aws

Managed Service for Apache Flink Developer Guide

Managed Service for Apache Flink

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Copyright © 2026 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

..... xvii

What is Managed Service for Apache Flink? 1

Decide between using Managed Service for Apache Flink or Managed Service for Apache Flink Studio 1

Choose which Apache Flink APIs to use in Managed Service for Apache Flink 3

Choose a Flink API 3

Get started with streaming data applications 5

How it works 6

Program your Apache Flink application 6

DataStream API 6

Table API 7

Create your Managed Service for Apache Flink application 7

Create an application 8

Build your Managed Service for Apache Flink application code 8

Create your Managed Service for Apache Flink application 9

Use customer managed keys 10

Start your Managed Service for Apache Flink application 11

Verify your Managed Service for Apache Flink application 11

Enable system rollbacks 11

Run an application 14

Identify application and job status 14

Run batch workloads 16

Application resources 16

Managed Service for Apache Flink application resources 16

Apache Flink application resources 17

Pricing 18

How it works 16

AWS Region availability 19

Pricing examples 20

Review DataStream API components 24

Connectors 24

Operators 34

Event tracking 35

Table API components 36

Table API connectors ... 36
Table API time attributes ... 38
Use Python ... 38
Program your Python application ... 39
Create your Python application ... 42
Monitor your Python application ... 43
Use runtime properties ... 44
Manage runtime properties using the console ... 44
Manage runtime properties using the CLI ... 45
Access runtime properties in a Managed Service for Apache Flink application ... 48
Use Apache Flink connectors ... 49
Connectors for Flink 2.0+ ... 50
Connectors for older Flink versions ... 52
Implement fault tolerance ... 54
Configure checkpointing in Managed Service for Apache Flink ... 54
Review checkpointing API examples ... 55
Manage application backups using snapshots ... 58
Manage automatic snapshot creation ... 59
Restore from a snapshot that contains incompatible state data ... 59
Review snapshot API examples ... 60
Use in-place version upgrades for Apache Flink ... 63
Upgrade applications ... 64
Upgrade to a new version ... 65
Roll back application upgrades ... 70
Best practices ... 71
Known issues ... 71
Upgrading to Flink 2.2 ... 73
State compatibility guide ... 82
Implement application scaling ... 87
Configure application parallelism and ParallelismPerKPU ... 87
Allocate Kinesis Processing Units ... 88
Update your application's parallelism ... 89
Use automatic scaling ... 90
maxParallelism considerations ... 92
Add tags to applications ... 93
Add tags when an application is created ... 94

Add or update tags for an existing application ... 94
List tags for an application ... 95
Remove tags from an application ... 95
Use CloudFormation ... 96
Before you begin ... 96
Write a Lambda function ... 96
Create a Lambda role ... 98
Invoke the Lambda function ... 98
Review an extended example ... 99
Use the Apache Flink Dashboard ... 105
Access your application's Apache Flink Dashboard ... 105
Supported and deprecated versions ... 107
Amazon Managed Service for Apache Flink 2.3 ... 114
Supported features ... 115
Changes in Amazon Managed Service for Apache Flink with Apache Flink 2.3 ... 116
Components ... 117
Unsupported and experimental features ... 117
Amazon Managed Service for Apache Flink Studio ... 117
Amazon Managed Service for Apache Flink 2.2 ... 118
What's new in Amazon Managed Service for Apache Flink 2.2 ... 118
Breaking changes and deprecations ... 119
Apache Flink 2.2 features supported ... 120
Connector availability ... 123
Unsupported and experimental features ... 125
Known issues ... 125
Upgrade experience ... 126
Next steps ... 126
Amazon Managed Service for Apache Flink 1.20 ... 126
Supported features ... 127
Components ... 127
Known issues ... 128
Amazon Managed Service for Apache Flink 1.19 ... 129
Supported features ... 129
Changes in Amazon Managed Service for Apache Flink 1.19.1 ... 131
Components ... 133
Known issues ... 133

Amazon Managed Service for Apache Flink 1.18 134
Changes in Amazon Managed Service for Apache Flink with Apache Flink 1.15 135
Components 137
Known issues 137
Amazon Managed Service for Apache Flink 1.15 138
Changes in Amazon Managed Service for Apache Flink with Apache Flink 1.15 140
Components 137
Known issues 141
Earlier versions 141
Using the Apache Flink Kinesis Streams connector with previous Apache Flink versions 143
Building applications with Apache Flink 1.8.2 144
Building applications with Apache Flink 1.6.2 145
Upgrading applications 146
Available connectors in Apache Flink 1.6.2 and 1.8.2 146
Getting Started: Flink 1.13.2 146
Getting Started: Flink 1.11.1 170
Getting started: Flink 1.8.2 - deprecating 195
Getting started: Flink 1.6.2 - deprecating 220
Legacy examples 243
Use Studio notebooks with Managed Service for Apache Flink 415
Use the correct Studio notebook Runtime version 416
Create a Studio notebook 417
Perform an interactive analysis of streaming data 418
Flink interpreters 418
Apache Flink table environment variables 419
Deploy as an application with durable state 420
Scala/Python criteria 421
SQL criteria 422
IAM permissions 422
Use connectors and dependencies 423
Default connectors 423
Add dependencies and custom connectors 424
User-defined functions 425
Considerations with user-defined functions 426
Enable checkpointing 428
Set the checkpointing interval 428

Set the checkpointing type 428
Upgrade Studio Runtime 429
Upgrade your notebook to a new Studio Runtime 429
Work with AWS Glue 433
Table properties 433
Examples and tutorials for Studio notebooks in Managed Service for Apache Flink 435
Tutorial: Create a Studio notebook in Managed Service for Apache Flink 436
Tutorial: Deploy a Studio notebook as a Managed Service for Apache Flink application with durable state 455
View example queries to analyza data in a Studio notebook 458
Troubleshoot Studio notebooks for Managed Service for Apache Flink 471
Stop a stuck application 471
Deploy as an application with durable state in a VPC with no internet access 471
Deploy-as-app size and build time reduction 472
Cancel jobs 474
Restart the Apache Flink interpreter 475
Create custom IAM policies for Managed Service for Apache Flink Studio notebooks 475
AWS Glue 476
CloudWatch Logs 476
Kinesis streams 477
Amazon MSK clusters 480
Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 481
Review application components 196
Complete the required prerequisites 482
Set up an account 483
Sign up for an AWS account 148
Set up the AWS CLI 483
Next step 485
Create an application 485
Create dependent resources 486
Set up your local development environment 487
Download and examine the Apache Flink streaming Java code 488
Write sample records to the input stream 492
Run your application locally 494
Observe input and output data in Kinesis streams 497
Stop your application running locally 498

Compile and package your application code ... 498
Upload the application code JAR file ... 498
Create and configure the Managed Service for Apache Flink application ... 499
Next step ... 506
Clean up resources ... 506
Delete your Managed Service for Apache Flink application ... 507
Delete your Kinesis data streams ... 507
Delete your Amazon S3 objects and bucket ... 507
Delete your IAM resources ... 508
Delete your CloudWatch resources ... 508
Explore additional resources for Apache Flink ... 508
Explore additional resources ... 509
Tutorial: Get started using the TableAPI in Managed Service for Apache Flink ... 510
Review application components ... 510
Complete the required prerequisites ... 511
Create an application ... 512
Create dependent resources ... 512
Set up your local development environment ... 513
Download and examine the Apache Flink streaming Java code ... 514
Run your application locally ... 520
Observe the application writing data to an S3 bucket ... 522
Stop your application running locally ... 523
Compile and package your application code ... 523
Upload the application code JAR file ... 524
Create and configure the Managed Service for Apache Flink application ... 525
Next step ... 531
Clean up resources ... 531
Delete your Managed Service for Apache Flink application ... 531
Delete your Amazon S3 objects and bucket ... 532
Delete your IAM resources ... 532
Delete your CloudWatch resources ... 533
Next step ... 533
Explore additional resources ... 533
Tutorial: Get started using Python in Managed Service for Apache Flink ... 534
Review application components ... 534
Fulfill the prerequisites ... 535

Create an application ... 537
Create dependent resources ... 537
Set up your local development environment ... 539
Download and examine the Apache Flink streaming Python code ... 540
Manage JAR dependencies ... 543
Write sample records to the input stream ... 545
Run your application locally ... 547
Observe input and output data in Kinesis streams ... 549
Stop your application running locally ... 549
Package your application code ... 549
Upload the application package to an Amazon S3 bucket ... 550
Create and configure the Managed Service for Apache Flink application ... 550
Next step ... 557
Clean up resources ... 557
Delete your Managed Service for Apache Flink application ... 557
Delete your Kinesis data streams ... 558
Delete your Amazon S3 objects and bucket ... 558
Delete your IAM resources ... 559
Delete your CloudWatch resources ... 559
Tutorial: Get started using Scala in Managed Service for Apache Flink ... 560
Create dependent resources ... 560
Write sample records to the input stream ... 561
Download and examine the application code ... 563
Compile and upload the application code ... 564
Create and run the application (console) ... 565
Create the Application ... 565
Configure the application ... 566
Edit the IAM policy ... 568
Run the application ... 570
Stop the application ... 570
Create and run the application (CLI) ... 570
Create a permissions policy ... 570
Create an IAM policy ... 572
Create the application ... 573
Start the application ... 575
Stop the application ... 411

Add a CloudWatch logging option ... 411
Update environment properties ... 411
Update the application code ... 412
Clean up AWS resources ... 578
Delete your Managed Service for Apache Flink application ... 578
Delete your Kinesis data streams ... 578
Delete your Amazon S3 object and bucket ... 579
Delete your IAM resources ... 579
Delete your CloudWatch resources ... 579
Use Apache Beam with Managed Service for Apache Flink applications ... 580
Limitations of Apache Flink runner with Managed Service for Apache Flink ... 580
Apache Beam capabilities with Managed Service for Apache Flink ... 580
Creating an application using Apache Beam ... 581
Create dependent resources ... 581
Write sample records to the input stream ... 582
Download and examine the application code ... 583
Compile the application code ... 584
Upload the Apache Flink streaming Java code ... 585
Create and run the Managed Service for Apache Flink application ... 585
Clean Up ... 589
Next steps ... 591
Training workshops, labs, and solution implementations ... 592
Managed Service for Apache Flink workshop ... 592
Develop Apache Flink applications locally before deploying to Managed Service for Apache Flink ... 592
Event detection with Managed Service for Apache Flink Studio ... 593
AWS Streaming Data Solution ... 593
Practice using a Clickstream lab with Apache Flink and Apache Kafka ... 593
Set up custom scaling using Application Auto Scaling ... 594
View a sample Amazon CloudWatch dashboard ... 594
Use templates for AWS Streaming data solution for Amazon MSK ... 594
Explore more Managed Service for Apache Flink solutions on GitHub ... 594
Use practical utilities for Managed Service for Apache Flink ... 596
Snapshot manager ... 596
Benchmarking ... 596
Examples for creating and working with Managed Service for Apache Flink applications ... 597

