the container is set up.</p>

Metric	Unit	Description	Level	Usage Notes
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
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.

Metric	Unit	Description	Level	Usage Notes
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 Managed Service for Apache Flink. Restarts higher than normal can indicate a problem with the application.

Metric	Unit	Description	Level	Usage Notes
heapMemoryUtilization	Percentage	Overall heap memory utilization across task managers. For example, if there are five task managers, Managed Service for Apache Flink publishes five samples of this metric per reporting interval.	Application	You can use this metric to monitor minimum, average, and maximum heap memory utilization in your application. The HeapMemoryUtilization only accounts for specific memory metrics like Heap Memory Usage of TaskManager JVM.

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	<p>*Available for Managed Service for Apache Flink applications running Flink version 1.13 only.</p> <p>These metrics can be useful in identifying bottlenecks in an application.</p>
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>

Metric	Unit	Description	Level	Usage Notes
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
managedMemoryUsed*	Bytes	The amount of managed memory currently used.	Application, Operator, Task, Parallelism	<p>*Available for Managed Service for Apache Flink applications running Flink version 1.13 only.</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
managedMemoryTotal*	Bytes	The total amount of managed memory.	Application, Operator, Task, Parallelism	<p>*Available for Managed Service for Apache Flink applications running Flink version 1.13 only.</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. The ManagedMemoryUtilizations metric only accounts for specific memory metrics like Managed Memory (memory usage outside JVM for native processes like</p>

Metric	Unit	Description	Level	Usage Notes
				RocksDB State Backend)
managedMemoryUtilization*	Percentage	Derived by $\text{managedMemoryUsed} / \text{managedMemoryTotal}$	Application, Operator, Task, Parallelism	<p>*Available for Managed Service for Apache Flink applications running Flink version 1.13 only.</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
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.

Metric	Unit	Description	Level	Usage Notes
numRecordsIn*	Count	The total number of records this application, operator, or task has received.	Application, Operator, Task, Parallelism	<p>*To apply the SUM statistic over a period of time (second/minute):</p> <ul style="list-style-type: none">• Select the metric at the correct Level. If you're tracking the metric for an Operator, you need to select the corresponding operator metrics.• As Managed Service for Apache Flink takes 4 metric snapshots per minute, the following metric math should be used: $m1/4$ where $m1$ is the SUM

Metric	Unit	Description	Level	Usage Notes
				<p>statistic over a period (second/minute)</p> <p>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.</p>

Metric	Unit	Description	Level	Usage Notes
numRecordsInPerSecond*	Count/Second	The total number of records this application, operator or task has received per second.	Application, Operator, Task, Parallelism	<p>*To apply the SUM statistic over a period of time (second/minute):</p> <ul style="list-style-type: none"> • Select the metric at the correct Level. If you're tracking the metric for an Operator, you need to select the corresponding operator metrics. • As Managed Service for Apache Flink takes 4 metric snapshots per minute, the following metric math should be used: $m1/4$ where $m1$ is the SUM

Metric	Unit	Description	Level	Usage Notes
				<p>statistic over a period (second/minute)</p> <p>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.</p>

Metric	Unit	Description	Level	Usage Notes
numRecordsOut*	Count	The total number of records this application, operator or task has emitted.	Application, Operator, Task, Parallelism	<p>*To apply the SUM statistic over a period of time (second/minute):</p> <ul style="list-style-type: none">• Select the metric at the correct Level. If you're tracking the metric for an Operator, you need to select the corresponding operator metrics.• As Managed Service for Apache Flink takes 4 metric snapshots per minute, the following metric math should be used: $m1/4$ where $m1$ is the SUM

Metric	Unit	Description	Level	Usage Notes
				<p>statistic over a period (second/minute)</p> <p>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.</p>

Metric	Unit	Description	Level	Usage Notes
numLateRecordsDropped*	Count	Application, Operator, Task, Parallelism		<p>*To apply the SUM statistic over a period of time (second/minute):</p> <ul style="list-style-type: none">• Select the metric at the correct Level. If you're tracking the metric for an Operator, you need to select the corresponding operator metrics.• As Managed Service for Apache Flink takes 4 metric snapshots per minute, the following metric math should be used: $m1/4$ where $m1$ is the SUM

Metric	Unit	Description	Level	Usage Notes
				<p>statistic over a period (second/minute)</p> <p>The number of records this operator or task has dropped due to arriving late.</p>

Metric	Unit	Description	Level	Usage Notes
numRecordsOutPerSecond*	Count/Second	The total number of records this application, operator or task has emitted per second.	Application, Operator, Task, Parallelism	<p>*To apply the SUM statistic over a period of time (second/minute):</p> <ul style="list-style-type: none"> • Select the metric at the correct Level. If you're tracking the metric for an Operator, you need to select the corresponding operator metrics. • As Managed Service for Apache Flink takes 4 metric snapshots per minute, the following metric math should be used: $m1/4$ where $m1$ is the SUM

Metric	Unit	Description	Level	Usage Notes
				<p>statistic over a period (second/minute)</p> <p>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.</p>
oldGenerationGCCount	Count	The total number of old garbage collection operations that have occurred across all task managers.	Application	

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.
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.
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.

Metric	Unit	Description	Level	Usage Notes
KPUs*	Count	The total number of KPUs used by the application.	Application	<p>*This metric receives one sample per billing period (one hour). To visualize the number of KPUs over time, use MAX or AVG over a period of at least one (1) hour.</p> <p>The KPU count includes the orchestration KPU. For more information, see Managed Service for Apache Flink Pricing.</p>

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	Application (for Stream),	<ul style="list-style-type: none"> A value of 0 indicates that record

Metric	Unit	Description	Level	Usage Notes
		is behind the head of the stream, indicating how far behind current time the consumer is.	Parallelism (for ShardId)	<p>processing is caught up, and there are no new records to process at this moment. A particular shard's metric can be specified by stream name and shard id.</p> <ul style="list-style-type: none"> A value of -1 indicates that the service has not yet reported a value for the metric.
bytesRequestedPerFetch	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	Application (for Topic), Parallelism	

Metric	Unit	Description	Level	Usage Notes
		offset, for each partition . A particular partition's metric can be specified by topic name and partition id.	sm (for Partition Id)	
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	

Metric	Unit	Description	Level	Usage Notes
committed offsets	N/A	The last successfully committed offsets to Kafka, for each partition . A particular partition's metric can be specified by topic name and partition id.	Application (for Topic), Parallelism (for 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_rate	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 Managed Service for Apache Flink, 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. Managed Service for Apache Flink supports the following metrics levels:

- **Application:** The application only reports the highest level of metrics for each application. Managed Service for Apache Flink 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 Managed Service for Apache Flink

Managed Service 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](#)

- [Examples](#)
- [Viewing custom metrics](#)

How it works

Custom metrics in Managed Service for Apache Flink 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 Managed Service for Apache Flink, 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. Managed Service for Apache Flink 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 or [MonitoringConfigurationUpdate](#) parameter of the [UpdateApplication](#) action.

Examples

The following code examples demonstrate how to create a mapping class that 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.

Using CloudWatch Alarms with Amazon Managed Service 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 Managed Service for Apache Flink 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
<code>downtime > 0</code>	Average	0	A downtime greater than zero indicates that the application has failed. If the value is larger than 0, the application is not processing any data. Recommended for all applications. The Downtime metric measures the duration of an outage. A downtime greater than zero indicates that the application has failed. For troubleshooting, see Application is restarting .
<code>RATE (numberOfFailedCheckpoints) > 0</code>	Average	0	This metric counts the number of failed checkpoints since the application started. Depending on the application, it can be tolerable if checkpoin

Metric Expression	Statistic	Threshold	Description
			<p>ts fail occasionally. But if checkpoints are regularly failing, the application is likely unhealthy and needs further attention . We recommend monitoring <code>RATE(numberOfFailedCheckpoints)</code> to alarm on the gradient and not on absolute values. 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 Checkpointing is timing out.</p>

Metric Expression	Statistic	Threshold	Description
Operator. numRecord sOutPerSecond < threshold	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 Throughput is Too Slow.

Metric Expression	Statistic	Threshold	Description
<code>records_lag_max millisbehindLatest > threshold</code>	Maximum	The maximum expected latency during normal conditions.	If the application is consuming from Kinesis or Kafka, these metrics indicate if the application is falling behind and needs to be scaled in order to keep up with the current load. This is a good generic metric that is easy to track for all kinds of applications. But it can only be used for reactive scaling, i.e., when the application has already fallen behind. Recommended for all applications. Use the <code>records_lag_max</code> metric for a Kafka source, or the <code>millisbehindLatest</code> for a Kinesis stream source. Rising above this threshold can indicate that the application isn't making expected progress on the input data. For troubleshooting, see

Metric Expression	Statistic	Threshold	Description
			Throughput is Too Slow.

Metric Expression	Statistic	Threshold	Description
<code>lastCheckpointDuration > threshold</code>	Maximum	The maximum expected checkpoint duration during normal conditions.	Monitors how much data is stored in state and how long it takes to take a checkpoint. If checkpoints grow or take long, the application is continuously spending time on checkpointing and has less cycles for actual processing. At some points, checkpoints may grow too large or take so long that they fail. In addition to monitoring absolute values, customers should also consider monitoring the change rate with <code>RATE(lastCheckpointSize)</code> and <code>RATE(lastCheckpointDuration)</code> . If the <code>lastCheckpointDuration</code> continuously increases, rising above this threshold can indicate that

Metric Expression	Statistic	Threshold	Description
			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 Unbounded state growth .

Metric Expression	Statistic	Threshold	Description
<code>lastCheckpointSize > threshold</code>	Maximum	The maximum expected checkpoint size during normal conditions.	Monitors how much data is stored in state and how long it takes to take a checkpoint. If checkpoints grow or take long, the application is continuously spending time on checkpointing and has less cycles for actual processing. At some points, checkpoints may grow too large or take so long that they fail. In addition to monitoring absolute values, customers should also consider monitoring the change rate with <code>RATE(lastCheckpointSize)</code> and <code>RATE(lastCheckpointDuration)</code> . If the <code>lastCheckpointSize</code> continuously increases, rising above this threshold can indicate that

Metric Expression	Statistic	Threshold	Description
<code>heapMemoryUtilization</code> > threshold	Maximum	This gives a good indication of the overall resource utilization of the application and can be used for proactive scaling unless the application is I/O bound. The maximum expected <code>heapMemoryUtilization</code> size during normal conditions, with a recommended value of 90 percent.	<p>the application is accumulating state data. If the state data becomes too large, the application can run out of memory when recovering from a checkpoint, or recovering from a checkpoint might take too long. For troubleshooting, see Unbounded state growth.</p> <p>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 Scaling.</p>

Metric Expression	Statistic	Threshold	Description
<code>cpuUtilization > threshold</code>	Maximum	This gives a good indication of the overall resource utilization of the application and can be used for proactive scaling unless the application is I/O bound. 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 Scaling .
<code>threadsCount > 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') > 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(oldGarbageCollectionCount) > threshold</code>	Maximum	The maximum expected oldGarbageCollectionCount 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.

Metric Expression	Statistic	Threshold	Description
Operator. currentOutputWatermark - Operator. currentInputWatermark > threshold	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 Managed Service for Apache Flink 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](#)
- [Write to CloudWatch logs using SLF4J](#)

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 =
    LogManager.getLogger.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.managed-
flink.StreamingJob.main(StreamingJob.java:95)",
  "logger": "com.amazonaws.services.managed-flink.StreamingJob",
  "message": "This message will be written to the application's CloudWatch log",
  "threadName": "Flink-DispatcherRestEndpoint-thread-2",
  "applicationARN": "arn:aws:kinesisanalyticsus-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.managed-
flink.StreamingJob.main(StreamingJob.java:95)",
  "logger": "com.amazonaws.services.managed-flink.StreamingJob",
  "message": "This message will be written to the application's CloudWatch log",
  "threadName": "Flink-DispatcherRestEndpoint-thread-2",
  "applicationARN": "arn:aws:kinesisanalyticsus-east-1:123456789012:application/test",
  "applicationVersionId": "1", "messageSchemaVersion": "1",
  "messageType": "INFO"
}
```

Logging Managed Service for Apache Flink API calls with AWS CloudTrail

Managed Service for Apache Flink is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Managed Service for Apache Flink. CloudTrail captures all API calls for Managed Service for Apache Flink as events. The calls captured include calls from the Managed Service for Apache Flink console and code calls to the Managed Service for Apache Flink API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Managed Service for Apache Flink. 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 Managed Service for Apache Flink, 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](#).

Managed Service for Apache Flink information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Managed Service for Apache Flink, 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 Managed Service for Apache Flink, 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 Managed Service for Apache Flink actions are logged by CloudTrail and are documented in the [Managed Service for Apache Flink 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 Managed Service for Apache Flink 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:kinesisanalyticsus-
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 Managed Service for Apache Flink

This topic describes techniques to monitor and improve the performance of your Managed Service for Apache Flink application.

Topics

- [Troubleshooting performance](#)
- [Performance best practices](#)
- [Monitoring performance](#)

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](#)
- [Application CPU metric](#)
- [Application parallelism](#)
- [Application logging](#)
- [Operator parallelism](#)
- [Application logic](#)
- [Application memory](#)

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](#).

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](#) section following, and the [Scaling](#).

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 KPIs for an application is 64 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](#) 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"](#).

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](#).

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](#)
- [Manage operator scaling properly](#)

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](#).

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](#), 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](#).

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, 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 Managed Service for Apache Flink.

For information about creating a local Flink installation, see [Standalone](#) 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 Managed Service for Apache Flink application, see [Metrics and dimensions in Managed Service for Apache Flink](#).

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 Managed Service for Apache Flink application, see [Alarms](#).

Managed Service for Apache Flink and Studio notebook quota

Note

Apache Flink versions **1.6, 1.8, and 1.11** have not been supported by the Apache Flink community for over three years. We plan to deprecate these versions in Amazon Managed Service for Apache Flink on **November 5, 2024**. Starting from this date, you will not be able to create new applications for these Flink versions. You can continue running existing applications at this time. You can upgrade your applications statefully using the in-place version upgrades feature in Amazon Managed Service for Apache Flink. For more information, see [In-place version upgrades for Apache Flink](#).

When working with Amazon Managed Service for Apache Flink, note the following quota:

- You can create up to 50 Managed Service for Apache Flink 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 Managed Service for Apache Flink, see [Managed Service for Apache Flink Regions and Endpoints](#).

- The number of Kinesis processing units (KPU) is limited to 64 by default. For instructions on how to request an increase to this quota, see **To request a quota increase** in [Service Quotas](#). Make sure you specify the application prefix to which the new KPU limit needs to be applied.

With Managed Service for Apache Flink, 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 Managed Service for Apache Flink [Snapshots](#) 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
- Max cumulative UDF size = 100 MiB
- Max cumulative dependency jar size = 300 MiB

Managed Service for Apache Flink Maintenance

Managed Service for Apache Flink 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 Managed Service for Apache Flink 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”](#).

To change the time window during which Managed Service for Apache Flink performs maintenance on your application, use the [UpdateApplicationMaintenanceConfiguration](#) API.

Region	Maintenance time window
AWS GovCloud (US-West)	06:00–14:00 UTC
AWS GovCloud (US-East)	03:00–11:00 UTC
US East (N. Virginia)	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
Asia Pacific (Hong Kong)	13:00–21:00 UTC
Asia Pacific (Mumbai)	16:30–00:30 UTC
Asia Pacific (Hyderabad)	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

Region	Maintenance time window
Asia Pacific (Jakarta)	15:00–23:00 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 (Zurich)	20:00–04:00 UTC
Europe (Ireland)	22:00–06:00 UTC
Europe (London)	22:00–06:00 UTC
Europe (Stockholm)	23:00–07:00 UTC
Europe (Milan)	21:00–05:00 UTC
Europe (Spain)	21:00–05:00 UTC
Africa (Cape Town)	20:00–04:00 UTC
Europe (Ireland)	22:00–06:00 UTC
Europe (London)	23:00–07:00 UTC
Europe (Paris)	23:00–07:00 UTC
Europe (Stockholm)	23:00–07:00 UTC
Middle East (Bahrain)	13:00–21:00 UTC
Middle East (UAE)	18:00–02:00 UTC
South America (São Paulo)	19:00–03:00 UTC

Region	Maintenance time window
Israel (Tel Aviv)	20:00–04:00 UTC

Set a UUID for all operators

When Managed Service for Apache Flink starts a Flink job for an application with a snapshot, the Flink job can fail to start due to certain issues. One of them is *operator ID mismatch*. Flink expects explicit, consistent operator IDs for Flink job graph operators. If not set explicitly, Flink auto-generates an ID for the operators. This is because Flink uses these operator IDs to uniquely identify the operators in a job graph and uses them to store the state of each operator in a savepoint.

The *operator ID mismatch* issue happens when Flink does not find a 1:1 mapping between the operator IDs of a job graph and the operator IDs defined in a savepoint. This happens when explicit consistent operator IDs are not set and Flink auto-generates operator IDs that may not be consistent with every job graph creation. The likelihood of applications running into this issue is high during maintenance runs. To avoid this, we recommend customers set UUID for all operators in Flink code. For more information, see the topic *Set a UUID for all operators* under [Production readiness](#).

Identify when maintenance has occurred on your application

You can find if Managed Service for Apache Flink has performed a maintenance action on your application by using the `ListApplicationOperations` API.

The following is an example request for `ListApplicationOperations` that can help you filter the list for maintenance on the application:

```
{
  "ApplicationName": "MyApplication",
  "operation": "ApplicationMaintenance"
}
```

Production readiness

This is a collection of important aspects of running production applications on Managed Service for Apache Flink. It's not an exhaustive list, but rather the bare minimum of what you should pay attention to before putting an application into production.

Load testing applications

Some problems with applications only manifest under heavy load. We have seen cases where applications seemed healthy, yet an operational event substantially amplified the load on the application. This can happen completely independent of the application itself. If the data source or the data sink is unavailable for a couple of hours, the Flink application cannot make progress. When that issue is fixed, there is a backlog of unprocessed data that has accumulated, which can completely exhaust the available resources. The load can then amplify bugs or performance issues that had not emerged before.

It is therefore essential that you run proper load tests for production applications. Questions that should be answered during those load tests include:

- Is the application stable under sustained high load?
- Can the application still take a savepoint under peak load?
- How long does it take to process a backlog of 1 hour? And how long for 24 hours (depending on the max retention of the data in the stream)?
- Does the throughput of the application increase when the application is scaled?

When consuming from a data stream, these scenarios can be simulated by producing into the stream for some time. Then start the application and have it consume data from the beginning of time. For example, use a start position of `TRIM_HORIZON` in the case of a Kinesis data stream.

Max parallelism

The max parallelism defines the maximum parallelism a stateful application can scale to. This is defined when the state is first created and there is no way of scaling the operator beyond this maximum without discarding the state.

Max parallelism is set when the state is first created.

By default, Max parallelism is set to:

- 128, if parallelism \leq 128
- $\text{MIN}(\text{nextPowerOfTwo}(\text{parallelism} + (\text{parallelism} / 2)), 2^{15})$: if parallelism $>$ 128

If you are planning to scale your application $>$ 128 parallelism, you should explicitly define the Max parallelism.

You can define Max parallelism at level of application, with `env.setMaxParallelism(x)` or single operator. Unless differently specified, all operators inherit the Max parallelism of the application.

For more information, see [Setting the Maximum Parallelism](#) in the Apache Flink Documentation.

Set a UUID for all operators

A UUID is used in the operation in which Flink maps a savepoint back to an individual operator. Setting a specific UUID for each operator gives a stable mapping for the savepoint process to restore.

```
.map(...).uid("my-map-function")
```

For more information, see [Production Readiness Checklist](#).

Best Practices for Managed Service for Apache Flink

This section contains information and recommendations for developing stable, performant Managed Service for Apache Flink applications.

Topics

- [Fault tolerance: checkpoints and savepoints](#)
- [Unsupported connector versions](#)
- [Performance and parallelism](#)
- [Setting per-operator parallelism](#)
- [Logging](#)
- [Coding](#)
- [Managing credentials](#)
- [Reading from sources with few shards/partitions](#)
- [Studio notebook refresh interval](#)
- [Studio notebook optimum performance](#)
- [How watermark strategies and idle shards affect time windows](#)
- [Set a UUID for all operators](#)
- [Add ServiceResourceTransformer to the Maven shade plugin](#)

Fault tolerance: checkpoints and savepoints

Use checkpoints and savepoints to implement fault tolerance in your a Managed Service 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](#).
- 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 in Managed Service for Apache Flink](#).

For more information about implementing fault tolerance, see [Fault tolerance](#).

Unsupported connector versions

From Apache Flink version 1.15 or later, Managed Service for Apache Flink automatically prevents applications from starting or updating if they are using unsupported Kinesis connector versions bundled into application JARs. When upgrading to Managed Service for Apache Flink version 1.15 or later, make sure that you are using the most recent Kinesis connector. This is any version equal to or newer than version 1.15.2. All other versions are not supported by Managed Service for Apache Flink because they might cause consistency issues or failures with the **Stop with Savepoint** feature, preventing clean stop/update operations. To learn more about connector compatibility in Amazon Managed Service for Apache Flink versions, see [Apache Flink connectors](#).

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](#).

Setting per-operator parallelism

By default, all operators have the parallelism set at application level. You can override the parallelism of a single operator using the DataStream API using `.setParallelism(x)`. You can set an operator parallelism to any parallelism equal or lower than the application parallelism.

If possible, define the operator parallelism as a function of the application parallelism. This way, the operator parallelism will vary with the application parallelism. If you are using autoscaling, for example, all operators will vary their parallelism in the same proportion:

```
int appParallelism = env.getParallelism();
...
...ops.setParallelism(appParallelism/2);
```

In some cases, you may want to set the operator parallelism to a constant. For example, setting the parallelism of a Kinesis Stream source to the number of shards. In these cases, you should consider passing the operator parallelism as application configuration parameter, in order to change it without changing the code, if you need, for example, to reshard the source stream.

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](#)

For more information about implementing logging, see [Logging and monitoring](#).

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](#).

Managing credentials

You should not bake any long-term credentials into production (or any other) applications. Long-term credentials are likely checked into a version control system and can easily get lost. Instead, you can associate a role to the Managed Service for Apache Flink application and grant privileges

to that role. The running Flink application can then pick up temporary credentials with the respective privileges from the environment. In case authentication is needed for a service that is not natively integrated with IAM, e.g., a database that requires a username and password for authentication, you should consider storing secrets in [AWS Secrets Manager](#).

Many AWS native services support authentication:

- Kinesis Data Streams – [ProcessTaxiStream.java](#)
- Amazon MSK – <https://github.com/aws/aws-msk-iam-auth/#using-the-amazon-msk-library-for-iam-authentication>
- Amazon Elasticsearch Service – [AmazonElasticsearchSink.java](#)
- Amazon S3 – works out of the box on Managed Service for Apache Flink

Reading from sources with few shards/partitions

When reading from Apache Kafka or a Kinesis Data Stream, there may be a mismatch between the parallelism of the stream (i.e., the number of partitions for Kafka and the number of shards for Kinesis) and the parallelism of the application. With a naive design, the parallelism of an application cannot scale beyond the parallelism of a stream: Each subtask of a source operator can only read from 1 or more shards/partitions. That means for a stream with only 2 shards and an application with a parallelism of 8, that only two subtasks are actually consuming from the stream and 6 subtasks remain idle. This can substantially limit the throughput of the application, in particular if the deserialization is expensive and carried out by the source (which is the default).

To mitigate this effect, you can either scale the stream. But that may not always be desirable or possible. Alternatively, you can restructure the source so that it does not do any serialization and just passes on the `byte[]`. You can then [rebalance](#) the data to distribute it evenly across all tasks and then deserialize the data there. In this way, you can leverage all subtasks for the deserialization and this potentially expensive operation is no longer bound by the number of shards/partitions of the stream.

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
```

How watermark strategies and idle shards affect time windows

When reading events from Apache Kafka and Kinesis Data Streams, the source can set the event time based on attributes of the stream. In case of Kinesis, the event time equals the approximate arrival time of events. But setting event time at the source for events is not sufficient for a Flink application to use event time. The source must also generate watermarks that propagate information about event time from the source to all other operators. The [Flink documentation](#) has a good overview of how that process works.

By default, the timestamp of an event read from Kinesis is set to the approximate arrival time determined by Kinesis. An additional prerequisite for event time to work in the application is a watermark strategy.

```
WatermarkStrategy<String> s = WatermarkStrategy  
    .<String>forMonotonousTimestamps()  
    .withIdleness(Duration.ofSeconds(...));
```

The watermark strategy is then applied to a `DataStream` with the `assignTimestampsAndWatermarks` method. There are some useful build-in strategies:

- `forMonotonousTimestamps()` will just use the event time (approximate arrival time) and periodically emit the maximum value as a watermark (for each specific subtask)
- `forBoundedOutOfOrderness(Duration.ofSeconds(...))` similar to the previous strategy, but will use the event time – duration for watermark generation.

This works, but there are a couple of caveats to be aware of. Watermarks are generated at a subtask level and flow through the operator graph.

From the [Flink documentation](#):

Each parallel subtask of a source function usually generates its watermarks independently. These watermarks define the event time at that particular parallel source.

As the watermarks flow through the streaming program, they advance the event time at the operators where they arrive. Whenever an operator advances its event time, it generates a new watermark downstream for its successor operators.

Some operators consume multiple input streams; a union, for example, or operators following a `keyBy(...)` or `partition(...)` function. Such an operator's current event time is the minimum of its input streams' event times. As its input streams update their event times, so does the operator.

That means, if a source subtask is consuming from an idle shard, downstream operators do not receive new watermarks from that subtask and hence processing stalls for all downstream operators that use time windows. To avoid this, customers can add the `withIdleness` option to the watermark strategy. With that option, an operator excludes the watermarks from idle upstream subtasks when computing the event time of the operator. Idle subtask therefore no longer block the advancement of event time in downstream operators.

However, the idleness option with the build-in watermark strategies will not advance the event time if no subtask is reading any event, i.e., there are no events in the stream. This becomes particularly visible for test cases where a finite set of events is read from the stream. As event time does not advance after the last event has been read, the last window (containing the last event) will never close.

Summary

- the `withIdleness` setting will not generate new watermarks in case a shard is idle, it will just exclude the last watermark sent by idle subtasks from the min watermark calculation in downstream operators
- with the build-in watermark strategies the last open window will never close (unless new events that advance the watermark will be sent, but that creates a new window that then remains open)
- even when the time is set by the Kinesis stream, late arriving events can still happen if one shard is consumed faster than others (eg, during app initialization or when using `TRIM_HORIZON` where all existing shards are consumed in parallel ignoring their parent/child relationship)
- the `withIdleness` settings of the watermark strategy seem to deprecate the the Kinesis source specific settings for idle shards (`ConsumerConfigConstants.SHARD_IDLE_INTERVAL_MILLIS`)

Example

The following application is reading from a stream and creating session windows based on event time.

```
Properties consumerConfig = new Properties();
consumerConfig.put(AWSConfigConstants.AWS_REGION, "eu-west-1");
consumerConfig.put(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "TRIM_HORIZON");

FlinkKinesisConsumer<String> consumer = new FlinkKinesisConsumer<>("...", new
    SimpleStringSchema(), consumerConfig);

WatermarkStrategy<String> s = WatermarkStrategy
    .<String>forMonotonousTimestamps()
    .withIdleness(Duration.ofSeconds(15));

env.addSource(consumer)
    .assignTimestampsAndWatermarks(s)
    .map(new MapFunction<String, Long>() {
        @Override
        public Long map(String s) throws Exception {
            return Long.parseLong(s);
        }
    })
    .keyBy(1 -> 01)
    .window(EventTimeSessionWindows.withGap(Time.seconds(10)))
    .process(new ProcessWindowFunction<Long, Object, Long, TimeWindow>() {
        @Override
        public void process(Long aLong, ProcessWindowFunction<Long, Object, Long,
            TimeWindow>.Context context, Iterable<Long>iterable, Collector<Object> collector)
            throws Exception {
            long count = StreamSupport.stream(iterable.spliterator(), false).count();
            long timestamp = context.currentWatermark();

            System.out.print("XXXXXXXXXXXXXXXXX Window with " + count + " events");
            System.out.println("; Watermark: " + timestamp + ", " +
                Instant.ofEpochMilli(timestamp));

            for (Long l : iterable) {
                System.out.println(l);
            }
        }
    })
```

```
});
```