Java examples for Managed Service for Apache Flink 597
Python examples for Managed Service for Apache Flink 601
Scala examples for Managed Service for Apache Flink 602
Security in Managed Service for Apache Flink 604
Data protection 605
Data encryption 605
Key management in Amazon MSF 606
Transparent encryption in Amazon MSF 606
Customer managed keys in Amazon MSF 606
Using customer managed keys 613
Managing CMK using console 616
Managing CMK using APIs 617
Identity and Access Management for Managed Service for Apache Flink 627
Audience 627
Authenticating with identities 627
Managing access using policies 629
How Amazon Managed Service for Apache Flink works with IAM 630
Identity-based policy examples 640
Troubleshooting 643
Cross-service confused deputy prevention 645
Compliance validation 646
FedRAMP 647
Resilience and disaster recovery in Managed Service for Apache Flink 648
Disaster recovery 648
Versioning 648
Infrastructure security in Managed Service for Apache Flink 649
Security best practices for Managed Service for Apache Flink 649
Implement least privilege access 649
Use IAM roles to access other Amazon services 650
Implement server-side encryption in dependent resources 650
Use CloudTrail to monitor API calls 650
Logging and monitoring in Amazon Managed Service for Apache Flink 651
Logging in Managed Service for Apache Flink 652
Querying Logs with CloudWatch Logs Insights 652
Monitoring in Managed Service for Apache Flink 652
Set up application logging in Managed Service for Apache Flink 654

Set up CloudWatch logging using the console ... 654
Set up CloudWatch logging using the CLI ... 655
Control application monitoring levels ... 660
Apply logging best practices ... 661
Perform logging troubleshooting ... 661
Use CloudWatch Logs Insights ... 662
Analyze logs with CloudWatch Logs Insights ... 662
Run a sample query ... 662
Review example queries ... 663
Metrics and dimensions in Managed Service for Apache Flink ... 666
Application metrics ... 666
Kinesis Data Streams connector metrics ... 698
Amazon MSK connector metrics ... 699
Apache Zeppelin metrics ... 701
View CloudWatch metrics ... 702
Set CloudWatch metrics reporting levels ... 702
Use custom metrics with Amazon Managed Service for Apache Flink ... 704
Use CloudWatch Alarms with Amazon Managed Service for Apache Flink ... 708
Write custom messages to CloudWatch Logs ... 720
Write to CloudWatch logs using Log4J ... 720
Write to CloudWatch logs using SLF4J ... 721
Log Managed Service for Apache Flink API calls with AWS CloudTrail ... 722
Managed Service for Apache Flink information in CloudTrail ... 723
Understand Managed Service for Apache Flink log file entries ... 724
Tune performance ... 726
Troubleshoot performance issues ... 726
Understand the data path ... 726
Performance troubleshooting solutions ... 727
Use performance best practices ... 729
Manage scaling properly ... 729
Monitor external dependency resource usage ... 731
Run your Apache Flink application locally ... 731
Monitor performance ... 732
Monitor performance using CloudWatch metrics ... 732
Monitor performance using CloudWatch logs and alarms ... 732
Managed Service for Apache Flink and Studio notebook quota ... 733

Manage maintenance tasks for Managed Service for Apache Flink 735
Choose a maintenance window 737
Identify maintenance instances 737
Achieve production readiness for your Managed Service for Apache Flink applications 739
Load-test your applications 739
Define Max parallelism 739
Set a UUID for all operators 740
Best practices 741
Minimize the size of the uber JAR 741
Fault tolerance: checkpoints and savepoints 744
Unsupported connector versions 744
Performance and parallelism 745
Setting per-operator parallelism 745
Logging 746
Coding 746
Managing credentials 747
Reading from sources with few shards/partitions 747
Studio notebook refresh interval 748
Studio notebook optimum performance 748
How watermark strategies and idle shards affect time windows 748
Summary 750
Example 750
Set a UUID for all operators 759
Add ServiceResourceTransformer to the Maven shade plugin 760
Apache Flink stateful functions 761
Apache Flink application template 761
Location of the module configuration 762
Learn about Apache Flink settings 763
Apache Flink configuration 763
State backend 764
Checkpointing 764
Savepointing 765
Heap sizes 766
Buffer debloating 766
Modifiable Flink configuration properties 766
Restart strategy 766

Checkpoints and state backends ... 767
Checkpointing ... 767
RocksDB native metrics ... 767
RocksDB options ... 768
Advanced state backends options ... 768
Full TaskManager options ... 769
Memory configuration ... 769
RPC / Akka ... 770
Client ... 770
Advanced cluster options ... 770
Filesystem configurations ... 770
Advanced fault tolerance options ... 770
Memory configuration ... 769
Metrics ... 771
Advanced options for the REST endpoint and client ... 771
Advanced SSL security options ... 771
Advanced scheduling options ... 771
Advanced options for Flink web UI ... 771
Programmatic Flink configuration properties ... 771
Pipeline configuration ... 771
Network ... 772
Python API ... 773
Table API / SQL ... 773
View configured Flink properties ... 777
Configure MSF to access resources in an Amazon VPC ... 778
Amazon VPC concepts ... 778
VPC application permissions ... 779
Add a permissions policy for accessing an Amazon VPC ... 779
Establish internet and service access for a VPC-connected Managed Service for Apache Flink application ... 780
Related information ... 782
Use the Managed Service for Apache Flink VPC API ... 782
Create application ... 782
AddApplicationVpcConfiguration ... 783
DeleteApplicationVpcConfiguration ... 783
Update application ... 784

Example: Use a VPC ... 784
Troubleshoot Managed Service for Apache Flink ... 785
Development troubleshooting ... 785
System rollback best practices ... 786
Hudi configuration best practices ... 787
Apache Flink Flame Graphs ... 787
Credential provider issue with EFO connector 1.15.2 ... 787
Applications with unsupported Kinesis connectors ... 788
Compile error: "Could not resolve dependencies for project" ... 790
Invalid choice: "kinesisanalyticsv2" ... 791
UpdateApplication action isn't reloading application code ... 791
S3 StreamingFileSink FileNotFoundExceptions ... 791
FlinkKafkaConsumer issue with stop with savepoint ... 793
Flink 1.15 Async Sink Deadlock ... 794
Amazon Kinesis data streams source processing out of order during re-sharding ... 803
Real-time vector embedding blueprints FAQ and troubleshooting ... 804
Runtime troubleshooting ... 816
Troubleshooting tools ... 816
Application issues ... 816
Application is restarting ... 821
Throughput is too slow ... 824
Unbounded state growth ... 825
I/O bound operators ... 826
Upstream or source throttling from a Kinesis data stream ... 826
Checkpoints ... 827
Checkpointing is timing out ... 833
Checkpoint failure for Apache Beam ... 835
Backpressure ... 837
Data skew ... 838
State skew ... 838
Integrate with resources in different Regions ... 839
Document history ... 840
API example code ... 846
AddApplicationCloudWatchLoggingOption ... 847
AddApplicationInput ... 847
AddApplicationInputProcessingConfiguration ... 848

AddApplicationOutput 849
AddApplicationReferenceDataSource 849
AddApplicationVpcConfiguration 850
CreateApplication 850
CreateApplicationSnapshot 852
DeleteApplication 852
DeleteApplicationCloudWatchLoggingOption 852
DeleteApplicationInputProcessingConfiguration 852
DeleteApplicationOutput 853
DeleteApplicationReferenceDataSource 853
DeleteApplicationSnapshot 853
DeleteApplicationVpcConfiguration 854
DescribeApplication 854
DescribeApplicationSnapshot 854
DiscoverInputSchema 854
ListApplications 855
ListApplicationSnapshots 855
StartApplication 856
StopApplication 856
UpdateApplication 856
API Reference 858
Release versions 859

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.

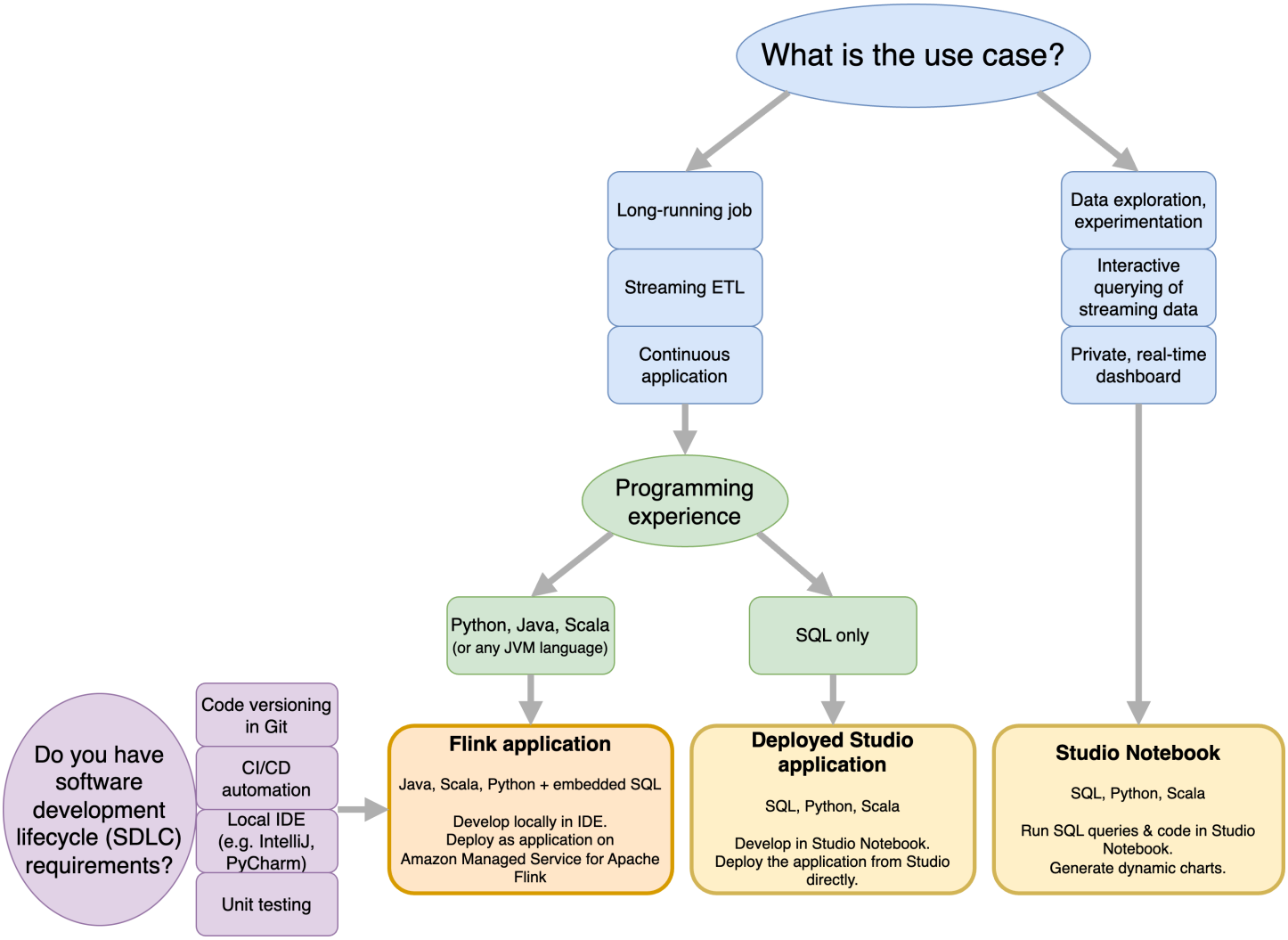
You can build applications with the language of your choice in Managed Service for Apache Flink using open-source libraries based on Apache Flink. Apache Flink is a popular framework and engine for processing data streams.

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.

Decide between using 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.

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

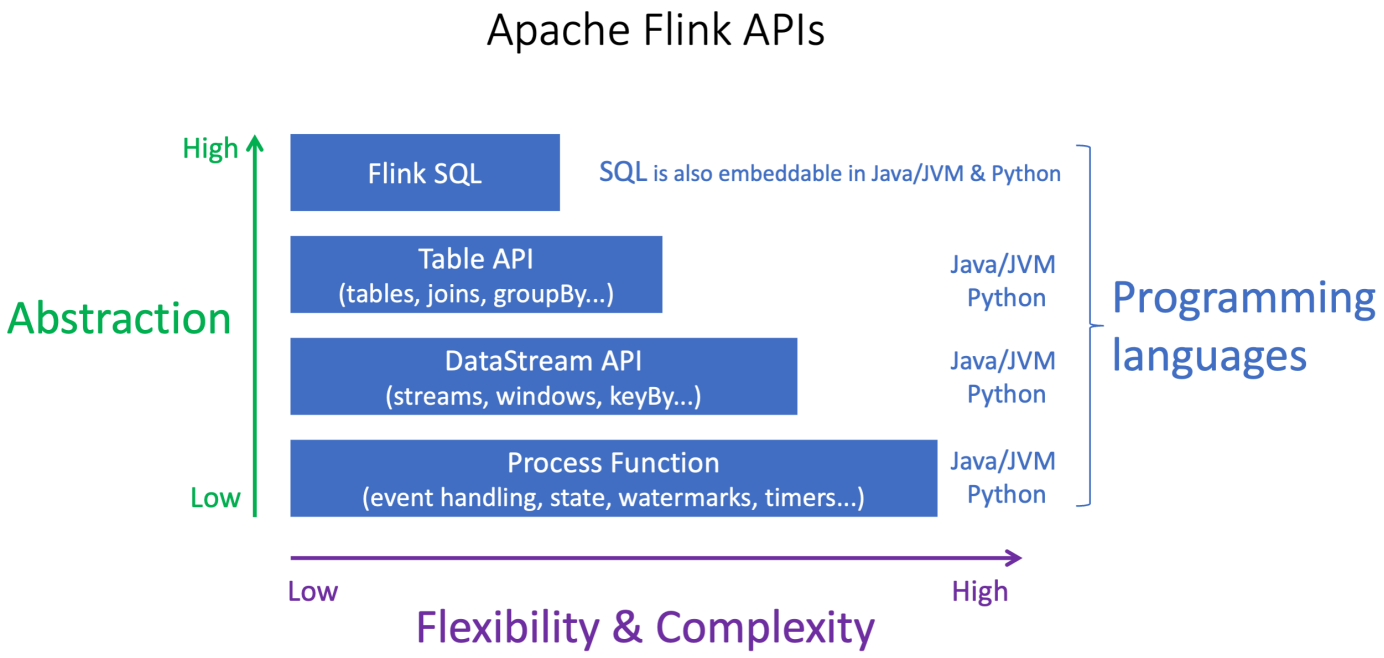
You can build applications using Java, Python, and Scala in Managed Service for Apache Flink using Apache Flink APIs in an IDE of your choice. You can find guidance on how to build applications using the Flink Datastream and Table API in the documentation. You can select the language you create your 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.

Choose 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



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 UUIDs For All Operators in the Apache Flink Documentation.

Get 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
- Get started with Amazon Managed Service for Apache Flink (DataStream API)

Altenatively, 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:

- Use a Studio notebook with Managed Service for Apache Flink
- Create a Studio notebook

Managed Service for Apache Flink: How it works

Managed Service for Apache Flink is a fully managed Amazon service that lets you use an Apache Flink application to process streaming data. First, you program your Apache Flink application, and then you create your Managed Service for Apache Flink application.

Program 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 Review DataStream API components.

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 Review Table API components.

Create 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::

- **Use runtime properties:** Parameters that you can provide to your application. You can change these parameters without recompiling your application code.
- **Implement fault tolerance**: How your application recovers from interrupts and restarts.
- **Logging and monitoring in Amazon Managed Service for Apache Flink**: How your application logs events to CloudWatch Logs.
- **Implement application 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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink.

Create a Managed Service for Apache Flink application

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

This topic contains the following sections:

- Build your Managed Service for Apache Flink application code
- Create your Managed Service for Apache Flink application
- Use customer managed keys
- Start your Managed Service for Apache Flink application
- Verify your Managed Service for Apache Flink application
- Enable system rollbacks for your Managed Service for Apache Flink application

Build 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 Use in-place version upgrades for Apache Flink.

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 Use Apache Beam with Managed Service for Apache Flink applications.

Specify 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 2.3.0, provide the following:

```
mvn package -Dflink.version=2.3.0
```

For building applications with earlier versions of Apache Flink, see Earlier versions.

Create your Managed Service for Apache Flink application

After you've built your application code, you do the following to create your Managed Service for Apache Flink (Amazon MSF) 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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink tutorial.
- **Create your Managed Service for Apache Flink application**: Use one of the following methods to create your Amazon MSF application:

  > **Note**
  >
  > Amazon MSF encrypts your application by default using AWS owned keys. You can also create your new application using AWS KMS customer managed keys (CMKs) to create,

own, and manage your keys yourself. For information about CMKs, see Key management in Amazon Managed Service for Apache Flink.

- **Create your Amazon MSF 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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink tutorial.
- **Create your Amazon MSF 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.

> **Note**
>
> You can change the `RuntimeEnvironment` of an existing application. To learn how, see Use in-place version upgrades for Apache Flink.

Use customer managed keys

In Amazon MSF, customer managed keys (CMKs) is a feature using which you can encrypt your application's data with a key that you create, own, and manage on AWS Key Management Service (AWS KMS). For an Amazon MSF application, this means all data subject to a Flink checkpoint or snapshot are encrypted with a CMK you define for that application.

To use CMK with your application, you must first create your new application, and then apply a CMK. For more information about using CMKs, see Key management in Amazon Managed Service for Apache Flink.

Start 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.

Verify 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 in Amazon Managed Service for Apache Flink.
- **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.