In the following example, 8 events are written to a 16 shard stream (the first 2 and the last event happen to land in the same shard).

```
$ aws kinesys put-record --stream-name hp-16 --partition-key 1 --data MQ==
$ aws kinesys put-record --stream-name hp-16 --partition-key 2 --data Mg==
$ aws kinesys put-record --stream-name hp-16 --partition-key 3 --data Mw==
$ date

{
  "ShardId": "shardId-000000000012",
  "SequenceNumber": "49627894338614655560500811028721934184977530127978070210"
}
{
  "ShardId": "shardId-000000000012",
  "SequenceNumber": "49627894338614655560500811028795678659974022576354623682"
}
{
  "ShardId": "shardId-000000000014",
  "SequenceNumber": "49627894338659257050897872275134360684221592378842022114"
}
Wed Mar 23 11:19:57 CET 2022

$ sleep 10
$ aws kinesys put-record --stream-name hp-16 --partition-key 4 --data NA==
$ aws kinesys put-record --stream-name hp-16 --partition-key 5 --data NQ==
$ date

{
  "ShardId": "shardId-000000000010",
  "SequenceNumber": "49627894338570054070103749783042116732419934393936642210"
}
{
  "ShardId": "shardId-000000000014",
  "SequenceNumber": "49627894338659257050897872275659034489934342334017700066"
}
Wed Mar 23 11:20:10 CET 2022

$ sleep 10
$ aws kinesys put-record --stream-name hp-16 --partition-key 6 --data Ng==
$ date
```

```

{
  "ShardId": "shardId-000000000001",
  "SequenceNumber": "49627894338369347363316974173886988345467035365375213586"
}
Wed Mar 23 11:20:22 CET 2022

$ sleep 10
$ aws kinesis put-record --stream-name hp-16 --partition-key 7 --data Nw==
$ date

{
  "ShardId": "shardId-000000000008",
  "SequenceNumber": "49627894338525452579706688535878947299195189349725503618"
}
Wed Mar 23 11:20:34 CET 2022

$ sleep 60
$ aws kinesis put-record --stream-name hp-16 --partition-key 8 --data OA==
$ date

{
  "ShardId": "shardId-000000000012",
  "SequenceNumber": "49627894338614655560500811029600823255837371928900796610"
}
Wed Mar 23 11:21:27 CET 2022

```

This input should result in 5 session windows: event 1,2,3; event 4,5; event 6; event 7; event 8. However, the program only yields the first 4 windows.

```

11:59:21,529 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 5 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000006,HashKeyRange: {StartingHashKey:
127605887595351923798765477786913079296,EndingHashKey:
148873535527910577765226390751398592511},SequenceNumberRange: {StartingSequenceNumber:
49627894338480851089309627289524549239292625588395704418,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,530 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 5 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000006,HashKeyRange: {StartingHashKey:
127605887595351923798765477786913079296,EndingHashKey:
148873535527910577765226390751398592511},SequenceNumberRange: {StartingSequenceNumber:

```

```
49627894338480851089309627289524549239292625588395704418,}}}' from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,530 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 6 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000007,HashKeyRange: {StartingHashKey:
148873535527910577765226390751398592512,EndingHashKey:
170141183460469231731687303715884105727},SequenceNumberRange: {StartingSequenceNumber:
49627894338503151834508157912666084957565273949901684850,}}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,530 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 6 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000010,HashKeyRange: {StartingHashKey:
212676479325586539664609129644855132160,EndingHashKey:
233944127258145193631070042609340645375},SequenceNumberRange: {StartingSequenceNumber:
49627894338570054070103749782090692112383219034419626146,}}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,530 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 6 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000007,HashKeyRange: {StartingHashKey:
148873535527910577765226390751398592512,EndingHashKey:
170141183460469231731687303715884105727},SequenceNumberRange: {StartingSequenceNumber:
49627894338503151834508157912666084957565273949901684850,}}}' from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,531 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 4 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000005,HashKeyRange: {StartingHashKey:
106338239662793269832304564822427566080,EndingHashKey:
127605887595351923798765477786913079295},SequenceNumberRange: {StartingSequenceNumber:
49627894338458550344111096666383013521019977226889723986,}}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 4 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000005,HashKeyRange: {StartingHashKey:
106338239662793269832304564822427566080,EndingHashKey:
127605887595351923798765477786913079295},SequenceNumberRange: {StartingSequenceNumber:
49627894338458550344111096666383013521019977226889723986,}}}' from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,532 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 3 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000004,HashKeyRange: {StartingHashKey:
85070591730234615865843651857942052864,EndingHashKey:
106338239662793269832304564822427566079},SequenceNumberRange: {StartingSequenceNumber:
```

```
49627894338436249598912566043241477802747328865383743554,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 2 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000003,HashKeyRange: {StartingHashKey:
63802943797675961899382738893456539648,EndingHashKey:
85070591730234615865843651857942052863},SequenceNumberRange: {StartingSequenceNumber:
49627894338413948853714035420099942084474680503877763122,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 3 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000015,HashKeyRange: {StartingHashKey:
319014718988379809496913694467282698240,EndingHashKey:
340282366920938463463374607431768211455},SequenceNumberRange: {StartingSequenceNumber:
49627894338681557796096402897798370703746460841949528306,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 2 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000014,HashKeyRange: {StartingHashKey:
297747071055821155530452781502797185024,EndingHashKey:
319014718988379809496913694467282698239},SequenceNumberRange: {StartingSequenceNumber:
49627894338659257050897872274656834985473812480443547874,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 3 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000004,HashKeyRange: {StartingHashKey:
85070591730234615865843651857942052864,EndingHashKey:
106338239662793269832304564822427566079},SequenceNumberRange: {StartingSequenceNumber:
49627894338436249598912566043241477802747328865383743554,}}'} from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,532 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 2 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000003,HashKeyRange: {StartingHashKey:
63802943797675961899382738893456539648,EndingHashKey:
85070591730234615865843651857942052863},SequenceNumberRange: {StartingSequenceNumber:
49627894338413948853714035420099942084474680503877763122,}}'} from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,532 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 0 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000001,HashKeyRange: {StartingHashKey:
21267647932558653966460912964485513216,EndingHashKey:
42535295865117307932921825928971026431},SequenceNumberRange: {StartingSequenceNumber:
```

```
49627894338369347363316974173816870647929383780865802258,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 0 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000009,HashKeyRange: {StartingHashKey:
191408831393027885698148216680369618944,EndingHashKey:
212676479325586539664609129644855132159},SequenceNumberRange: {StartingSequenceNumber:
49627894338547753324905219158949156394110570672913645714,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 7 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000000,HashKeyRange: {StartingHashKey: 0,EndingHashKey:
21267647932558653966460912964485513215},SequenceNumberRange: {StartingSequenceNumber:
49627894338347046618118443550675334929656735419359821826,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 0 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000012,HashKeyRange: {StartingHashKey:
255211775190703847597530955573826158592,EndingHashKey:
276479423123262501563991868538311671807},SequenceNumberRange: {StartingSequenceNumber:
49627894338614655560500811028373763548928515757431587010,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 7 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000008,HashKeyRange: {StartingHashKey:
170141183460469231731687303715884105728,EndingHashKey:
191408831393027885698148216680369618943},SequenceNumberRange: {StartingSequenceNumber:
49627894338525452579706688535807620675837922311407665282,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 0 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000001,HashKeyRange: {StartingHashKey:
21267647932558653966460912964485513216,EndingHashKey:
42535295865117307932921825928971026431},SequenceNumberRange: {StartingSequenceNumber:
49627894338369347363316974173816870647929383780865802258,}}'} from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,533 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 7 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000011,HashKeyRange: {StartingHashKey:
233944127258145193631070042609340645376,EndingHashKey:
255211775190703847597530955573826158591},SequenceNumberRange: {StartingSequenceNumber:
49627894338592354815302280405232227830655867395925606578,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
```

```
11:59:21,533 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 7 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000000,HashKeyRange: {StartingHashKey: 0,EndingHashKey:
21267647932558653966460912964485513215},SequenceNumberRange: {StartingSequenceNumber:
49627894338347046618118443550675334929656735419359821826,}}'} from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,568 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 1 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000002,HashKeyRange: {StartingHashKey:
42535295865117307932921825928971026432,EndingHashKey:
63802943797675961899382738893456539647},SequenceNumberRange: {StartingSequenceNumber:
49627894338391648108515504796958406366202032142371782690,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,568 INFO org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
[] - Subtask 1 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000013,HashKeyRange: {StartingHashKey:
276479423123262501563991868538311671808,EndingHashKey:
297747071055821155530452781502797185023},SequenceNumberRange: {StartingSequenceNumber:
49627894338636956305699341651515299267201164118937567442,}}'}, starting state set as
sequence number EARLIEST_SEQUENCE_NUM
11:59:21,568 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 1 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000002,HashKeyRange: {StartingHashKey:
42535295865117307932921825928971026432,EndingHashKey:
63802943797675961899382738893456539647},SequenceNumberRange: {StartingSequenceNumber:
49627894338391648108515504796958406366202032142371782690,}}'} from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:23,209 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 0 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000009,HashKeyRange: {StartingHashKey:
191408831393027885698148216680369618944,EndingHashKey:
212676479325586539664609129644855132159},SequenceNumberRange: {StartingSequenceNumber:
49627894338547753324905219158949156394110570672913645714,}}'} from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,244 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 6 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000010,HashKeyRange: {StartingHashKey:
212676479325586539664609129644855132160,EndingHashKey:
233944127258145193631070042609340645375},SequenceNumberRange: {StartingSequenceNumber:
```

```
49627894338570054070103749782090692112383219034419626146,}}' } from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 1
event: 6; timestamp: 1648030822428, 2022-03-23T10:20:22.428Z
11:59:23,377 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 3 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000015,HashKeyRange: {StartingHashKey:
319014718988379809496913694467282698240,EndingHashKey:
340282366920938463463374607431768211455},SequenceNumberRange: {StartingSequenceNumber:
49627894338681557796096402897798370703746460841949528306,}}' } from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,405 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 2 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000014,HashKeyRange: {StartingHashKey:
297747071055821155530452781502797185024,EndingHashKey:
319014718988379809496913694467282698239},SequenceNumberRange: {StartingSequenceNumber:
49627894338659257050897872274656834985473812480443547874,}}' } from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,581 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 7 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000008,HashKeyRange: {StartingHashKey:
170141183460469231731687303715884105728,EndingHashKey:
191408831393027885698148216680369618943},SequenceNumberRange: {StartingSequenceNumber:
49627894338525452579706688535807620675837922311407665282,}}' } from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,586 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 1 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000013,HashKeyRange: {StartingHashKey:
276479423123262501563991868538311671808,EndingHashKey:
297747071055821155530452781502797185023},SequenceNumberRange: {StartingSequenceNumber:
49627894338636956305699341651515299267201164118937567442,}}' } from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:24,790 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 0 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000012,HashKeyRange: {StartingHashKey:
255211775190703847597530955573826158592,EndingHashKey:
276479423123262501563991868538311671807},SequenceNumberRange: {StartingSequenceNumber:
49627894338614655560500811028373763548928515757431587010,}}' } from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 2
event: 4; timestamp: 1648030809282, 2022-03-23T10:20:09.282Z
```



```

event: 3; timestamp: 1648030797697, 2022-03-23T10:19:57.697Z
event: 5; timestamp: 1648030810871, 2022-03-23T10:20:10.871Z
11:59:24,907 INFO
org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
Subtask 7 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
shard='{ShardId: shardId-000000000011,HashKeyRange: {StartingHashKey:
233944127258145193631070042609340645376,EndingHashKey:
255211775190703847597530955573826158591},SequenceNumberRange: {StartingSequenceNumber:
49627894338592354815302280405232227830655867395925606578,}}'} from sequence number
EARLIEST_SEQUENCE_NUM with ShardConsumer 2
event: 7; timestamp: 1648030834105, 2022-03-23T10:20:34.105Z
event: 1; timestamp: 1648030794441, 2022-03-23T10:19:54.441Z
event: 2; timestamp: 1648030796122, 2022-03-23T10:19:56.122Z
event: 8; timestamp: 1648030887171, 2022-03-23T10:21:27.171Z
XXXXXXXXXXXXXXXXX Window with 3 events; Watermark: 1648030809281, 2022-03-23T10:20:09.281Z
3
1
2
XXXXXXXXXXXXXXXXX Window with 2 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
4
5
XXXXXXXXXXXXXXXXX Window with 1 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
6
XXXXXXXXXXXXXXXXX Window with 1 events; Watermark: 1648030887170, 2022-03-23T10:21:27.170Z
7

```

The output is only showing 4 windows (missing the last window containing event 8). This is due to event time and the watermark strategy. The last window cannot close because with the per-built watermark strategies the time never advances beyond the time of the last event that has been read from the stream. But for the window to close, time needs to advance more than 10 seconds after the last event. In this case the last watermark is 2022-03-23T10:21:27.170Z but in order for the session window to close, a watermark 10s and 1ms later is required.

If the `withIdleness` option is removed from the watermark strategy, no session window will ever close, because the the “global watermark” of the window operator cannot advance.

Note that when the Flink application starts (or if there is data skew), some shards may be consumed faster than others. This can cause some watermarks to be emitted too early from a subtask (the subtask may emit the watermark based on the content of one shard without having consumed from the other shards it’s subscribed to). Ways to mitigate are a different watermarking strategies that add a safety buffer

(`forBoundedOutOfOrderness(Duration.ofSeconds(30))`) or explicitly allowing late arriving events (`allowedLateness(Time.minutes(5))`).

Set a UUID for all operators

When Managed Service for Apache Flink starts a Flink job for an application with a snapshot, the Flink job can fail to start due to certain issues. One of them is *operator ID mismatch*. Flink expects explicit, consistent operator IDs for Flink job graph operators. If not set explicitly, Flink auto-generates an ID for the operators. This is because Flink uses these operator IDs to uniquely identify the operators in a job graph and uses them to store the state of each operator in a savepoint.

The *operator ID mismatch* issue happens when Flink does not find a 1:1 mapping between the operator IDs of a job graph and the operator IDs defined in a savepoint. This happens when explicit consistent operator IDs are not set and Flink auto-generates operator IDs that may not be consistent with every job graph creation. The likelihood of applications running into this issue is high during maintenance runs. To avoid this, we recommend customers set UUID for all operators in flink code. For more information, see the topic *Set a UUID for all operators* under [Production readiness](#).

Add ServiceResourceTransformer to the Maven shade plugin

Flink uses Java's [Service Provider Interfaces \(SPI\)](#) to load components such as connectors and formats. Multiple Flink dependencies using SPI [may cause clashes in the uber-jar](#) and unexpected application behaviours. It is recommended to add the [ServiceResourceTransformer](#) of the Maven shade plugin, defined in the pom.xml

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <executions>
        <execution>
          <id>shade</id>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
          <configuration>
```

```
        <transformers combine.children="append">
            <!-- The service transformer is needed to merge META-
INF/services files -->
                <transformer
implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"/>
                    <!-- ... -->
                </transformers>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Apache Flink stateful functions

[Stateful Functions](#) is an API that simplifies building distributed stateful applications. It's based on functions with persistent state that can interact dynamically with strong consistency guarantees.