Enable 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"
       }
    }
}
```

Review common scenarios for automatic system rollback

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.

Use operation APIs for system rollbacks

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.

Run 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.

Identify 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 Use automatic scaling in Managed Service for Apache Flink.
  - `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 *Manage maintenance tasks for Managed Service for Apache Flink*.

  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.

Run 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.

Review Managed Service for Apache Flink 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 (KPUs)**.

One KPU represents the following system resources:

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

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

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

For more information about application parallelism, see Implement application 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 KPU. The number of slots assigned to a single KPU is equal to your application's `ParallelismPerKPU` setting. For more information about task slots, see Job Scheduling in the Apache Flink Documentation.

Operator parallelism

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

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

For more information about 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.

Per second billing in Managed Service for Apache Flink

Managed Service for Apache Flink is now billed in one-second increments. There is a ten-minute minimum charge per application. Per-second billing is applicable to applications that are newly launched or already running. This section describes how Managed Service for Apache Flink meters and bills you for your usage. To learn more about Managed Service for Apache Flink pricing, see Amazon Managed Service for Apache Flink Pricing.

How it works

Managed Service for Apache Flink charges you for the duration and number of **Kinesis Processing Units (KPUs)** that are billed in one-second increments in the supported AWS Regions. A single KPU comprises 1vCPU compute and 4 GB of memory. You are charged an hourly rate based on the number of KPUs used to run your applications.

For example, an application running for 20 minutes and 10 seconds will be charged for 20 minutes and 10 seconds, multiplied by the resources it used. An application that is running for 5 minutes will be charged the ten-minute minimum, multiplied by the resources it used.

Managed Service for Apache Flink states usage in hours. For example, 15 minutes corresponds to 0.25 hours.

For Apache Flink applications, you are charged a single additional KPU per application, used for orchestration. Applications are also charged for running storage and durable backups. Running application storage is used for stateful processing capabilities in Managed Service for Apache Flink and is charged per GB/month. Durable backups are optional and provide point-in-time recovery for applications, charged per GB/month.

In streaming mode, Managed Service for Apache Flink automatically scales the number of KPUs required by your stream processing application as the demands of memory and compute fluctuate. You can choose to provision your application with the required number of KPUs.

AWS Region availability

> **Note**
>
> At this time, per second billing is not available in the following Regions: AWS GovCloud (US-East), AWS GovCloud (US-West), China (Beijing), and China (Ningxia).

Per second billing is available in the following AWS Regions:

- US East (N. Virginia) - us-east-1
- US East (Ohio) - us-east-2
- US West (N. California) - us-west-1
- US West (Oregon) - us-west-2
- Africa (Cape Town) - af-south-1
- Asia Pacific (Hong Kong) - ap-east-1
- Asia Pacific (Hyderabad) - ap-south-1
- Asia Pacific (Jakarta) - ap-southeast-3
- Asia Pacific (Melbourne) - ap-southeast-4
- Asia Pacific (Mumbai) - ap-south-1
- Asia Pacific (Osaka) - ap-northeast-3
- Asia Pacific (Seoul) - ap-northeast-2
- Asia Pacific (Singapore) - ap-southeast-1
- Asia Pacific (Sydney) - ap-southeast-2
- Asia Pacific (Tokyo) - ap-northeast-1
- Canada (Central) - ca-central-1
- Canada West (Calgary) - ca-west-1
- Europe (Frankfurt) - eu-central-1
- Europe (Ireland) - eu-west-1
- Europe (London) - eu-west-2
- Europe (Milan) - eu-south-1
- Europe (Paris) - eu-west-3

- Europe (Spain) - eu-south-2
- Europe (Stockholm) - eu-north-1
- Europe (Zurich) - eu-central-2
- Israel (Tel Aviv) - il-central-1
- Middle East (Bahrain) - me-south-1
- Middle East (UAE) - me-central-1
- South America (São Paulo) - sa-east-1

Pricing examples

You can find pricing examples on the Managed Service for Apache Flink pricing page. For more information, see Amazon Managed Service for Apache Flink Pricing. Following are further examples with Cost Usage Report illustrations for each.

A long running, heavy workload

You are a large Video streaming service and you would like to build a real-time video recommendation based on your users' interactions. You use an Apache Flink application in Managed Service for Apache Flink to continuously ingest user interaction events from multiple Kinesis data streams and to process events in real time before outputting to a downstream system. User interaction events are transformed using several operators. This includes partitioning data by event type, enriching data with additional metadata, sorting data by timestamp, and buffering data for 5 minutes before delivery. The application has many transformation steps that are compute-intensive and parallelizable. Your Flink application is configured to run with 20 KPUs to accommodate the workload. Your application uses 1 GB of durable application backup every day. The monthly Managed Service for Apache Flink charges will be computed as follows:

Monthly charges

The price in the US East (N. Virginia) Region is $0.11 per KPU-hour. Managed Service for Apache Flink allocates 50 GB of running application storage per KPU and charges $0.10 per GB/month.

- Monthly KPU charges: 24 hours * 30 days * (20 KPUs + 1 additional KPU for streaming application) * $0.11/hour = $1,584.00
- Monthly running application storage charges: 30 days * 20 KPUs * 50 GB/KPUs * $0.10/GB-month = $100.00

- Monthly durable application storage charges: 30 days * 1 GB * 0.023/GB-month = $0.03
- Total charges: $1,584.00 + $100 + $0.03 = **$1,684.03**

Cost usage report for Managed Service for Apache Flink on the Billing and Cost Management console for the month

Kinesis Analytics

- USD 1,684.03 - US East (N. Virginia)
- Amazon Kinesis Analytics CreateSnapshot
  - $0.023 per GB-month of durable application backups
    - 1 GB-month - USD 0.03
- Amazon Kinesis Analytics StartApplication
  - $0.10 per GB-month of running application storage
    - 1,000 GB-month - USD 100
  - $0.11 per Kinesis Processing Unit-hour for Apache Flink applications
    - 15,120 KPU-hour - USD 1,584

A batch workload that runs for ~15 minutes every day

You use an Apache Flink application in Managed Service for Apache Flink to transform log data in Amazon Simple Storage Service (Amazon S3) in batch mode. The log data is transformed using several operators. This includes applying a schema to the different log events, partitioning data by event type, and sorting data by timestamp. The application has many transformation steps, but none are computationally intensive. This application ingests data at 2,000 records/second for 15 minutes every day in a 30-day month. You do not create any durable application backups. The monthly Managed Service for Apache Flink charges will be computed as follows:

Monthly charges

The price in the US East (N. Virginia) Region is $0.11 per KPU-hour. Managed Service for Apache Flink allocates 50 GB of running application storage per KPU and charges $0.10 per GB/month.

- Batch Workload: During the 15 minutes per day, the Managed Service for Apache Flink application is processing 2,000 records/second, which takes 2KPUs. 30 days/month * 15 minutes/day = 450 minutes/month

- Monthly KPU charges: 450 minutes/month * (2KPUs + 1 additional KPU for streaming application) * $0.11/hour = $2.48
- Monthly running application storage charges: 450 minutes/month * 2 KPUs * 50 GB/KPUs * $0.10/GB-month = $0.11
- Total charges: $2.48 + 0.11 = **$2.59**

Cost usage report for Managed Service for Apache Flink on the Billing and Cost Management console for the month

Kinesis Analytics

- USD 2.59 - US East (N. Virginia)
- Amazon Kinesis Analytics StartApplication
  - $0.10 per GB-month of running application backups
    - 1.042 GB-month - USD 0.11
  - $0.11 per Kinesis Processing Unit-hour for Apache Flink applications
    - 22.5 KPU-Hour - USD 2.48

A test application that stops and starts continuously in the same hour, attracting multiple minimum charges

You're a large ecommerce platform that processes millions of transactions every day. You want to develop real-time fraud detection. You use an Apache Flink application in Managed Service for Apache Flink to ingest transaction events from Kinesis Data Streams and process events in real-time with different transformation steps. This includes using a sliding window to aggregate events, partitioning events by event types, and applying specific detection rules for different event types. During development, you start and stop your application multiple times to test and debug behavior. There are occasions when your application only runs for a few minutes. There is an hour when you're testing your application with 4 KPUs and your application does not use any durable application backups:

- At 10:05 AM, you start your application, which runs for 30 minutes before it's stopped at 10:35 AM.
- At 10:40 AM, you start your application again, which runs for 5 minutes before it's stopped at 10:45 AM.

- At 10:50 AM, you start the application again, which runs for 2 minutes before it's stopped at 10:52 AM.

Managed Service for Apache Flink charges a minimum of 10 minutes of usage each time an application starts running. The monthly Managed Service for Apache Flink usage for your application will be computed as follows:

- First time your application starts and stops: 30 minutes of usage
- Second time your application starts and stops: 10 minutes of usage (your application runs for 5 minutes rounded up to the 10 minutes minimum charge)
- Third time your application starts and stops: 10 minutes of usage (your application runs for 2 minutes, rounded up to the 10 minutes minimum charge)

In total, your application would be charged for 50 minutes of usage. If there are no other times in the month your application is running, the monthly Managed Service for Apache Flink charges will be computed as follows:

Monthly charges

The price in the US East (N. Virginia) Region is $0.11 per KPU-hour. Managed Service for Apache Flink allocates 50 GB of running application storage per KPU and charges $0.10 per GB/month.

- Monthly KPU charges: 50 minutes * (4KPUs + 1 additional KPU for streaming application) * $0.11/hour = $0.46 (rounded to the nearest penny)
- Monthly running application storage charges: 50 minutes * 4 KPUs * 50 GB/KPUs * $0.10/GB-month = $0.03 (rounded to the nearest penny)
- Total charges: $0.46 + 0.03 = **$0.49**

Cost usage report for Managed Service for Apache Flink on the Billing and Cost Management console for the month

Kinesis Analytics

- USD 0.49 - US East (N. Virginia)
- Amazon Kinesis Analytics StartApplication
  - $0.10 per GB-month of running application storage

  - 0.232 GB-month - USD 0.03
- $0.11 per Kinesis Processing Unit-hour for Apache Flink applications
  - 4.167 KPU-Hour - USD 0.46

Review DataStream API components

Your Apache Flink application uses the Apache Flink DataStream API to transform data in a data stream.

This section describes the different components that move, transform, and track data:

- Use 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.
- Transform data using operators in Managed Service for Apache Flink with the DataStream API: These components transform or group data elements within your application.
- Track 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.

Use 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 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:

- Add streaming data sources: Provide data to your application from a Kinesis data stream, file, or other data source.
- Write data using sinks: Send data from your application to a Kinesis data stream, Firehose stream, or other data destination.
- Use 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 might have 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 update your dependencies. To access the repository for Apache Flink AWS connectors, see flink-connector-aws.
The former Kinesis source `org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer` is discontinued and might be removed with a future release of Flink. Use Kinesis Source instead.
There is no state compatibility between the `FlinkKinesisConsumer` and `KinesisStreamsSource`. For details, see Migrating existing jobs to new Kinesis Streams Source in the Apache Flink documentation.
Following are the recommended guidelines:

Connector upgrades

Flink version	Connector used	Resolution
1.19, 1.20	Kinesis Source	When upgrading to Managed Service for Apache Flink version 1.19 and 1.20, make sure that you are using the most recent Kinesis Data Streams source connector . That must be any version 5.0.0 or later. For more information, see Amazon

Flink version	Connector used	Resolution
		Kinesis Data Streams Connector.
1.19, 1.20	Kinesis Sink	When upgrading to Managed Service for Apache Flink version 1.19 and 1.20, make sure that you are using the most recent Kinesis Data Streams sink connector. That must be any version 5.0.0 or later. For more information, see Kinesis Streams Sink.
1.19, 1.20	DynamoDB Streams Source	When upgrading to Managed Service for Apache Flink version 1.19 and 1.20, make sure that you are using the most recent DynamoDB Streams source connector. That must be any version 5.0.0 or later. For more informati on, see Amazon DynamoDB Connector.

Flink version	Connector used	Resolution
1.19, 1.20	DynamoDB Sink	When upgrading to Managed Service for Apache Flink version 1.19 and 1.20, make sure that you are using the most recent DynamoDB sink connector. That must be any version 5.0.0 or later. For more information, see Amazon DynamoDB Connector.
1.19, 1.20	Amazon SQS Sink	When upgrading to Managed Service for Apache Flink version 1.19 and 1.20, make sure that you are using the most recent Amazon SQS sink connector. That must be any version 5.0.0 or later. For more information, see Amazon SQS Sink.
1.19, 1.20	Amazon Managed Service for Prometheus Sink	When upgrading to Managed Service for Apache Flink version 1.19 and 1.20, make sure that you are using the most recent Amazon Managed Service for Prometheus sink connector. That must be any version 1.0.0 or later. For more informati on, see Prometheus Sink.

Add 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.

Use Kinesis data streams

The `KinesisStreamsSource` provides streaming data to your application from an Amazon Kinesis data stream.

Create a `KinesisStreamsSource`

The following code example demonstrates creating a `KinesisStreamsSource`:

```
// Configure the KinesisStreamsSource
Configuration sourceConfig = new Configuration();
sourceConfig.set(KinesisSourceConfigOptions.STREAM_INITIAL_POSITION,
 KinesisSourceConfigOptions.InitialPosition.TRIM_HORIZON); // This is optional, by
 default connector will read from LATEST

// Create a new KinesisStreamsSource to read from specified Kinesis Stream.
KinesisStreamsSource<String> kdsSource =
        KinesisStreamsSource.<String>builder()
                .setStreamArn("arn:aws:kinesis:us-east-1:123456789012:stream/test-
stream")
                .setSourceConfig(sourceConfig)
                .setDeserializationSchema(new SimpleStringSchema())

 .setKinesisShardAssigner(ShardAssignerFactory.uniformShardAssigner()) // This is
 optional, by default uniformShardAssigner will be used.
                .build();
```

For more information about using a `KinesisStreamsSource`, see Amazon Kinesis Data Streams Connector in the Apache Flink documentation and our public KinesisConnectors example on Github.

Create a `KinesisStreamsSource` that uses an EFO consumer

The `KinesisStreamsSource` 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:

- **READER_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 `KinesisStreamsSource` to use EFO, add the following parameters to the consumer:

```
sourceConfig.set(KinesisSourceConfigOptions.READER_TYPE,
 KinesisSourceConfigOptions.ReaderType.EFO);
sourceConfig.set(KinesisSourceConfigOptions.EFO_CONSUMER_NAME, "my-flink-efo-
consumer");
```

For an example of a Managed Service for Apache Flink application that uses an EFO consumer, see our public Kinesis Connectors example on Github.

Use Amazon MSK

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

Create 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.

Write 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:

Use 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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink.

Use 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.

Use 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”.

Use 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

- Create a FlinkKinesisFirehoseProducer
- FlinkKinesisFirehoseProducer Code Example

Create 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”.

Use 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.

Transform 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:

- Use transform operators
- Use aggregation operators

Use 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");
        }
    }
);
```

Use 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 for creating and working with Managed Service for Apache Flink applications.

Track 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.

Review Table API components

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.

Use 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 2.2 includes support for creating applications using Python version 3.12; support for Python version 3.8 is removed. 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.

Program your Managed Service for Apache Flink 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 Get started with Amazon Managed Service for Apache Flink for Python.

Read and write streaming data

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

Create 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)
```

Read 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))
```

Write 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))
```

Read 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]
```

Create 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.

Create your Managed Service for Apache Flink Python application

Specify 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"
          }
        },
```

Monitor your Managed Service for Apache Flink Python 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.

Query 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
```

Use 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:

- Manage runtime properties using the console
- Manage runtime properties using the CLI
- Access runtime properties in a Managed Service for Apache Flink application

Manage runtime properties using 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 Use in-place version upgrades for Apache Flink.

Update Runtime Properties for a Managed Service for Apache Flink application

1. Sign in to the AWS Management Console, and open the Amazon MSF 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**.

Manage runtime properties using 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.

Add 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"
               }
            }
         ]
      }
    }
}
```

Add and update 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.

Remove 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.

Access 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 Examples for creating and working with Managed Service for Apache Flink applications.

Use Apache Flink connectors with Managed Service for Apache Flink

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 2.0+

When upgrading to Flink 2.x, you need to update your connector dependencies to versions that are compatible with the Flink 2.x runtime. Flink connectors are released independently from the Flink runtime, and not all connectors have a Flink 2.x-compatible release yet. The following table summarizes the availability of commonly used connectors in Amazon Managed Service for Apache Flink as of this writing:

Connectors for Flink 2.0+

Connector	Flink 2.2 Version	Flink 2.3 Version
Apache Kafka	flink-connector-kafka 4.0.0-2.0	flink-connector-kafka 4.0.0-2.0
Kinesis Data Streams (source)	flink-connector-aws-kinesis-streams 6.0.1-2.0	flink-connector-aws-kinesis-streams 6.0.1-2.0
Kinesis Data Streams (sink)	flink-connector-aws-kinesis-streams 6.0.1-2.0	flink-connector-aws-kinesis-streams 6.0.1-2.0
FileSystem (S3, HDFS)	Bundled with Flink	Bundled with Flink
JDBC	flink-connector-jdbc-core 4.0.0-2.0	flink-connector-jdbc-core 4.0.0-2.0
OpenSearch	Not yet released for 2.x	Not yet released for 2.x
Elasticsearch	flink-connector-elasticsearch7 4.0.0-2.0	flink-connector-elasticsearch7 4.0.0-2.0

Connector	Flink 2.2 Version	Flink 2.3 Version
Amazon Managed Service for Prometheus	Not yet released for 2.x	Not yet released for 2.x

If your application depends on a connector that does not yet have a Flink 2.x release, you have two options: wait for the connector to release a compatible version, or evaluate whether you can replace it with an alternative (for example, using the JDBC catalog or a custom sink).

> **Note**
>
> Starting with version 4.0.0-2.0, the JDBC connector is modularized. The monolithic `flink-connector-jdbc` artifact is replaced by a core artifact (`flink-connector-jdbc-core`) and database-specific artifacts such as `flink-connector-jdbc-postgres` and `flink-connector-jdbc-mysql`. Update your Maven or Gradle dependencies accordingly when upgrading to Flink 2.x.

Known issues

- Applications using the `KinesisStreamsSource` with EFO (Enhanced Fan-Out / SubscribeToShard) path introduced in connector v5.0.0 and v6.0.0 may fail when Kinesis streams undergo resharding. This is a known issue in the community. For more information, see FLINK-37648.
- Applications using the `KinesisStreamsSource` with EFO (Enhanced Fan-Out / SubscribeToShard) path introduced in connector v5.0.0 and v6.0.0 together with `KinesisStreamsSink` may experience deadlocks if the Flink application is under backpressure, resulting in a complete stop of data processing in one or more TaskManagers. A force stop operation and a start app operation are needed to recover the app. This is a sub-case of the known issue in the community: FLINK-34071.

Connectors for older Flink versions

Connectors for older Flink versions

Connector	Flink version 1.15	Flink version 1.18	Flink versions 1.19	Flink versions 1.20
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, 5.0.0-1.19	flink-connector-kinesis, 5.0.0-1.20
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, 5.0.0-1.19	flink-connector-aws-kinesis-streams, 5.0.0-1.20
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, 5.0.0-1.19	flink-sql-connector-kinesis-streams, 5.0.0-1.20
Kafka - DataStream and Table API	flink-connector-kafka, 1.15.4	flink-connector-kafka, 3.2.0-1.18	flink-connector-kafka, 3.3.0-1.19	flink-connector-kafka, 3.3.0-1.20
Kafka - SQL	flink-sql-connector-kafka, 1.15.4	flink-sql-connector-kafka, 3.2.0-1.18	flink-sql-connector-kafka, 3.3.0-1.19	flink-sql-connector-kafka, 3.3.0-1.20
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, 5.0.0-1.19	flink-connector-aws-firehose, 5.0.0-1.20
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, 5.0.0-1.19	flink-sql-connector-aws-firehose, 5.0.0-1.20

Connector	Flink version 1.15	Flink version 1.18	Flink versions 1.19	Flink versions 1.20
DynamoDB - DataStream and Table API	flink-connector-dynamodb, 3.0.0-1.15	flink-connector-dynamodb, 4.3.0-1.18	flink-connector-dynamodb, 5.0.0-1.19	flink-connector-dynamodb, 5.0.0-1.20
DynamoDB - SQL	flink-sql-connector-dynamodb, 3.0.0-1.15	flink-sql-connector-dynamodb, 4.3.0-1.18	flink-sql-connector-dynamodb, 5.0.0-1.19	flink-sql-connector-dynamodb, 5.0.0-1.20
OpenSearch - DataStream and Table API	-	flink-connector-opensearch, 1.2.0-1.18	flink-connector-opensearch, 1.2.0-1.19	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	flink-sql-connector-opensearch, 1.2.0-1.19
Amazon Managed Service for Prometheus DataStream	-	flink-sql-connector-opensearch, 1.2.0-1.18	flink-connector-prometheus, 1.0.0-1.19	flink-connector-prometheus, 1.0.0-1.20
Amazon SQS DataStream and Table API	-	flink-sql-connector-opensearch, 1.2.0-1.18	flink-connector-sqs, 5.0.0-1.19	flink-connector-sqs, 5.0.0-1.20

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

- DataStream API connectors
- Table API connectors

Known issues

There is a known open source Apache Flink issue with the Apache Kafka connector in Apache Flink 1.15. This issue is resolved in later versions of Apache Flink.

For more information, see the section called "Known issues".

Implement 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 Manage 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.

Configure 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`.

Review 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:::amzn-s3-demo-bucket",
          "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:::amzn-s3-demo-bucket",
          "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
         }
      }
   }
}
```

Manage 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 Managed Service for Apache Flink and Studio notebook 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:

- Manage automatic snapshot creation
- Restore from a snapshot that contains incompatible state data

- Review snapshot API examples

Manage 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.

Restore 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*.