A Stateful Functions application is basically just an Apache Flink Application and hence can be deployed to Managed Service for Apache Flink. However, there are a couple of differences between packaging Stateful Functions for a Kubernetes cluster and for Managed Service for Apache Flink. The most important aspect of a Stateful Functions application is the [module configuration](#) contains all necessary runtime information to configure the Stateful Functions runtime. This configuration is usually packaged into a Stateful Functions specific container and deployed on Kubernetes. But that is not possible with Managed Service for Apache Flink.

Following is an adaptation of the StateFun Python example for Managed Service for Apache Flink:

Apache Flink application template

Instead of using a customer container for the Stateful Functions runtime, customers can compile a Flink application jar that just invokes the Stateful Functions runtime and contains the required dependencies. For Flink 1.13, the required dependencies look similar to this:

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>statefun-flink-distribution</artifactId>
  <version>3.1.0</version>
  <exclusions>
    <exclusion>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
    </exclusion>
    <exclusion>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

And the main method of the Flink application to invoke the Stateful Function runtime looks like this:

```
public static void main(String[] args) throws Exception {
    final StreamExecutionEnvironment env =
        StreamExecutionEnvironment.getExecutionEnvironment();

    StatefulFunctionsConfig stateFunConfig = StatefulFunctionsConfig.fromEnvironment(env);

    stateFunConfig.setProvider((StatefulFunctionsUniverseProvider) (classLoader,
        statefulFunctionsConfig) -> {
        Modules modules = Modules.loadFromClassPath();
        return modules.createStatefulFunctionsUniverse(stateFunConfig);
    });

    StatefulFunctionsJob.main(env, stateFunConfig);
}
```

Note that these components are generic and independent of the logic that is implemented in the Stateful Function.

Location of the module configuration

The Stateful Functions module configuration needs to be included in the class path to be discoverable for the Stateful Functions runtime. It's best to include it in the resources folder of the Flink application and package it into the jar file.

Similar to a common Apache Flink application, you can then use maven to create an uber jar file and deploy that on Managed Service for Apache Flink.

Apache Flink settings

Managed Service for Apache Flink is an implementation of the Apache Flink framework. Managed Service for Apache Flink uses the default values described in this section. Some of these values can be set by the Managed Service for Apache Flink applications in code, and others cannot be changed.

This topic contains the following sections:

- [Apache Flink configuration](#)
- [State backend](#)
- [Checkpointing](#)
- [Savepointing](#)
- [Heap sizes](#)
- [Buffer debloating](#)
- [Modifiable Flink configuration properties](#)
- [Viewing configured Flink properties](#)

Apache Flink configuration

Managed Service for Apache Flink provides a default Flink configuration consisting of Apache Flink-recommended values for most properties and a few based on common application profiles. For more information about Flink configuration, see [Configuration](#). Service-provided default configuration works for most applications. However, to tweak Flink configuration properties to improve performance for certain applications with high parallelism, high memory and state usage, or enable new debugging features in Apache Flink, you can change certain properties by requesting a support case. For more information, see [AWS Support Center](#). You can check the current configuration for your application using the [Apache Flink Dashboard](#).

State backend

Managed Service for Apache Flink stores transient data in a state backend. Managed Service for Apache Flink 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

For more information about state backends, see [State Backends](#) in the Apache Flink Documentation.

Checkpointing

Managed Service for Apache Flink uses a default checkpoint configuration with the following values. Some of these values can be changed using [CheckpointConfiguration](#). You must set `CheckpointConfiguration.ConfigurationType` to `CUSTOM` for Managed Service for Apache Flink to use modified checkpointing values.

Setting	Can be modified?	How	Default Value
CheckpointingEnabled	Modifiable	Create Application Update Application AWS CloudFormation	True
CheckpointInterval	Modifiable	Create Application Update Application AWS CloudFormation	60000
MinPauseBetweenCheckpoints	Modifiable	Create Application Update Application AWS CloudFormation	5000
Unaligned checkpoints	Modifiable	Support case	False
Number of Concurrent Checkpoints	Not Modifiable	N/A	1

Setting	Can be modified?	How	Default Value
Checkpointing Mode	Not Modifiable	N/A	Exactly Once
Checkpoint Retention Policy	Not Modifiable	N/A	On Failure
Checkpoint Timeout	Not Modifiable	N/A	60 minutes
Max Checkpoints Retained	Not Modifiable	N/A	1
Checkpoint and Savepoint Location	Not Modifiable	N/A	We store durable checkpoint and savepoint data to a service-owned S3 bucket.

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

Managed Service for Apache Flink 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"](#).

For more information about JVM heap sizes, see [Configuration](#) in the [Apache Flink documentation](#).

Buffer debloating

Buffer debloating can help applications with high backpressure. If your application experiences failed checkpoints/savepoints, enabling this feature could be useful. To do this, request a [support case](#).

For more information, see [The Buffer Debloating Mechanism](#) in the [Apache Flink documentation](#).

Modifiable Flink configuration properties

Following are Flink configuration settings that you can modify using a [support case](#). You can modify more than one property at a time, and for multiple applications at the same time by specifying the application prefix. If there are other Flink configuration properties outside this list you want to modify, specify the exact property in your case.

Restart strategy

From Flink 1.19 and later, we use the `exponential-delay` restart strategy by default. All previous versions use the `fixed-delay` restart strategy by default.

```
restart-strategy:
```

```
restart-strategy.fixed-delay.delay:
```

```
restart-strategy.exponential-delay.backoff-multiplier:
```

```
restart-strategy.exponential-delay.initial-backoff:
```

```
restart-strategy.exponential-delay.jitter-factor:
```

```
restart-strategy.exponential-delay.reset-backoff-threshold:
```

Checkpoints and state backends

```
state.backend:
```

```
state.backend.fs.memory-threshold:
```

```
state.backend.incremental:
```

Checkpointing

`execution.checkpointing.unaligned:`

`execution.checkpointing.interval-during-backlog:`

RocksDB native metrics

RocksDB Native Metrics are not shipped to CloudWatch. Once enabled, these metrics can be accessed either from the Flink dashboard or the Flink REST API with custom tooling.

Managed Service for Apache Flink enables customers to access the latest Flink [REST API](#) (or the supported version you are using) in read-only mode using the [CreateApplicationPresignedUrl](#) API. This API is used by Flink's own dashboard, but it can also be used by custom monitoring tools.

`state.backend.rocksdb.compaction.style:`

`state.backend.rocksdb.memory.partitioned-index-filters:`

`state.backend.rocksdb.metrics.actual-delayed-write-rate:`

`state.backend.rocksdb.metrics.background-errors:`

`state.backend.rocksdb.metrics.block-cache-capacity:`

`state.backend.rocksdb.metrics.block-cache-pinned-usage:`

`state.backend.rocksdb.metrics.block-cache-usage:`

`state.backend.rocksdb.metrics.column-family-as-variable:`

`state.backend.rocksdb.metrics.compaction-pending:`

`state.backend.rocksdb.metrics.cur-size-active-mem-table:`

`state.backend.rocksdb.metrics.cur-size-all-mem-tables:`

`state.backend.rocksdb.metrics.estimate-live-data-size:`

`state.backend.rocksdb.metrics.estimate-num-keys:`

`state.backend.rocksdb.metrics.estimate-pending-compaction-bytes:`

state.backend.rocksdb.metrics.estimate-table-readers-mem:
state.backend.rocksdb.metrics.is-write-stopped:
state.backend.rocksdb.metrics.mem-table-flush-pending:
state.backend.rocksdb.metrics.num-deletes-active-mem-table:
state.backend.rocksdb.metrics.num-deletes-imm-mem-tables:
state.backend.rocksdb.metrics.num-entries-active-mem-table:
state.backend.rocksdb.metrics.num-entries-imm-mem-tables:
state.backend.rocksdb.metrics.num-immutable-mem-table:
state.backend.rocksdb.metrics.num-live-versions:
state.backend.rocksdb.metrics.num-running-compactions:
state.backend.rocksdb.metrics.num-running-flushes:
state.backend.rocksdb.metrics.num-snapshots:
state.backend.rocksdb.metrics.size-all-mem-tables:
state.backend.rocksdb.thread.num:

Advanced state backends options

state.storage.fs.memory-threshold:

Full TaskManager options

task.cancellation.timeout:
taskmanager.jvm-exit-on-oom:
taskmanager.numberOfTaskSlots:
taskmanager.slot.timeout:
taskmanager.network.memory.fraction:

`taskmanager.network.memory.max:`

`taskmanager.network.request-backoff.initial:`

`taskmanager.network.request-backoff.max:`

`taskmanager.network.memory.buffer-debloat.enabled:`

`taskmanager.network.memory.buffer-debloat.period:`

`taskmanager.network.memory.buffer-debloat.samples:`

`taskmanager.network.memory.buffer-debloat.threshold-percentages:`

Memory configuration

`taskmanager.memory.jvm-metaspace.size:`

`taskmanager.memory.jvm-overhead.fraction:`

`taskmanager.memory.jvm-overhead.max:`

`taskmanager.memory.managed.consumer-weights:`

`taskmanager.memory.managed.fraction:`

`taskmanager.memory.network.fraction:`

`taskmanager.memory.network.max:`

`taskmanager.memory.segment-size:`

`taskmanager.memory.task.off-heap.size:`

RPC / Akka

`akka.ask.timeout:`

`akka.client.timeout:`

`akka.framesize:`

`akka.lookup.timeout:`

`akka.tcp.timeout:`

Client

`client.timeout:`

Advanced cluster options

`cluster.intercept-user-system-exit:`

`cluster.processes.halt-on-fatal-error:`

Filesystem configurations

`fs.s3.connection.maximum:`

`fs.s3a.connection.maximum:`

`fs.s3a.threads.max:`

`s3.upload.max.concurrent.uploads:`

Advanced fault tolerance options

`heartbeat.timeout:`

`jobmanager.execution.failover-strategy:`

Memory configuration

`jobmanager.memory.heap.size:`

Metrics

`metrics.latency.interval:`

Advanced options for the REST endpoint and client

`rest.flamegraph.enabled:`

`rest.server.numThreads:`

Advanced SSL security options

`security.ssl.internal.handshake-timeout:`

Advanced scheduling options

`slot.request.timeout:`

Advanced options for Flink web UI

`web.timeout:`

Viewing configured Flink properties

You can view Apache Flink properties you have configured yourself or requested to be modified through a [support case](#) via the Apache Flink Dashboard and following these steps:

1. Go to the Flink Dashboard
2. Choose **Job Manager** in the left-hand side navigation pane.
3. Choose **Configuration** to view the list of Flink properties.

Configuring Managed Service for Apache Flink to access resources in an Amazon VPC

You can configure a Managed Service for Apache Flink 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](#)
- [VPC application permissions](#)
- [Internet and service access for a VPC-connected Managed Service for Apache Flink application](#)
- [Managed Service for Apache Flink VPC API](#)
- [Example: Using a VPC to access data in an Amazon MSK cluster](#)

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](#).

Managed Service for Apache Flink 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](#).

Note

VPC configurations are not supported for SQL applications.

Note

The Managed Service for Apache Flink 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 for Amazon Managed Service for Apache Flink](#).

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 Managed Service for Apache Flink application

By default, when you connect a Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application with private subnets:

- Open the Managed Service for Apache Flink console at <https://console.aws.amazon.com/flink>
- On the **Managed Service for Apache Flink 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](#)

Managed Service for Apache Flink VPC API

Use the following Managed Service for Apache Flink API operations to manage VPCs for your application. For information on using the Managed Service for Apache Flink API, see [API example code](#).

Create application

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_15",
  "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 DeleteApplicationVpcConfiguration action removes an existing VPC configuration from an application:

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 9,
  "VpcConfigurationId": "1.1"
}
```

Update application

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](#).

Troubleshooting Managed Service for Apache Flink

The following can help you troubleshoot problems that you might encounter with Amazon Managed Service for Apache Flink.

Topics

- [Development troubleshooting](#)
- [Runtime troubleshooting](#)

Development troubleshooting

Topics

- [FlinkRuntimeException: "Not allowed configuration change\(s\) were detected"](#)
- [System rollback best practices](#)
- [Hudi configuration best practices](#)
- [Apache Flink Flame Graphs](#)
- [Credential provider issue with EFO connector 1.15.2](#)
- [Applications with unsupported Kinesis connectors](#)
- [Compile error: "Could not resolve dependencies for project"](#)
- [Invalid choice: "kinesisanalyticsv2"](#)
- [UpdateApplication action isn't reloading application code](#)
- [S3 StreamingFileSink FileNotFoundExceptions](#)
- [FlinkKafkaConsumer issue with stop with savepoint](#)
- [Flink 1.15 Async Sink Deadlock](#)
- [Amazon Kinesis data streams source processing out of order during re-sharding](#)

FlinkRuntimeException: "Not allowed configuration change(s) were detected"

From Flink version 1.19 and later, we have disabled Flink job configuration changes using Flink job codes.

If you perform such an unsupported configuration change, you receive the following exception:

```
Caused by: org.apache.flink.client.program.ProgramInvocationException: The main method
  caused an error: Not allowed configuration change(s) were detected:
  - Configuration execution.checkpointing.unaligned.enabled:true not allowed.
  - Setter CheckpointConfig#setCheckpointStorage has been used
    at
  org.apache.flink.client.program.PackagedProgram.callMainMethod(PackagedProgram.java:372)
    at
  org.apache.flink.client.program.PackagedProgram.invokeInteractiveModeForExecution(PackagedProgram.java:108)
    at org.apache.flink.client.ClientUtils.executeProgram(ClientUtils.java:108)
    at
  org.apache.flink.client.deployment.application.DetachedApplicationRunner.tryExecuteJobs(DetachedApplicationRunner.java:108)
    ... 9 more
```

The exception message indicates which not allowed configurations or setters you have used. In the previous example, the application attempts to configure `execution.checkpointing.unaligned.enabled` and to use `setCheckpointStorage`.

Configurations that you can continue to modify in code include:

1. `pipeline.auto-watermark-interval`
2. `pipeline.closure-cleaner-level`
3. `pipeline.max-parallelism`
4. `pipeline.name`
5. `pipeline.operator-chaining.chain-operators-with-different-max-parallelism`
6. `pipeline.operator-chaining.enabled`
7. `pipeline.vertex-description-mode`
8. `pipeline.vertex-name-include-index-prefix`
9. `python.execution-mode`
10. `python.operator-chaining.enabled`

You can still request modifications to some job or cluster configurations using a support case. For more information, see [Modifiable Flink configuration properties](#).

System rollback best practices

With automatic system rollback and operations visibility capabilities in Amazon Managed Service for Apache Flink, you can identify and resolve issues with your applications.

System rollbacks

If your application update or scaling operation fails due to a customer error, such as a code bug or permission issue, Amazon Managed Service for Apache Flink automatically attempts to roll back to the previous running version if you have opted in to this functionality. For more information, see [Enabling system rollbacks for your Managed Service for Apache Flink application](#). If this autorollback fails or you have not opted in or opted out, your application will be placed into the READY state. To update your application, complete the following steps:

Manual rollback

If the application is not progressing and is in a transient state for long, or if the application successfully transitioned to Running, but you see downstream issues like processing errors in a successfully updated Flink application, you can manually roll it back using the `RollbackApplication` API.

1. Call `RollbackApplication` - this will revert to the previous running version and restore the previous state.
2. Monitor the rollback operation using the `DescribeApplicationOperation` API.
3. If rollback fails, use the previous system rollback steps.

Operations visibility

The `ListApplicationOperations` API shows the history of all customer and system operations on your application.

1. Get the *operationId* of the failed operation from the list.
2. Call `DescribeApplicationOperation` and check the status and *statusDescription*.
3. If an operation failed, the description points to a potential error to investigate.

Common error code bugs: Use the rollback capabilities to revert to the last working version. Resolve bugs and retry the update.

Permission issues: Use the `DescribeApplicationOperation` to see the required permissions. Update application permissions and retry.

Amazon Managed Service for Apache Flink service issues: Check the AWS Health Dashboard or open a support case.

Hudi configuration best practices

To run Hudi connectors on Managed Service for Apache Flink we recommend the following configuration changes.

Disable `hoodie.embed.timeline.server`

Hudi connector on Flink sets up an embedded timeline (TM) server on the Flink jobmanager (JM) to cache metadata to improve performance when job parallelism is high. We recommend that you disable this embedded server on Managed Service for Apache Flink because we disable non-Flink communication between JM and TM.

If this server is enabled, Hudi writes will first attempt to connect to the embedded server on JM, and then fall back to reading metadata from Amazon S3. This means that Hudi incurs a connection timeout that delays Hudi writes and causes a performance impact on Managed Service for Apache Flink.

Apache Flink Flame Graphs

Flame Graphs are enabled by default on applications in Managed Service for Apache Flink versions that support it. Flame Graphs may affect application performance if you keep the graph open, as mentioned in [Flink documentation](#).

If you want to disable Flame Graphs for your application, create a case to request it to be disabled for your application ARN. For more information, see the [AWS Support Center](#).

Credential provider issue with EFO connector 1.15.2

There is a [known issue](#) with Kinesis Data Streams EFO connector versions up to 1.15.2 where the `FlinkKinesisConsumer` is not respecting `CredentialProvider` configuration. Valid configurations are being disregarded due to the issue, which results in the `AUTO` credential provider being used. This can cause a problem using cross-account access to Kinesis using EFO connector.

To resolve this error please use EFO connector version 1.15.3 or higher.

Applications with unsupported Kinesis connectors

Managed Service for Apache Flink for Apache Flink version 1.15 or later will [automatically reject applications from starting or updating](#) if they are using unsupported Kinesis Connector versions (pre-version 1.15.2) bundled into application JARs or archives (ZIP).

Rejection error

You will see the following error when submitting create / update application calls through:

```
An error occurred (InvalidArgumentException) when calling the CreateApplication operation: An unsupported Kinesis connector version has been detected in the application. Please update flink-connector-kinesis to any version equal to or newer than 1.15.2.
For more information refer to connector fix: https://issues.apache.org/jira/browse/FLINK-23528
```

Steps to remediate

- Update the application's dependency on `flink-connector-kinesis`. If you are using Maven as your project's build tool, follow [Update a Maven dependency](#). If you are using Gradle, follow [Update a Gradle dependency](#).
- Repackage the application.
- Upload to an Amazon S3 bucket.
- Resubmit the create / update application request with the revised application just uploaded to the Amazon S3 bucket.
- If you continue to see the same error message, re-check your application dependencies. If the problem persists please create a support ticket.

Update a Maven dependency

1. Open the project's `pom.xml`.
2. Find the project's dependencies. They look like:

```
<project>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-connector-kinesis</artifactId>
```

```
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

3. Update `flink-connector-kinesis` to a version that is equal to or newer than 1.15.2. For instance:

```
<project>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-connector-kinesis</artifactId>
      <version>1.15.2</version>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

Update a Gradle dependency

1. Open the project's `build.gradle` (or `build.gradle.kts` for Kotlin applications).
2. Find the project's dependencies. They look like:

```
...
```

```
dependencies {  
    ...  
    implementation("org.apache.flink:flink-connector-kinesis")  
    ...  
}  
...
```

3. Update `flink-connector-kinesis` to a version that is equal to or newer than 1.15.2. For instance:

```
...  
dependencies {  
    ...  
    implementation("org.apache.flink:flink-connector-kinesis:1.15.2")  
    ...  
}  
...
```

Compile error: "Could not resolve dependencies for project"

In order to compile the Managed Service 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"](#).

Invalid choice: "kinesisanalyticsv2"

To use v2 of the Managed Service for Apache Flink 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](#).

S3 StreamingFileSink FileNotFoundExceptions

Managed Service for Apache Flink applications can run into In-progress part file `FileNotFoundException` when starting from snapshots if an In-progress part file referred to by its savepoint is missing. When this failure mode occurs, the Managed Service for Apache Flink application's operator state is usually non-recoverable and must be restarted without snapshot using `SKIP_RESTORE_FROM_SNAPSHOT`. See following example stacktrace:

```
java.io.FileNotFoundException: No such file or directory: s3://your-s3-bucket/pathj/
INSERT/2023/4/19/7/_part-2-1234_tmp_12345678-1234-1234-1234-123456789012
    at
    org.apache.hadoop.fs.s3a.S3AFileSystem.s3GetFileStatus(S3AFileSystem.java:2231)
    at
    org.apache.hadoop.fs.s3a.S3AFileSystem.innerGetFileStatus(S3AFileSystem.java:2149)
    at
    org.apache.hadoop.fs.s3a.S3AFileSystem.getFileStatus(S3AFileSystem.java:2088)
    at org.apache.hadoop.fs.s3a.S3AFileSystem.open(S3AFileSystem.java:699)
    at org.apache.hadoop.fs.FileSystem.open(FileSystem.java:950)
```

```
        at
org.apache.flink.fs.s3hadoop.HadoopS3AccessHelper.getObject(HadoopS3AccessHelper.java:98)
        at
org.apache.flink.fs.s3.common.writer.S3RecoverableMultipartUploadFactory.recoverInProgressPart
...

```

Flink `StreamingFileSink` writes records to filesystems supported by the [File Systems](#). Given that the incoming streams can be unbounded, data is organized into part files of finite size with new files added as data is written. Part lifecycle and rollover policy determine the timing, size and the naming of the part files.

During checkpointing and savepointing (snapshotting), all Pending files are renamed and committed. However, In-progress part files are not committed but renamed and their reference is kept within checkpoint or savepoint metadata to be used when restoring jobs. These In-progress part files will eventually rollover to Pending, renamed and committed by a subsequent checkpoint or savepoint.

Following are the root causes and mitigation for missing In-progress part file:

- Stale snapshot used to start the Managed Service for Apache Flink application – only the latest system snapshot taken when an application is stopped or updated can be used to start a Managed Service for Apache Flink application with Amazon S3 `StreamingFileSink`. To avoid this class of failure, use the latest system snapshot.
 - This happens for example when you pick a snapshot created using `CreateSnapshot` instead of a system-triggered `Snapshot` during stop or update. The older snapshot's savepoint keeps an out-of-date reference to In-progress part file that has been renamed and committed by subsequent checkpoint or savepoint.
 - This can also happen when a system triggered snapshot from non-latest `Stop/Update` event is picked. An example is an application with system snapshot disabled but has `RESTORE_FROM_LATEST_SNAPSHOT` configured. Generally, Managed Service for Apache Flink applications with Amazon S3 `StreamingFileSink` should always have system snapshot enabled and `RESTORE_FROM_LATEST_SNAPSHOT` configured.
- In-progress part file removed – As the In-progress part file is located in an S3 bucket, it can be removed by other components or actors which have access to the bucket.
 - This can happen when you have stopped your app for too long and the In-progress part file referred to by your app's savepoint has been removed by [S3 bucket MultiPartUpload](#) lifecycle policy. To avoid this class of failure, make sure that your S3 Bucket MPU lifecycle policy covers a sufficiently large period for your use case.

- This can also happen when the In-progress part file has been removed manually or by another one of your system's components. To avoid this class of failure, please make sure that In-progress part files are not removed by other actors or components.
- Race condition where an automated checkpoint is triggered after savepoint – This affects Managed Service for Apache Flink versions up to and including 1.13. This issue is fixed in Managed Service for Apache Flink version 1.15. Migrate your application to the latest version of Managed Service for Apache Flink to prevent recurrence. We also suggest migrating from `StreamingFileSink` to [FileSink](#).
- When applications are stopped or updated, Managed Service for Apache Flink triggers a savepoint and stops the application in two steps. If an automated checkpoint triggers between the two steps, the savepoint will be unusable as its In-progress part file would be renamed and potentially committed.

FlinkKafkaConsumer issue with stop with savepoint

When using the legacy `FlinkKafkaConsumer` there is a possibility your application may get stuck in `UPDATING`, `STOPPING` or `SCALING`, if you have system snapshots enabled. There is no published fix available for this [issue](#), therefore we recommend you upgrade to the new [KafkaSource](#) to mitigate this issue.

If you are using the `FlinkKafkaConsumer` with snapshots enabled, there is a possibility when the Flink job processes a stop with savepoint API request, the `FlinkKafkaConsumer` can fail with a runtime error reporting a `IOException`. Under these conditions the Flink application becomes stuck, manifesting as Failed Checkpoints.

Flink 1.15 Async Sink Deadlock

There is a [known issue](#) with AWS connectors for Apache Flink implementing `AsyncSink` interface. This affects applications using Flink 1.15 with the following connectors:

- For Java applications:
 - `KinesisStreamsSink` – `org.apache.flink:flink-connector-kinesis`
 - `KinesisStreamsSink` – `org.apache.flink:flink-connector-aws-kinesis-streams`
 - `KinesisFirehoseSink` – `org.apache.flink:flink-connector-aws-kinesis-firehose`
 - `DynamoDbSink` – `org.apache.flink:flink-connector-dynamodb`

- Flink SQL/TableAPI/Python applications:
 - `kinesis – org.apache.flink:flink-sql-connector-kinesis`
 - `kinesis – org.apache.flink:flink-sql-connector-aws-kinesis-streams`
 - `firehose – org.apache.flink:flink-sql-connector-aws-kinesis-firehose`
 - `dynamodb – org.apache.flink:flink-sql-connector-dynamodb`

Affected applications will experience the following symptoms:

- Flink job is in `RUNNING` state, but not processing data;
- There are no job restarts;
- Checkpoints are timing out.

The issue is caused by a [bug](#) in AWS SDK resulting in it not surfacing certain errors to the caller when using the async HTTP client. This results in the sink waiting indefinitely for an “in-flight request” to complete during a checkpoint flush operation.

This issue had been fixed in AWS SDK starting from version **2.20.144**.

Following are instructions on how to update affected connectors to use the new version of AWS SDK in your applications:

Topics

- [Update Java applications](#)
- [Update Python applications](#)

Update Java applications

Follow the procedures below to update Java applications:

flink-connector-kinesis

If the application uses `flink-connector-kinesis`:

Kinesis connector uses shading to package some dependencies, including the AWS SDK, into the connector jar. To update the AWS SDK version, use the following procedure to replace these shaded classes:

Maven

1. Add Kinesis connector and required AWS SDK modules as project dependencies.
2. Configure maven-shade-plugin:
 - a. Add filter to exclude shaded AWS SDK classes when copying content of the Kinesis connector jar.
 - b. Add relocation rule to move updated AWS SDK classes to package, expected by Kinesis connector.

pom.xml

```
<project>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-connector-kinesis</artifactId>
      <version>1.15.4</version>
    </dependency>

    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>kinesis</artifactId>
      <version>2.20.144</version>
    </dependency>

    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>netty-nio-client</artifactId>
      <version>2.20.144</version>
    </dependency>

    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>sts</artifactId>
      <version>2.20.144</version>
    </dependency>
    ...
  </dependencies>
  ...
</build>
```

```

...
<plugins>
  ...
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>3.1.1</version>
    <executions>
      <execution>
        <phase>package</phase>
        <goals>
          <goal>shade</goal>
        </goals>
        <configuration>
          ...
          <filters>
            ...
            <filter>
              <artifact>org.apache.flink:flink-connector-
kinesis</artifact>
              <excludes>
                <exclude>org/apache/flink/kinesis/
shaded/software/amazon/awssdk/**</exclude>
                <exclude>org/apache/flink/kinesis/
shaded/org/reactivestreams/**</exclude>
                <exclude>org/apache/flink/kinesis/
shaded/io/netty/**</exclude>
                <exclude>org/apache/flink/kinesis/
shaded/com/typesafe/netty/**</exclude>
              </excludes>
            </filter>
            ...
          </filters>
          <relocations>
            ...
            <relocation>
              <pattern>software.amazon.awssdk</pattern>
              <shadedPattern>org.apache.flink.kinesis.shaded.software.amazon.awssdk</
shadedPattern>
            </relocation>
            <relocation>
              <pattern>org.reactivestreams</pattern>

```

```

    <shadedPattern>org.apache.flink.kinesis.shaded.org.reactivestreams</
shadedPattern>
        </relocation>
        <relocation>
            <pattern>io.netty</pattern>

    <shadedPattern>org.apache.flink.kinesis.shaded.io.netty</shadedPattern>
        </relocation>
        <relocation>
            <pattern>com.typesafe.netty</pattern>

    <shadedPattern>org.apache.flink.kinesis.shaded.com.typesafe.netty</
shadedPattern>
        </relocation>
        ...
    </relocations>
    ...
    </configuration>
    </execution>
    </executions>
    </plugin>
    ...
    </plugins>
    ...
    </build>
</project>

```

Gradle

1. Add Kinesis connector and required AWS SDK modules as project dependencies.
2. Adjust shadowJar configuration:
 - a. Exclude shaded AWS SDK classes when copying content of the Kinesis connector jar.
 - b. Relocate updated AWS SDK classes to a package expected by Kinesis connector.

build.gradle

```

...
dependencies {
    ...

```

```

    flinkShadowJar("org.apache.flink:flink-connector-kinesis:1.15.4")

    flinkShadowJar("software.amazon.awssdk:kinesis:2.20.144")
    flinkShadowJar("software.amazon.awssdk:sts:2.20.144")
    flinkShadowJar("software.amazon.awssdk:netty-nio-client:2.20.144")
    ...
}
...
shadowJar {
    configurations = [project.configurations.flinkShadowJar]

    exclude("software/amazon/kinesis/shaded/software/amazon/awssdk/**/*")
    exclude("org/apache/flink/kinesis/shaded/org/reactivestreams/**/* .class")
    exclude("org/apache/flink/kinesis/shaded/io/netty/**/* .class")
    exclude("org/apache/flink/kinesis/shaded/com/typesafe/netty/**/* .class")

    relocate("software.amazon.awssdk",
"org.apache.flink.kinesis.shaded.software.amazon.awssdk")
    relocate("org.reactivestreams",
"org.apache.flink.kinesis.shaded.org.reactivestreams")
    relocate("io.netty", "org.apache.flink.kinesis.shaded.io.netty")
    relocate("com.typesafe.netty",
"org.apache.flink.kinesis.shaded.com.typesafe.netty")
}
...

```

Other affected connectors

If the application uses another affected connector:

In order to update the AWS SDK version, the SDK version should be enforced in the project build configuration.

Maven

Add AWS SDK bill of materials (BOM) to the dependency management section of the `pom.xml` file to enforce SDK version for the project.

pom.xml

```

<project>
    ...
    <dependencyManagement>

```

```
<dependencies>
  ...
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>bom</artifactId>
    <version>2.20.144</version>
    <scope>import</scope>
    <type>pom</type>
  </dependency>
  ...
</dependencies>
</dependencyManagement>
...
</project>
```

Gradle

Add platform dependency on the AWS SDK bill of materials (BOM) to enforce SDK version for the project. This requires Gradle 5.0 or newer:

build.gradle

```
...
dependencies {
  ...
  flinkShadowJar(platform("software.amazon.awssdk:bom:2.20.144"))
  ...
}
...
```

Update Python applications

Python applications can use connectors in 2 different ways: packaging connectors and other Java dependencies as part of single uber-jar, or use connector jar directly. To fix applications affected by Async Sink deadlock:

- If the application uses an uber jar, follow the instructions for [Update Java applications](#).
- To rebuild connector jars from source, use the following steps:

Building connectors from source:

Prerequisites, similar to Flink [build requirements](#):

- Java 11
- Maven 3.2.5

flink-sql-connector-kinesis

1. Download source code for Flink 1.15.4:

```
wget https://archive.apache.org/dist/flink/flink-1.15.4/flink-1.15.4-src.tgz
```

2. Uncompress source code:

```
tar -xvf flink-1.15.4-src.tgz
```

3. Navigate to kinesis connector directory

```
cd flink-1.15.4/flink-connectors/flink-connector-kinesis/
```

4. Compile and install connector jar, specifying required AWS SDK version. To speed up build use `-DskipTests` to skip test execution and `-Dfast` to skip additional source code checks:

```
mvn clean install -DskipTests -Dfast -Daws.sdkv2.version=2.20.144
```

5. Navigate to kinesis connector directory

```
cd ../flink-sql-connector-kinesis
```

6. Compile and install sql connector jar:

```
mvn clean install -DskipTests -Dfast
```

7. Resulting jar will be available at:

```
target/flink-sql-connector-kinesis-1.15.4.jar
```

flink-sql-connector-aws-kinesis-streams

1. Download source code for Flink 1.15.4:

```
wget https://archive.apache.org/dist/flink/flink-1.15.4/flink-1.15.4-src.tgz
```

2. Uncompress source code:

```
tar -xvf flink-1.15.4-src.tgz
```

3. Navigate to kinesis connector directory

```
cd flink-1.15.4/flink-connectors/flink-connector-aws-kinesis-streams/
```

4. Compile and install connector jar, specifying required AWS SDK version. To speed up build use `-DskipTests` to skip test execution and `-Dfast` to skip additional source code checks:

```
mvn clean install -DskipTests -Dfast -Daws.sdk.version=2.20.144
```

5. Navigate to kinesis connector directory

```
cd ../flink-sql-connector-aws-kinesis-streams
```

6. Compile and install sql connector jar:

```
mvn clean install -DskipTests -Dfast
```

7. Resulting jar will be available at:

```
target/flink-sql-connector-aws-kinesis-streams-1.15.4.jar
```

flink-sql-connector-aws-kinesis-firehose

1. Download source code for Flink 1.15.4:

```
wget https://archive.apache.org/dist/flink/flink-1.15.4/flink-1.15.4-src.tgz
```

2. Uncompress source code:

```
tar -xvf flink-1.15.4-src.tgz
```

3. Navigate to connector directory

```
cd flink-1.15.4/flink-connectors/flink-connector-aws-kinesis-firehose/
```

4. Compile and install connector jar, specifying required AWS SDK version. To speed up build use `-DskipTests` to skip test execution and `-Dfast` to skip additional source code checks:

```
mvn clean install -DskipTests -Dfast -Daws.sdk.version=2.20.144
```

5. Navigate to sql connector directory

```
cd ../flink-sql-connector-aws-kinesis-firehose
```

6. Compile and install sql connector jar:

```
mvn clean install -DskipTests -Dfast
```

7. Resulting jar will be available at:

```
target/flink-sql-connector-aws-kinesis-firehose-1.15.4.jar
```

flink-sql-connector-dynamodb

1. Download source code for Flink 1.15.4:

```
wget https://archive.apache.org/dist/flink/flink-connector-aws-3.0.0/flink-connector-aws-3.0.0-src.tgz
```

2. Uncompress source code:

```
tar -xvf flink-connector-aws-3.0.0-src.tgz
```

3. Navigate to connector directory

```
cd flink-connector-aws-3.0.0
```

4. Compile and install connector jar, specifying required AWS SDK version. To speed up build use `-DskipTests` to skip test execution and `-Dfast` to skip additional source code checks:

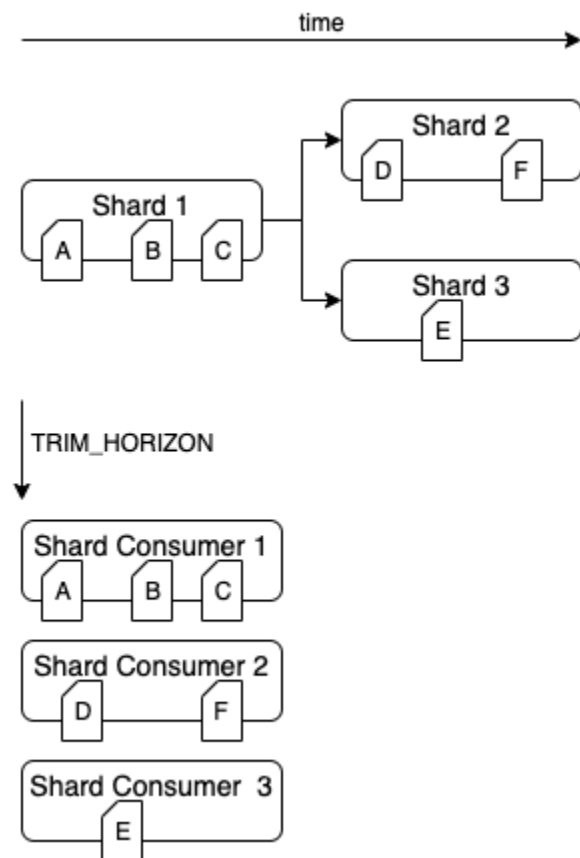
```
mvn clean install -DskipTests -Dfast -Dflink.version=1.15.4 -  
Daws.sdk.version=2.20.144
```


5. Resulting jar will be available at:

```
flink-sql-connector-dynamodb/target/flink-sql-connector-dynamodb-3.0.0.jar
```

Amazon Kinesis data streams source processing out of order during re-sharding

The current `FlinkKinesisConsumer` implementation doesn't provide strong ordering guarantees between Kinesis shards. This may lead to out-of-order processing during re-sharding of Kinesis Stream, in particular for Flink applications that experience processing lag. Under some circumstances, for example windows operators based on event times, events might get discarded because of the resulting lateness.



This is a [known problem](#) in Open Source Flink. Until connector fix is made available, ensure your Flink applications are not falling behind Kinesis Data Streams during re-partitioning. By ensuring that the processing delay is tolerated by your Flink apps, you can minimize the impact of out-of-order processing and risk of data loss.

Runtime troubleshooting

This section contains information about diagnosing and fixing runtime issues with your Managed Service for Apache Flink application.

Topics

- [Troubleshooting tools](#)
- [Application issues](#)
- [Application is restarting](#)
- [Throughput is too slow](#)
- [Unbounded state growth](#)
- [I/O bound operators](#)
- [Upstream or source throttling from a Kinesis data stream](#)
- [Checkpoints](#)
- [Checkpointing is timing out](#)
- [Checkpoint failure for Apache Beam application](#)
- [Backpressure](#)
- [Data skew](#)
- [State skew](#)
- [Integrating with resources in different Regions](#)

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 Managed Service for Apache Flink](#).

Application issues

This section contains solutions for error conditions that you may encounter with your Managed Service for Apache Flink application.

Topics

- [Application is stuck in a transient status](#)
- [Snapshot creation fails](#)
- [Cannot access resources in a VPC](#)
- [Data is lost when writing to an Amazon S3 bucket](#)
- [Application is in the RUNNING status but isn't processing data](#)
- [Snapshot, application update, or application stop error: InvalidApplicationConfigurationException](#)
- [java.nio.file.NoSuchFileException: /usr/local/openjdk-8/lib/security/cacerts](#)

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, Managed Service for Apache Flink 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:** Managed Service for Apache Flink 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](#) 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:** Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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](#).
- 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](#).

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](#).

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 Firehose. For more information about using Firehose with Managed Service for Apache Flink, see [Firehose sink](#).

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](#). 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 Managed Service for
Apache Flink 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 Managed Service for Apache Flink 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](#).
- 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](#).

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"](#).

- **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](#).
- **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](#).
- 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. Managed Service for Apache Flink 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 Managed Service for Apache Flink, see [Using CloudFormation with Managed Service for Apache Flink](#).

- **Application fails with `java.io.IOException: Insufficient number of network buffers`**

This happens when an application does not have enough memory allocated for network buffers. Network buffers facilitate communication between subtasks. They are used to store records before transmission over a network, and to store incoming data before dissecting it into records and handing them to subtasks. The number of network buffers required scales directly with the parallelism and complexity of your job graph. There are a number of approaches to mitigate this issue:

- You can configure a lower `parallelismPerKpu` so that there is more memory allocated per-subtask and network buffers. Note that lowering `parallelismPerKpu` will increase KPU and therefore cost. To avoid this, you can keep the same amount of KPU by lowering parallelism by the same factor.
- You can simplify your job graph by reducing the number of operators or chaining them so that fewer buffers are needed.

- Otherwise, you can reach out to <https://aws.amazon.com/premiumsupport/> for custom network buffer configuration.

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](#).
- 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](#).
- 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](#).
- If your application reads from a Kafka cluster in a different Region and `FlinkKafkaConsumer` or `KafkaSource` are mostly idle (high `idleTimeMsPerSecond` or low `CPUUtilization`)

despite high consumer lag, you can increase the value for `receive.buffer.byte`, such as 2097152. For more information, see the high latency environment section in [Custom MSK configurations](#).

For troubleshooting steps for slow throughput or consumer lag increasing in the application source, see [Troubleshooting performance](#).

Unbounded state growth

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](#).

Sometimes applications are facing ever growing state size growth, which is not sustainable in the long term (a Flink application runs indefinitely, after all). Sometimes, this can be traced back to applications storing data in state and not aging out old information properly. But sometimes there are just unreasonable expectations on what Flink can deliver. Applications can use aggregations

over large time windows spanning days or even weeks. Unless [AggregateFunctions](#) are used, which allow incremental aggregations, Flink needs to keep the events of the entire window in state.

Moreover, when using process functions to implement custom operators, the application needs to remove data from state that is no longer required for the business logic. In that case, [state time-to-live](#) can be used to automatically age out data based on processing time. Managed Service for Apache Flink is using incremental checkpoints and thus state ttl is based on [RocksDB compaction](#). You can only observe an actual reduction in state size (indicated by checkpoint size) after a compaction operation occurs. In particular for checkpoint sizes below 200 MB, it's unlikely that you observe any checkpoint size reduction as a result of state expiring. However, savepoints are based on a clean copy of the state that does not contain old data, so you can trigger a snapshot in Managed Service for Apache Flink to force the removal of outdated state.

For debugging purposes, it can make sense to disable incremental checkpoints to verify more quickly that the checkpoint size actually decreases or stabilizes (and avoid the effect of compaction in RocksDB). This requires a ticket to the service team, though.

I/O bound operators

It's best to avoid dependencies to external systems on the data path. It's often much more performant to keep a reference data set in state rather than querying an external system to enrich individual events. However, sometimes there are dependencies that cannot be easily moved to state, e.g., if you want to enrich events with a machine learning model that is hosted on Amazon Sagemaker.

Operators that are interfacing with external systems over the network can become a bottleneck and cause backpressure. It is highly recommended to use [AsyncIO](#) to implement the functionality, to reduce the wait time for individual calls and avoid the entire application slowing down.

Moreover, for applications with I/O bound operators it can also make sense to increase the [ParallelismPerKPU](#) setting of the Managed Service for Apache Flink application. This configuration describes the number of parallel subtasks an application can perform per Kinesis Processing Unit (KPU). By increasing the value from the default of 1 to, say, 4, the application leverages the same resources (and has the same cost) but can scale to 4 times the parallelism. This works well for I/O bound applications, but it causes additional overhead for applications that are not I/O bound.

Upstream or source throttling from a Kinesis data stream

Symptom: The application is encountering `LimitExceededExceptions` from their upstream source Kinesis data stream.

Potential Cause: The default setting for the Apache Flink library Kinesis connector is set to read from the Kinesis data stream source with a very aggressive default setting for the maximum number of records fetched per `GetRecords` call. Apache Flink is configured by default to fetch 10,000 records per `GetRecords` call (this call is made by default every 200 ms), although the limit per shard is only 1,000 records.

This default behavior can lead to throttling when attempting to consume from the Kinesis data stream, which will affect the applications performance and stability.

You can confirm this by checking the CloudWatch `ReadProvisionedThroughputExceeded` metric and seeing prolonged or sustained periods where this metric is greater than zero.

You can also see this in CloudWatch logs for your Amazon Managed Service for Apache Flink application by observing continued `LimitExceededException` errors.

Resolution: You can do one of two things to resolve this scenario:

- Lower the default limit for the number of records fetched per `GetRecords` call
- Enable Adaptive Reads in your Amazon Managed Service for Apache Flink application. For more information on the Adaptive Reads feature, see [SHARD_USE_ADAPTIVE_READS](#)

Checkpoints

Checkpoints are Flink's mechanism to ensure that the state of an application is fault tolerant. The mechanism allows Flink to recover the state of operators if the job fails and gives the application the same semantics as failure-free execution. With Managed Service for Apache Flink, the state of an application is stored in RocksDB, an embedded key/value store that keeps its working state on disk. When a checkpoint is taken the state is also uploaded to Amazon S3 so even if the disk is lost then the checkpoint can be used to restore the applications state.

For more information, see [How does State Snapshotting Work?](#).

Checkpointing stages

For a checkpointing operator subtask in Flink there are 5 main stages:

- **Waiting [Start Delay]** – Flink uses checkpoint barriers that get inserted into the stream so time in this stage is the time the operator waits for the checkpoint barrier to reach it.
- **Alignment [Alignment Duration]** – In this stage the subtask has reached one barrier but it's waiting for barriers from other input streams.

- Sync checkpointing [**Sync Duration**] – This stage is when the subtask actually snapshots the state of the operator and blocks all other activity on the subtask.
- Async checkpointing [**Async Duration**] – The majority of this stage is the subtask uploading the state to Amazon S3. During this stage, the subtask is no longer blocked and can process records.
- Acknowledging – This is usually a short stage and is simply the subtask sending an acknowledgement to the JobManager and also performing any commit messages (e.g. with Kafka sinks).

Each of these stages (apart from Acknowledging) maps to a duration metric for checkpoints that is available from the Flink WebUI, which can help isolate the cause of the long checkpoint.

To see an exact definition of each of the metrics available on checkpoints, go to [History Tab](#).

Investigating

When investigating long checkpoint duration, the most important thing to determine is the bottleneck for the checkpoint, i.e., what operator and subtask is taking the longest to checkpoint and which stage of that subtask is taking an extended period of time. This can be determined using the Flink WebUI under the jobs checkpoint task. Flink's Web interface provides data and information that helps to investigate checkpointing issues. For a full breakdown, see [Monitoring Checkpointing](#).

The first thing to look at is the **End to End Duration** of each operator in the Job graph to determine which operator is taking long to checkpoint and warrants further investigation. Per the Flink documentation, the definition of the duration is:

The duration from the trigger timestamp until the latest acknowledgement (or n/a if no acknowledgement received yet). This end to end duration for a complete checkpoint is determined by the last subtask that acknowledges the checkpoint. This time is usually larger than single subtasks need to actually checkpoint the state.

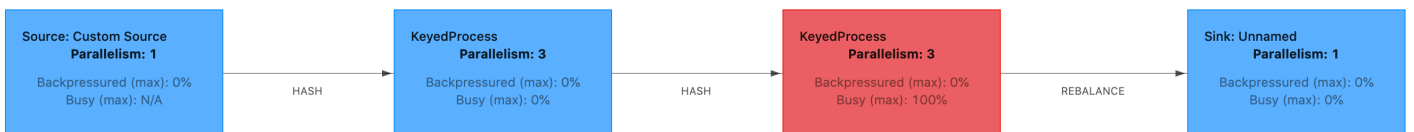
The other durations for the checkpoint also gives more fine-grained information as to where the time is being spent.

If the **Sync Duration** is high then this indicates something is happening during the snapshotting. During this stage `snapshotState()` is called for classes that implement the `snapshotState` interface; this can be user code so thread-dumps can be useful for investigating this.

A long **Async Duration** would suggest that a lot of time is being spent on uploading the state to Amazon S3. This can occur if the state is large or if there is a lot of state files that are being uploaded. If this is the case it is worth investigating how state is being used by the application and ensuring that the Flink native data structures are being used where possible ([Using Keyed State](#)). Managed Service for Apache Flink configures Flink in such a way as to minimize the number of Amazon S3 calls to ensure this doesn't get too long. Following is an example of an operator's checkpointing statistics. It shows that the **Async Duration** is relatively long compared to the preceding operator checkpointing statistics.

SubTasks:										
	End to End Duration	Checkpointed Data Size	Sync Duration	Async Duration	Processed (persisted) Data	Alignment Duration	Start Delay			
Minimum	495ms	11.1 KB	8ms	357ms	0 B (0 B)	0ms	126ms			
Average	813ms	586 KB	28ms	653ms	0 B (0 B)	0ms	126ms			
Maximum	1s	1.70 MB	69ms	1s	0 B (0 B)	1ms	128ms			
ID	Acknowledged	End to End Duration	Checkpointed Data Size	Sync Duration	Async Duration	Processed (persisted) Data	Alignment Duration	Start Delay	Unaligned Checkpoint	
0	2022-03-02 14:16:49	566ms	11.1 KB	8ms	429ms	0 B (0 B)	0ms	126ms	false	
1	2022-03-02 14:16:50	1s	1.70 MB	69ms	1s	0 B (0 B)	0ms	128ms	false	
2	2022-03-02 14:16:49	495ms	11.1 KB	8ms	357ms	0 B (0 B)	1ms	126ms	false	
Sink: Unnamed			1/1 (100%)	2022-03-02 14:16:49	131ms	0 B	0 B (0 B)			
SubTasks:										

The **Start Delay** being high would show that the majority of the time is being spent on waiting for the checkpoint barrier to reach the operator. This indicates that the application is taking a while to process records, meaning the barrier is flowing through the job graph slowly. This is usually the case if the Job is backpressured or if an operator(s) is constantly busy. Following is an example of a JobGraph where the second KeyedProcess operator is busy.



You can investigate what is taking so long by either using Flink Flame Graphs or TaskManager thread dumps. Once the bottle-neck has been identified, it can be investigated further using either Flame-graphs or thread-dumps.

Thread dumps

Thread dumps are another debugging tool that is at a slightly lower level than flame graphs. A thread dump outputs the execution state of all threads at a point in time. Flink takes a JVM thread dump, which is an execution state of all threads within the Flink process. The state of a thread is presented by a stack trace of the thread as well as some additional information. Flame graphs are actually built using multiple stack traces taken in quick succession. The graph is a visualisation made from these traces that makes it easy to identify the common code paths.

```
"KeyedProcess (1/3)#0" prio=5 Id=1423 RUNNABLE
  at app//scala.collection.immutable.Range.foreach$mVc$sp(Range.scala:154)
  at $line33.$read$$iw$$iw$ExpensiveFunction.processElement(<console>>19)
  at $line33.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:14)
  at app//
org.apache.flink.streaming.api.operators.KeyedProcessOperator.processElement(KeyedProcessOperator
  at app//org.apache.flink.streaming.runtime.tasks.OneInputStreamTask
$StreamTaskNetworkOutput.emitRecord(OneInputStreamTask.java:205)
  at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.processElement(AbstractStr
  at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.emitNext(AbstractStreamTas
  at app//
org.apache.flink.streaming.runtime.io.StreamOneInputProcessor.processInput(StreamOneInputProces
  ...
```

Above is a snippet of a thread dump taken from the Flink UI for a single thread. The first line contains some general information about this thread including:

- The thread name *KeyedProcess (1/3)#0*
- Priority of the thread *prio=5*
- A unique thread Id *Id=1423*
- Thread state *RUNNABLE*

The name of a thread usually gives information as to the general purpose of the thread. Operator threads can be identified by their name since operator threads have the same name as the

operator, as well as an indication of which subtask it is related to, e.g., the *KeyedProcess (1/3)#0* thread is from the *KeyedProcess* operator and is from the 1st (out of 3) subtask.

Threads can be in one of a few states:

- NEW – The thread has been created but has not yet been processed
- RUNNABLE – The thread is execution on the CPU
- BLOCKED – The thread is waiting for another thread to release its lock
- WAITING – The thread is waiting by using a `wait()`, `join()`, or `park()` method
- TIMED_WAITING – The thread is waiting by using a `sleep`, `wait`, `join` or `park` method, but with a maximum wait time.

Note

In Flink 1.13, the maximum depth of a single stacktrace in the thread dump is limited to 8.

Note

Thread dumps should be the last resort for debugging performance issues in a Flink application as they can be challenging to read, require multiple samples to be taken and manually analysed. If at all possible it is preferable to use flame graphs.

Thread dumps in Flink

In Flink, a thread dump can be taken by choosing the **Task Managers** option on the left navigation bar of the Flink UI, selecting a specific task manager, and then navigating to the **Thread Dump** tab. The thread dump can be downloaded, copied to your favorite text editor (or thread dump analyzer), or analyzed directly inside the text view in the Flink Web UI (however, this last option can be a bit clunky).

To determine which Task Manager to take a thread dump of the **TaskManagers** tab can be used when a particular operator is chosen. This shows that the operator is running on different subtasks of an operator and can run on different Task Managers.

The screenshot shows a Flink dashboard. On the left, a red box represents a 'KeyedProcess' operator with 'Parallelism: 3'. It shows 'Backpressured (max): 0%' and 'Busy (max): 100%'. A 'HASH' arrow points to it from the left, and a 'REBALANCE' arrow points to it from the right. On the right, a table titled 'TaskManagers' displays the following data:

Host	LOG	Bytes received	Records received	Bytes sent	Records sent	Status
ip-142-151-131-22:61 21	LOG	936 B	0	0 B	0	RUNNING
ip-142-151-146-195:6 121	LOG	103 KB	1,423	71.1 KB	1,422	RUNNING

The dump will be comprised of multiple stack traces. However when investigating the dump the ones related to an operator are the most important. These can easily be found since operator threads have the same name as the operator, as well as an indication of which subtask it is related to. For example the following stack trace is from the *KeyedProcess* operator and is the first subtask.

```
"KeyedProcess (1/3)#0" prio=5 Id=595 RUNNABLE
  at app//scala.collection.immutable.Range.foreach$mVc$sp(Range.scala:155)
  at $line360.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:19)
  at $line360.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:14)
  at app//
org.apache.flink.streaming.api.operators.KeyedProcessOperator.processElement(KeyedProcessOperator
  at app//org.apache.flink.streaming.runtime.tasks.OneInputStreamTask
$StreamTaskNetworkOutput.emitRecord(OneInputStreamTask.java:205)
  at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.processElement(AbstractStr
  at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.emitNext(AbstractStreamTas
  at app//
org.apache.flink.streaming.runtime.io.StreamOneInputProcessor.processInput(StreamOneInputProces
  ...
```

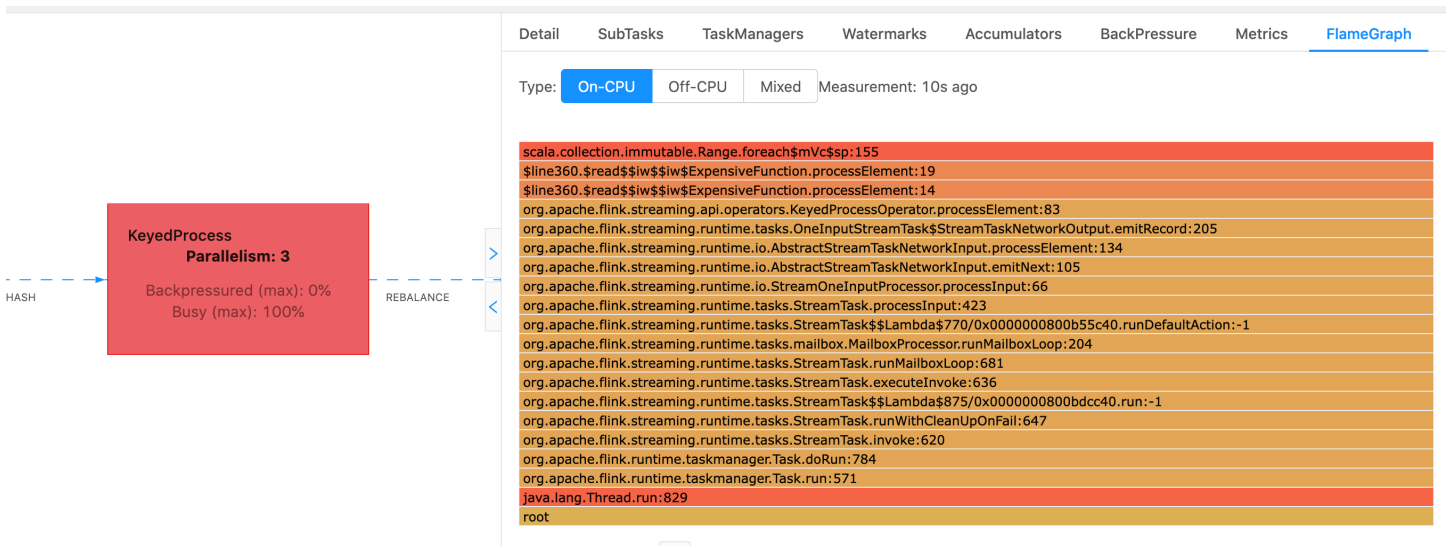
This can become confusing if there are multiple operators with the same name but we can name operators to get around this. For example:

```
....
.process(new ExpensiveFunction).name("Expensive function")
```

Flame graphs

Flame graphs are a useful debugging tool that visualize the stack traces of the targeted code, which allows the most frequent code paths to be identified. They are created by sampling stack traces a number of times. The x-axis of a flame graph shows the different stack profiles, while the y-axis shows the stack depth, and calls in the stack trace. A single rectangle in a flame graph represents on stack frame, and the width of a frame shows how frequently it appears in the stacks. For more details about flame graphs and how to use them, see [Flame Graphs](#).

In Flink, the flame graph for an operator can be accessed via the Web UI by selecting an operator and then choosing the **FlameGraph** tab. Once enough samples have been collected the flamegraph will be displayed. Following is the FlameGraph for the ProcessFunction that was taking a lot of time to checkpoint.



This is a very simple flame graph and shows that all the CPU time is being spent within a foreach look within the processElement of the ExpensiveFunction operator. You also get the line number to help determine where in the code execution is taking place.

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 [Data Types and Serialization](#) 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](#).
- 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.

Checkpoint failure for Apache Beam application

If your Beam application is configured with [shutdownSourcesAfterIdleMs](#) set to 0ms, checkpoints can fail to trigger because tasks are in "FINISHED" state. This section describes symptoms and resolution for this condition.

Symptom

Go to your Managed Service for Apache Flink application CloudWatch logs and check if the following log message has been logged. The following log message indicates that checkpoint failed to trigger as some tasks has been finished.

```
{
  "locationInformation":
    "org.apache.flink.runtime.checkpoint.CheckpointCoordinator.onTriggerFailure(CheckpointCoordinator",
    "logger": "org.apache.flink.runtime.checkpoint.CheckpointCoordinator",
    "message": "Failed to trigger checkpoint for job your job ID since some
tasks of job your job ID has been finished, abort the checkpoint Failure reason: Not
all required tasks are currently running.",
    "threadName": "Checkpoint Timer",
    "applicationARN": your application ARN,
    "applicationVersionId": "5",
    "messageSchemaVersion": "1",
    "messageType": "INFO"
}
```

This can also be found on Flink dashboard where some tasks have entered "FINISHED" state, and checkpointing is not possible anymore.

Detail	SubTasks	TaskManagers	Watermarks	Accumulators	BackPressure	Metrics	FlameGraph						
ID	Bytes Received	Records Received	Bytes Sent	Records Sent	Attempt	Host	Start Time	Duration	Status	More			
0	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	13m 57s	RUNNING	...			
1	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	3s	FINISHED	...			
2	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	3s	FINISHED	...			
3	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	3s	FINISHED	...			
4	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	3s	FINISHED	...			

Cause

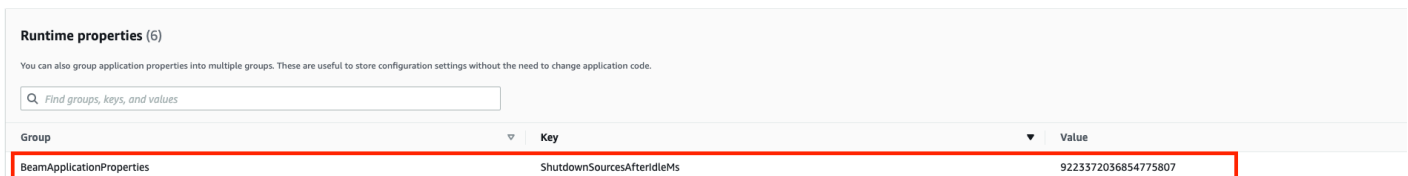
`shutdownSourcesAfterIdleMs` is a Beam config variable that shuts down sources which have been idle for the configured time of milliseconds. Once a source has been shut down, checkpointing is not possible anymore. This could lead to [checkpoint failure](#).

One of the causes for tasks entering "FINISHED" state is when `shutdownSourcesAfterIdleMs` is set to 0ms, which means that tasks that are idle will be shutdown immediately.

Solution

To prevent tasks from entering "FINISHED" state immediately, set `shutdownSourcesAfterIdleMs` to `Long.MAX_VALUE`. This can be done in two ways:

- Option 1: If your beam configuration is set in your Managed Service for Apache Flink application configuration page, then you can add a new key value pair to set `shutdpwnSourcesAfteridleMs` as follows:



Group	Key	Value
BeamApplicationProperties	ShutdownSourcesAfterIdleMs	9223372036854775807

- Option 2: If your beam configuration is set in your JAR file, then you can set `shutdownSourcesAfterIdleMs` as follows:

```
FlinkPipelineOptions options =
PipelineOptionsFactory.create().as(FlinkPipelineOptions.class); // Initialize Beam
Options object

options.setShutdownSourcesAfterIdleMs(Long.MAX_VALUE); // set
shutdownSourcesAfterIdleMs to Long.MAX_VALUE
options.setRunner(FlinkRunner.class);

Pipeline p = Pipeline.create(options); // attach specified
options to Beam pipeline
```

Backpressure

Flink uses backpressure to adapt the processing speed of individual operators.

The operator can struggle to keep up processing the message volume it receives for many reasons. The operation may require more CPU resources than the operator has available, The operator may wait for I/O operations to complete. If an operator cannot process events fast enough, it build backpressure in the upstream operators feeding into the slow operator. This causes the upstream operators to slow down, which can further propagate the backpressure to the source and cause the source to adapt to the overall throughput of the application by slowing down as well. You can find a deeper description of backpressure and how it works at [How Apache Flink™ handles backpressure](#).

Knowing which operators in an applications are slow gives you crucial information to understand the root cause of performance problems in the application. Backpressure information is [exposed through the Flink Dashboard](#). To identify the slow operator, look for the operator with a high backpressure value that is closest to a sink (operator B in the following example). The operator causing the slowness is then one of the downstream operators (operator C in the example). B could process events faster, but is backpressured as it cannot forward the output to the actual slow operator C.

```
A (backpressured 93%) -> B (backpressured 85%) -> C (backpressured 11%) -> D  
(backpressured 0%)
```

Once you have identified the slow operator, try to understand why it's slow. There could be a myriad of reasons and sometimes it's not obvious what's wrong and can require days of debugging and profiling to resolve. Following are some obvious and more common reasons, some of which are further explained below:

- The operator is doing slow I/O, e.g., network calls (consider using AsyncIO instead).
- There is a skew in the data and one operator is receiving more events than others (verify by looking at the number of messages in/out of individual subtasks (i.e., instances of the same operator) in the Flink dashboard).
- It's a resource intensive operation (if there is no data skew consider scaling out for CPU/memory bound work or increasing `ParallelismPerKPU` for I/O bound work)
- Extensive logging in the operator (reduce the logging to a minimum for production application or consider sending debug output to a data stream instead).

Testing throughput with the Discarding Sink

The [Discarding Sink](#) simply disregards all events it receives while still executing the application (an application without any sink fails to execute). This is very useful for throughput testing, profiling, and to verify if the application is scaling properly. It's also a very pragmatic sanity check to verify if the sinks are causing back pressure or the application (but just checking the backpressure metrics is often easier and more straightforward).

By replacing all sinks of an application with a discarding sink and creating a mock source that generates data that resembles production data, you can measure the maximum throughput of the application for a certain parallelism setting. You can then also increase the parallelism to verify that the application scales properly and does not have a bottleneck that only emerges at higher throughput (e.g., because of data skew).

Data skew

A Flink application is executed on a cluster in a distributed fashion. To scale out to multiple nodes, Flink uses the concept of keyed streams, which essentially means that the events of a stream are partitioned according to a specific key, e.g., customer id, and Flink can then process different partitions on different nodes. Many of the Flink operators are then evaluated based on these partitions, e.g., [Keyed Windows](#), [Process Functions](#) and [Async I/O](#).

Choosing a partition key often depends on the business logic. At the same time, many of the best practices for, e.g., [DynamoDB](#) and Spark, equally apply to Flink, including:

- ensuring a high cardinality of partition keys
- avoiding skew in the event volume between partitions

You can identify skew in the partitions by comparing the records received/sent of subtasks (i.e., instances of the same operator) in the Flink dashboard. In addition, Managed Service for Apache Flink monitoring can be configured to expose metrics for `numRecordsIn/Out` and `numRecordsInPerSecond/OutPerSecond` on a subtask level as well.

State skew

For stateful operators, i.e., operators that maintain state for their business logic such as windows, data skew always leads to state skew. Some subtasks receive more events than others because of the skew in the data and hence are also persisting more data in state. However, even for an

application that has evenly balanced partitions, there can be a skew in how much data is persisted in state. For instance, for session windows, some users and sessions respectively may be much longer than others. If the longer sessions happen to be part of the same partition, it can lead to an imbalance of the state size kept by different subtasks of the same operator.

State skew not only increases more memory and disk resources required by individual subtasks, it can also decrease the overall performance of the application. When an application is taking a checkpoint or savepoint, the operator state is persisted to Amazon S3, to protect the state against node or cluster failure. During this process (especially with exactly once semantics that are enabled by default on Managed Service for Apache Flink), the processing stalls from an external perspective until the checkpoint/savepoint has completed. If there is data skew, the time to complete the operation can be bound by a single subtask that has accumulated a particularly high amount of state. In extreme cases, taking checkpoints/savepoints can fail because of a single subtask not being able to persist state.

So similar to data skew, state skew can substantially slow down an application.

To identify state skew, you can leverage the Flink dashboard. Find a recent checkpoint or savepoint and compare the amount of data that has been stored for individual subtasks in the details.

Integrating with resources in different Regions

You can enable using `StreamingFileSink` to write to an Amazon S3 bucket in a different Region from your Managed Service for Apache Flink application via a setting required for cross Region replication in the Flink configuration. To do this, file a support ticket at [AWS Support Center](#).

Document history for Amazon Managed Service for Apache Flink

The following table describes the important changes to the documentation since the last release of Managed Service for Apache Flink.

- **API version: 2018-05-23**
- **Latest documentation update: August 30, 2023**

Change	Description	Date
Kinesis Data Analytics is now known as Managed Service for Apache Flink	There are no changes to the service endpoints, APIs, the Command Line Interface, IAM access policies, CloudWatch Metrics, or the AWS Billing dashboards. Your existing applications will continue to work as they did previously. For more information, see What Is Managed Service for Apache Flink?	August 30, 2023
Support for Apache Flink version 1.15.2	Managed Service for Apache Flink now supports applications that use Apache Flink version 1.15.2. Create Kinesis Data Analytics applications using the Apache Flink Table API. For more information, see Creating applications .	November 22, 2022
Support for Apache Flink version 1.13.2	Managed Service for Apache Flink now supports applications that use Apache Flink	October 13, 2021

Change	Description	Date
	version 1.13.2. Create Kinesis Data Analytics applications using the Apache Flink Table API. For more information, see Getting Started: Flink 1.13.2 .	
Support for Python	Managed Service for Apache Flink now supports applications that use Python with the Apache Flink Table API & SQL. For more information, see Using Python .	March 25, 2021
Support for Apache Flink 1.11.1	Managed Service for Apache Flink 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 Creating applications .	November 19, 2020
Apache Flink Dashboard	Use the Apache Flink Dashboard to monitor application health and performance. For more information, see Apache Flink Dashboard .	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 EFO Consumer .	October 6, 2020

Change	Description	Date
Apache Beam	Create applications that use Apache Beam to process streaming data. For more information, see Using CloudFormation with Managed Service for Apache Flink .	September 15, 2020
Performance	How to troubleshoot application performance issues, and how to create a performant application. For more information, see Performance .	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 Custom Truststore .	June 10, 2020
CloudWatch Alarms	Recommendations for creating CloudWatch alarms with Managed Service for Apache Flink. For more information, see Alarms .	June 5, 2020
New CloudWatch Metrics	Managed Service for Apache Flink now emits 22 metrics to Amazon CloudWatch Metrics. For more information, see Metrics and dimensions in Managed Service for Apache Flink .	May 12, 2020

Change	Description	Date
Custom CloudWatch Metrics	Define application-specific metrics and emit them to Amazon CloudWatch Metrics. For more information, see Custom metrics .	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 Managed Service for Apache Flink application. For more information, see Cross-Account .	March 30, 2020
Support for Apache Flink 1.8.2	Managed Service for Apache Flink now supports applications that use Apache Flink 1.8.2. Use the Flink Streaming FileSink connector to write output directly to S3. For more information, see Creating applications .	December 17, 2019
Managed Service for Apache Flink VPC	Configure a Managed Service for Apache Flink application to connect to a virtual private cloud. For more information, see Using an Amazon VPC .	November 25, 2019
Managed Service for Apache Flink Best Practices	Best practices for creating and administering Managed Service for Apache Flink applications. For more information, see Best practices .	October 14, 2019

Change	Description	Date
Analyze Managed Service for Apache Flink Application Logs	Use CloudWatch Logs Insights to monitor your Managed Service for Apache Flink application. For more information, see Analyzing logs .	June 26, 2019
Managed Service for Apache Flink Application Runtime Properties	Work with Runtime Properties in Managed Service for Apache Flink. For more information, see Runtime properties .	June 24, 2019
Tagging Managed Service for Apache Flink Applications	Use application tagging to determine per-application costs, control access, or for user-defined purposes. For more information, see Using tagging .	May 8, 2019
Logging Managed Service for Apache Flink API Calls with AWS CloudTrail	Managed Service for Apache Flink is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Managed Service for Apache Flink. For more information, see Using AWS CloudTrail .	March 22, 2019

Change	Description	Date
Create an Application (Firehose Sink)	Exercise to create a Managed Service for Apache Flink with an Amazon Kinesis data stream as a source, and an Amazon Data Firehose stream as a sink. For more information, see Firehose sink .	December 13, 2018
Public release	This is the initial release of the <i>Managed Service for Apache Flink Developer Guide for Java Applications</i> .	November 27, 2018

Managed Service for Apache Flink API example code

This topic contains example request blocks for Managed Service for Apache Flink 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](#)
- [AddApplicationInput](#)
- [AddApplicationInputProcessingConfiguration](#)
- [AddApplicationOutput](#)
- [AddApplicationReferenceDataSource](#)
- [AddApplicationVpcConfiguration](#)
- [CreateApplication](#)
- [CreateApplicationSnapshot](#)
- [DeleteApplication](#)
- [DeleteApplicationCloudWatchLoggingOption](#)
- [DeleteApplicationInputProcessingConfiguration](#)
- [DeleteApplicationOutput](#)
- [DeleteApplicationReferenceDataSource](#)
- [DeleteApplicationSnapshot](#)
- [DeleteApplicationVpcConfiguration](#)
- [DescribeApplication](#)
- [DescribeApplicationSnapshot](#)

- [DiscoverInputSchema](#)
- [ListApplications](#)
- [ListApplicationSnapshots](#)
- [StartApplication](#)
- [StopApplication](#)
- [UpdateApplication](#)

AddApplicationCloudWatchLoggingOption

The following example request code for the [AddApplicationCloudWatchLoggingOption](#) action adds an Amazon CloudWatch logging option to a Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application:

```
{
  "ApplicationName": "MyApplication",
```

```
"ApplicationDescription": "My-Application-Description",
"RuntimeEnvironment": "FLINK-1_15",
"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 Managed Service for Apache Flink application:

```
{"ApplicationName": "MyApplication",
 "CreateTimestamp": 12345678912}
```

DeleteApplicationCloudWatchLoggingOption

The following example request code for the [DeleteApplicationCloudWatchLoggingOption](#) action deletes an Amazon CloudWatch logging option from a Managed Service for Apache Flink 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 Managed Service for Apache Flink application:

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 4,
  "InputId": "2.1"
}
```

DeleteApplicationOutput

The following example request code for the [DeleteApplicationOutput](#) action removes an application output from a Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink 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 Managed Service for Apache Flink application:

```
{"ApplicationName": "MyApplication"}
```

UpdateApplication

The following example request code for the [UpdateApplication](#) action updates a Managed Service for Apache Flink 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"  
  }  
}  
}
```

Managed Service for Apache Flink API Reference

For information about the APIs that Managed Service for Apache Flink provides, see [Managed Service for Apache Flink API Reference](#).

This content was moved to Release versions. See [Release versions](#).