Review 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": 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"
      }
   }
}
```

Use 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:

- Upgrade applications using in-place version upgrades for Apache Flink
- Upgrade your application to a new Apache Flink version

- Roll back application upgrades
- General best practices and recommendations for application upgrades
- Precautions and known issues with application upgrades
- Upgrading to Flink 2.2: Complete guide
- State compatibility guide for Flink 2.2 upgrades

Upgrade 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: Update 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 with application upgrades.
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.

Upgrade 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 CloudFormation (CFN).
  - Update the RuntimeEnvironment field. Previously, CloudFormation deleted the application and created a new one, causing your snapshots and other app history to be lost. Now 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.

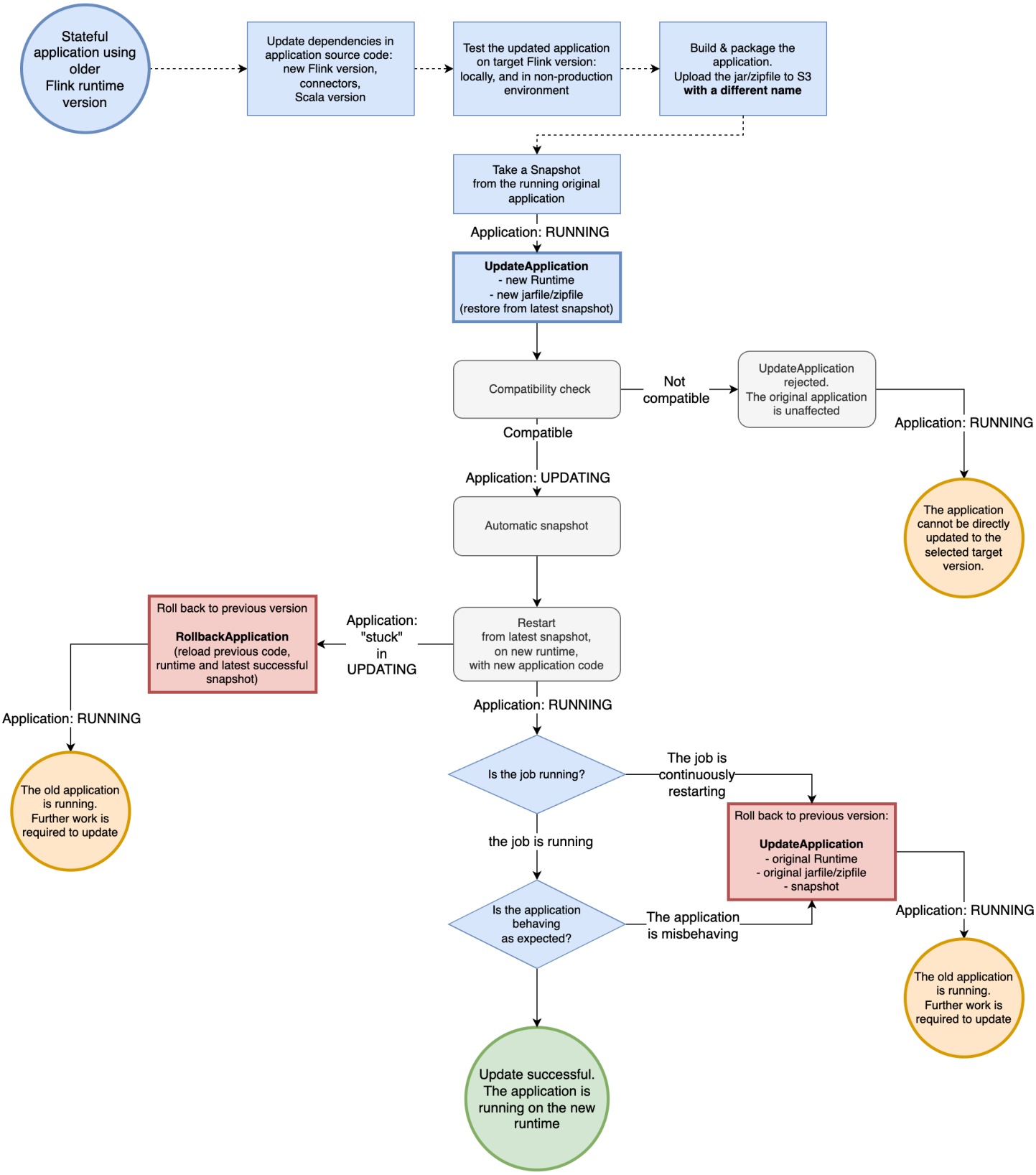
Upgrade an application in RUNNING state

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 kinesisanalyticsv2 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 Roll back application upgrades section to return to the healthy state.

Process flow for running state applications


Stateful application using older Flink runtime version
Update dependencies in application source code: new Flink version, connectors, Scala version
Test the updated application on target Flink version: locally, and in non-production environment
Build & package the application. Upload the jar/zipfile to S3 with a different name
Take a Snapshot from the running original application
Application: RUNNING
UpdateApplication - new Runtime - new jarfile/zipfile (restore from latest snapshot)
Compatibility check
Not compatible
UpdateApplication rejected. The original application is unaffected
Application: RUNNING
The application cannot be directly updated to the selected target version.
Compatible
Application: UPDATING
Automatic snapshot
Restart from latest snapshot, on new runtime, with new application code
Application: "stuck" in UPDATING
Roll back to previous version
RollbackApplication (reload previous code, runtime and latest successful snapshot)
Application: RUNNING
The old application is running. Further work is required to update
Application: RUNNING
Is the job running?
The job is continuously restarting
Roll back to previous version:
UpdateApplication
- original Runtime
- original jarfile/zipfile
- snapshot
Application: RUNNING
The old application is running. Further work is required to update
the job is running
Is the application behaving as expected?
The application is misbehaving
Update successful. The application is running on the new runtime

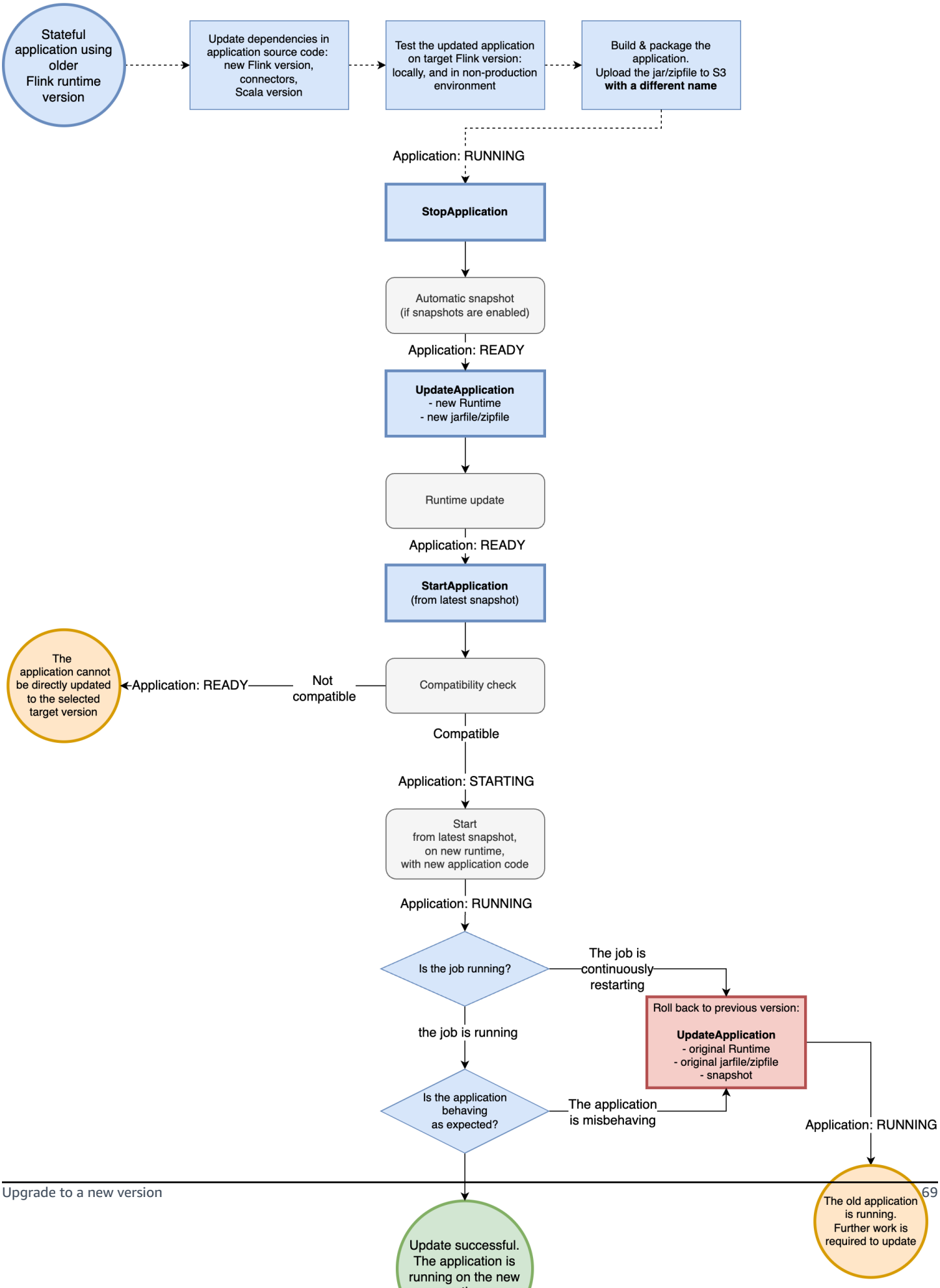
Upgrade an application in READY state

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 kinesisanalyticsv2 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


Stateful application using older Flink runtime version
Update dependencies in application source code: new Flink version, connectors, Scala version
Test the updated application on target Flink version: locally, and in non-production environment
Build & package the application. Upload the jar/zipfile to S3 with a different name
Application: RUNNING
StopApplication
Automatic snapshot (if snapshots are enabled)
Application: READY
UpdateApplication - new Runtime - new jarfile/zipfile
Runtime update
Application: READY
StartApplication (from latest snapshot)
Compatibility check
Not compatible
Application: READY
The application cannot be directly updated to the selected target version
Compatible
Application: STARTING
Start from latest snapshot, on new runtime, with new application code
Application: RUNNING
Is the job running?
The job is continuously restarting
Roll back to previous version: UpdateApplication - original Runtime - original jarfile/zipfile - snapshot
the job is running
Is the application behaving as expected?
The application is misbehaving
Application: RUNNING
Update successful. The application is running on the new
The old application is running. Further work is required to update

Roll back application upgrades

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, 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 for application upgrades

- 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 with application upgrades

Kafka Commit on checkpointing fails repeatedly after a broker restart

There is a known open source Apache Flink issue with the Apache Kafka connector in Flink version 1.15 caused by a critical open source Kafka Client bug in Kafka Client 2.8.1. For more information, see Kafka Commit on checkpointing fails repeatedly after a broker restart and KafkaConsumer is unable to recover connection to group coordinator after commitOffsetAsync exception.

To avoid this issue, we recommend that you use Apache Flink 1.18 or later in Amazon Managed Service for Apache Flink.

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.

Upgrading to Flink 2.2: Complete guide

This guide provides step-by-step instructions for upgrading your Amazon Managed Service for Apache Flink application from Flink 1.x to Flink 2.2. This is a major version upgrade with breaking changes that require careful planning and testing.

> **Major version upgrade is uni-directional**
>
> The Upgrade operation can move your application from Flink 1.x to 2.2 with state preservation, but you cannot move back from 2.2 to 1.x with 2.2 state. If your application becomes unhealthy after upgrading, use the Rollback API to return to the 1.x version with your original 1.x state from the latest snapshot.

Prerequisites

Before beginning your upgrade:

- Review Breaking changes and deprecations
- Review State compatibility guide for Flink 2.2 upgrades
- Ensure you have a non-production environment for testing
- Document your current application configuration and dependencies

Understanding your migration paths

Your upgrade experience depends on your application's compatibility with Flink 2.2. Understanding these paths helps you prepare appropriately and set realistic expectations.

Path 1: Compatible binary and application state

What to expect:

- Invoke the Upgrade operation
- Complete the migration to 2.2 with the application status transitioning: `RUNNING` → `UPDATING` → `RUNNING`
- Preserve all application state without data loss or reprocessing
- Same experience as minor version migrations

Best for: Stateless applications or applications using compatible serialization (Avro, compatible Protobuf schemas, POJOs without collections)

Path 2: Binary incompatibilities

What to expect:

- Invoke the Upgrade operation
- Operation fails and surfaces the binary incompatibility through Operations API and logs
- With auto-rollback enabled: Applications automatically roll back within minutes without your intervention
- With auto-rollback disabled: Applications remain in running state without data processing; you manually roll back to older version
- Once the binary is fixed, use the UpdateApplication API for an experience similar to Path 1

Best for: Applications using removed APIs that are detected during Flink job startup

Path 3: Incompatible application state

What to expect:

- Invoke the Upgrade operation

- Migration appears to succeed initially
- Applications enter restart loops within seconds as state restoration fails
- Detect failures through CloudWatch Metrics showing continuous restarts
- Manually invoke the Rollback operation
- Return to production within minutes after initiating rollback
- Review State migration for your application

Best for: Applications with state serialization incompatibilities (POJOs with collections, certain Kryo-serialized state)

> **Note**
>
> It is highly recommended to create a replica of your production application and test each of the following phases of the upgrade on the replica before following the same steps for your production application.

Phase 1: Preparation

Update application code

Update your application code to be compatible with Flink 2.2:

- **Update Flink dependencies** to version 2.2.1 in your `pom.xml` or `build.gradle`
- **Update connector dependencies** to Flink 2.2-compatible versions (see Connector availability)
- **Remove deprecated API usage**:
  - Replace DataSet API with DataStream API or Table API/SQL
  - Replace legacy `SourceFunction`/`SinkFunction` with FLIP-27 Source and FLIP-143 Sink APIs
  - Replace Scala API usage with Java API
- **Update to Java 17**

Upload updated application code

- Build your application JAR with Flink 2.2 dependencies

- Upload to Amazon S3 with a **different file name** than your current JAR (for example, `my-app-flink-2.2.jar`)
- Note the S3 bucket and key for use in the upgrade step

Phase 2: Enable auto-rollback

Auto-rollback allows Amazon Managed Service for Apache Flink to automatically revert to the previous version if the upgrade fails.

Check auto-rollback status

AWS Management Console:

1. Navigate to your application
2. Choose **Configuration**
3. Under **Application settings**, verify **System rollback** is enabled

AWS CLI:

```
aws kinesisanalyticsv2 describe-application \
    --application-name MyApplication \
    --query
 'ApplicationDetail.ApplicationConfigurationDescription.ApplicationSystemRollbackConfigurationD
```

Enable auto-rollback (if not enabled)

```
aws kinesisanalyticsv2 update-application \
    --application-name MyApplication \
    --current-application-version-id <version-id> \
    --application-configuration-update '{
        "ApplicationSystemRollbackConfigurationUpdate": {
            "RollbackEnabledUpdate": true
        }
    }'
```

Phase 3: Take snapshot (optional)

If automatic snapshots are enabled for your application you can skip this step, otherwise take a snapshot of your application to save the state of your application before upgrading.

Take snapshot from running application

AWS Management Console:

1. Navigate to your application
2. Choose **Snapshots**
3. Choose **Create snapshot**
4. Enter a snapshot name (for example, `pre-flink-2.2-upgrade`)
5. Choose **Create**

AWS CLI:

```
aws kinesisanalyticsv2 create-application-snapshot \
    --application-name MyApplication \
    --snapshot-name pre-flink-2.2-upgrade
```

Verify snapshot creation

```
aws kinesisanalyticsv2 describe-application-snapshot \
    --application-name MyApplication \
    --snapshot-name pre-flink-2.2-upgrade
```

Wait until `SnapshotStatus` is `READY` before proceeding.

Phase 4: Upgrade application

You can upgrade your Flink application by using the `UpdateApplication` action.

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

- **Use 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 CloudFormation.**

- Update the `RuntimeEnvironment` field. Previously, CloudFormation deleted the application and created a new one, causing your snapshots and other app history to be lost. Now 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 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.

Upgrade from RUNNING state

```
aws kinesisanalyticsv2 update-application \
    --application-name MyApplication \
    --current-application-version-id <version-id> \
    --runtime-environment-update FLINK-2_2 \
    --application-configuration-update '{
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "FileKeyUpdate": "my-app-flink-2.2.jar"
                }
            }
        }
    }'
```

Upgrade from READY state

```
aws kinesisanalyticsv2 update-application \
    --application-name MyApplication \
    --current-application-version-id <version-id> \
    --runtime-environment-update FLINK-2_2 \
    --application-configuration-update '{
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
```

```
                "S3ContentLocationUpdate": {
                    "FileKeyUpdate": "my-app-flink-2.2.jar"
                }
            }
        }
    }'
```

Phase 5: Monitor upgrade

Compatibility check

- Use the Operations API to check the status of the upgrade. If there are binary incompatibilities or issues with job startup, the upgrade operation will fail with logs.
- If the Upgrade Operation has succeeded but the application is stuck in restart loops, this means the state is incompatible with the new Flink version or there is a problem with the updated code. Review State compatibility guide for Flink 2.2 upgrades on how to identify state incompatibility issues.

Monitor application health

Application state:

- Application status should transition: `RUNNING` → `UPDATING` → `RUNNING`
- Check the runtime of the application. If it is 2.2, the upgrade operation was successful.
- If your application is in `RUNNING` but still on the older runtime, auto-rollback kicked in. Operations API will show operation as `FAILED`. Check logs to find the exception for failure.

In addition, monitor these metrics in CloudWatch:

Restart metric:

- `numRestarts`: Monitor for unexpected restarts — the upgrade is successful if `numRestarts` is zero and `uptime` or `runningTime` is increasing.

Checkpoint metrics:

- `lastCheckpointDuration`: Should be similar to pre-upgrade values
- `numberOfFailedCheckpoints`: Should remain at 0

Phase 6: Validate application behavior

After the application is running on Flink 2.2:

Functional validation

- Verify data is being read from sources
- Verify data is being written to sinks
- Verify business logic produces expected results
- Compare output with pre-upgrade baseline

Performance validation

- Monitor latency metrics (end-to-end processing time)
- Monitor throughput metrics (records per second)
- Monitor checkpoint duration and size
- Monitor memory and CPU utilization

Run for 24+ hours

Allow the application to run for at least 24 hours in production to ensure:

- No memory leaks
- Stable checkpoint behavior
- No unexpected restarts
- Consistent throughput

Phase 7: Rollback procedures

If the upgrade fails or the application is running but unhealthy, roll back to the previous version.

Automatic rollback

If auto-rollback is enabled and the upgrade fails during startup, Amazon Managed Service for Apache Flink automatically reverts to the previous version.

Manual rollback

If the application is running but unhealthy, use the `RollbackApplication` API:

AWS Management Console:

1. Navigate to your application
2. Choose **Actions** → **Roll back**
3. Confirm the rollback

AWS CLI:

```
aws kinesisanalyticsv2 rollback-application \
    --application-name MyApplication \
    --current-application-version-id <version-id>
```

What happens during rollback:

- Application stops
- Runtime reverts to previous Flink version
- Application code reverts to previous JAR
- Application restarts from the last successful snapshot taken **before** the upgrade

> **Important**
>
> - You cannot restore a Flink 2.2 snapshot on Flink 1.x
> - Rollback uses the snapshot taken before the upgrade
> - Always take a snapshot before upgrading (Phase 3)

Next steps

For questions or issues during upgrade, see the Troubleshoot Managed Service for Apache Flink or contact AWS Support.

State compatibility guide for Flink 2.2 upgrades

When upgrading from Flink 1.x to Flink 2.2, state compatibility issues may prevent your application from restoring from snapshots. This guide helps you identify potential compatibility issues and provides migration strategies.

Understanding state compatibility changes

Amazon Managed Service for Apache Flink 2.2 introduces several serialization changes that affect state compatibility. The following are the major ones:

- **Kryo Version Upgrade**: Apache Flink 2.2 upgrades the bundled Kryo serializer from version 2 to version 5. Because Kryo v5 uses a different binary encoding format than Kryo v2, any operator state that was serialized via Kryo in a Flink 1.x savepoint cannot be restored in Flink 2.2.
- **Java Collections Serialization**: In Flink 1.x, Java collections (such as `HashMap`, `ArrayList`, and `HashSet`) within POJOs were serialized using Kryo. Flink 2.2 introduces collection-specific optimized serializers that are incompatible with the Kryo-serialized state from 1.x. Applications using Java collections with POJO or Kryo serializers in 1.x cannot restore this state in Flink 2.2. See Flink documentation for more details on data types and serialization.
- **Kinesis Connector Compatibility**: The Kinesis Data Streams (KDS) connector version lower than 5.0 maintains state that is not compatible with the Flink 2.2 Kinesis connector version 6.0. You must migrate to connector version 5.0 or greater before your upgrade.

Serialization compatibility reference

Review all state declarations in your application and match serialization types to the table below. If any state type is incompatible, see the State migration section before proceeding with your upgrade.

Serialization compatibility reference

Serialization Type	Compatible?	Details
Avro (`SpecificRecord` , `GenericRecord`)	Yes	Uses its own binary format independent of Kryo. Ensure you are using Flink's native Avro type information, not

Serialization Type	Compatible?	Details
		Avro registered as a Kryo serializer.
Protobuf	Yes	Uses its own binary encoding independent of Kryo. Verify schema changes follow backward-compatible evolution rules.
POJOs without collections	Yes	Handled by Flink's POJO serializer — but only if the class meets all POJO criteria: public class, public no-arg constructor, all fields either public or accessible via getters/setters, and all field types themselves serializable by Flink. A POJO that violates any of these silently falls back to Kryo and becomes incompatible.
Custom TypeSerializers	Yes	Compatible only if your serializer does not delegate to Kryo internally.
SQL and Table API state	Yes (with caveat)	Uses Flink's internal serialize rs. However, Apache Flink does not guarantee state compatibility between major versions for Table API applications. Test in a non-production environment first.

Serialization Type	Compatible?	Details
POJOs with Java collections (`HashMap`, `ArrayList`, `HashSet`)	No	In Flink 1.x, collections within POJOs were serialized via Kryo v2. Flink 2.2 introduces dedicated collection serializers whose binary format is incompatible with the Kryo v2 format.
Scala case classes	No	Serialized via Kryo in Flink 1.x. The Kryo v2 to v5 upgrade changes the binary format.
Java records	No	Typically fall back to Kryo serialization in Flink 1.x. Verify by testing with `disableGenericTypes()`.
Third-party library types	No	Types without a registered custom serializer fall back to Kryo. The Kryo v2 to v5 binary format change breaks compatibility.
Any type using Kryo fallback	No	If Flink cannot handle a type with a built-in or registered serializer, it falls back to Kryo. All Kryo-serialized state from 1.x is incompatible with 2.2.

Diagnostic methods

You can either identify state compatibility issues proactively by looking at application logs or inspecting logs after the UpdateApplication API operation.

Identify Kryo fallback in your application

You can use the following regex pattern in your logs to identify Kryo fallback in your application:

```
Class class (?<className>[^\s]+) cannot be used as a POJO type
```

Sample log:

```
Class class org.apache.flink.streaming.connectors.kinesis.model.SequenceNumber
cannot be used as a POJO type because not all fields are valid POJO fields,
and must be processed as GenericType. Please read the Flink documentation on
"Data Types & Serialization" for details of the effect on performance and
schema evolution.
```

If the upgrade fails using the UpdateApplication API, the following exceptions might signal that you are encountering serializer-based state incompatibility:

IndexOutOfBoundsException

```
Caused by: java.lang.IndexOutOfBoundsException: Index 116 out of bounds for length 1
    at java.base/jdk.internal.util.Preconditions.outOfBounds(Unknown Source)
    at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Unknown Source)
    at java.base/jdk.internal.util.Preconditions.checkIndex(Unknown Source)
    at java.base/java.util.Objects.checkIndex(Unknown Source)
    at java.base/java.util.ArrayList.get(Unknown Source)
    at
 com.esotericsoftware.kryo.util.MapReferenceResolver.getReadObject(MapReferenceResolver.java:77
    at com.esotericsoftware.kryo.Kryo.readReferenceOrNull(Kryo.java:923)
    ... 23 more
```

StateMigrationException (POJOSerializer)

```
Caused by: org.apache.flink.util.StateMigrationException: The new state serializer
(org.apache.flink.api.java.typeutils.runtime.PojoSerializer@8bf85b5d) must not be
incompatible with the old state serializer
(org.apache.flink.api.java.typeutils.runtime.PojoSerializer@3282ee3).
```

Pre-upgrade checklist

- Review all state declarations in your application
- Check for POJOs with collections (`HashMap`, `ArrayList`, `HashSet`)

- Verify serialization methods for each state type
- Create a prod replica application and test state compatibility using UpdateApplication API on this replica
- If state is incompatible, select a strategy from State migration
- Enable auto-rollback in your production Flink application configuration

State migration

Rebuild complete state

Best for applications where state can be rebuilt from source data.

If your application can rebuild state from source data:

1. Stop the Flink 1.x application
2. Upgrade to Flink 2.x with updated code
3. Start with `SKIP_RESTORE_FROM_SNAPSHOT`
4. Allow application to rebuild state

```
aws kinesisanalyticsv2 start-application \
    --application-name MyApplication \
    --run-configuration '{
        "ApplicationRestoreConfiguration": {
            "ApplicationRestoreType": "SKIP_RESTORE_FROM_SNAPSHOT"
        }
    }'
```

Best practices

1. **Always use Avro or Protobuf for complex state** — These provide schema evolution and are Kryo-independent
2. **Avoid collections in POJOs** — Use Flink's native `ListState` and `MapState` instead
3. **Test state restoration locally** — Before production upgrade, test with actual snapshots
4. **Take snapshots frequently** — Especially before major version upgrades
5. **Enable auto-rollback** — Configure your MSF application to automatically rollback on failure

6. **Document your state types** — Maintain documentation of all state types and their serialization methods
7. **Monitor checkpoint sizes** — Growing checkpoint sizes may indicate serialization issues

Next steps

Plan your upgrade: See Upgrading to Flink 2.2: Complete guide.

For questions or issues during migration, see the Troubleshoot Managed Service for Apache Flink or contact AWS Support.

Implement 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

- Configure application parallelism and ParallelismPerKPU
- Allocate Kinesis Processing Units
- Update your application's parallelism
- Use automatic scaling in Managed Service for Apache Flink
- maxParallelism considerations

Configure 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.

Allocate 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 Update 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.

Update 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:::amzn-s3-demo-bucket",
            "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"
         }
      }
   }
}
```

Use automatic scaling in Managed Service for Apache Flink

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. Automatic scaling is enabled by default. 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 maximum `containerCPUUtilization` is larger than 75 percent or above for 15 minutes. That means the `ScaleUp` action is initiated when there are 15 consecutive datapoints with 1 minute period equal to or over 75 percent. A `ScaleUp` action doubles the `CurrentParallelism` of your application. `ParallelismPerKPU` is not modified. As a consequence, the number of allocated KPUs also doubles.

- Your application scales down (decreases parallelism) when your CPU usage remains below 10 percent for six hours. That means the `ScaleDown` action is initiated when there are 360 consecutive datapoints with 1 minute period less than 10 percent. A `ScaleDown` action halves (rounded up) the parallelism of the application. `ParallelismPerKPU` is not modified, and the number of allocated KPUs also halves (rounded up).

> **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 initialized.

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 KPUs that Managed Service for Apache Flink provisions which is a function of your application's `parallelism` and `parallelismPerKPU` 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 KPUs. For more information, see Managed Service for Apache Flink and Studio notebook 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 "Update your application's parallelism".
- 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".

Implement custom autoscaling

If you want finer grained control on autoscaling or use trigger metrics other than `containerCPUUtilization`, you can use this example:

- AutoScaling

  This examples illustrates how to scale your Managed Service for Apache Flink application using a different CloudWatch metric from the Apache Flink application, including metrics from Amazon MSK and Amazon Kinesis Data Streams, used as sources or sink.

For additional information, see Enhanced monitoring and automatic scaling for Apache Flink.

Implement scheduled autoscaling

If your workload follows a predictable profile over time, you might prefer to scale your Apache Flink application preemptively. This scales your application at a scheduled time, as opposed to scaling reactively based on a metric. To set up scaling up and down at fixed hours of the day, you can use this example:

- ScheduledScaling

maxParallelism considerations

The maximum parallelism a Flink job can scale is limited by the *minimum* `maxParallelism` across all operators of the job. For example, if you have a simple job with only a source and a sink, and the source has a `maxParallelism` of 16 and the sink has 8, the application can't scale beyond parallelism of 8.

To learn how the default `maxParallelism` of an operator is calculated and how to override the default, refer to Setting the Maximum Parallelism in the Apache Flink documentation.

As a basic rule, be aware that that if you don't define `maxParallelism` for any operator and you start your application with parallelism less than or equal to 128, all operators will have a `maxParallelism` of 128.

> **Note**
>
> The job's maximum parallelism is the upper limit of parallelism for scaling your application retaining the state.
> If you modify `maxParallelism` of an existing application, the application won't be able to restart from a previous snapshot taken with the old `maxParallelism`. You can only restart the application without snapshot.
> If you plan to scale your application to a parallelism greater that 128, you must explicitly set the `maxParallelism` in your application.

- Autoscaling logic will prevent scaling a Flink job to a parallelism that will exceed maximum parallelism of the job.
- If you use a custom autoscaling or scheduled scaling, configure them so that they don't exceed the maximum parallelism of the job.
- If you manually scale your application beyond maximum parallelism, the application fails to start.

Add tags to Managed Service for Apache Flink applications

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:

- Add tags when an application is created
- Add or update tags for an existing application
- List tags for an application
- Remove tags from an application

Add 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"
    }
]
```

Add or update 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"
      }
   ]
}
```

List 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"
}
```

Remove tags from an application

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

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

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

Use CloudFormation with Managed Service for Apache Flink

The following exercise shows how to start a Flink application created with 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 CloudFormation at AWS::KinesisAnalytics::Application.

Write 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 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 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))
              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 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.

Create 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.

Invoke 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.

Review an extended 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:
                - kinesisanlaytics.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))
```

```
                cfnresponse.send(event, context, cfnresponse.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))
              cfnresponse.send(event, context, cfnresponse.SUCCESS, {})
            except Exception as err:
              logger.error(err)
              cfnresponse.send(event,context, cfnresponse.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.

Use 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.

Access 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.

Access 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.

Access 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.

Supported and deprecated Apache Flink versions

This topic contains information about the supported versions of Apache Flink in Managed Service for Apache Flink. This topic also lists the supported Apache Flink features in each release.

> **Note**
>
> If you're using a version of Apache Flink that's deprecating, we recommend that you upgrade your application to the most recent supported Flink version using the Use 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	Note
2.3.0	Supported	Supported	Amazon Managed Service for Apache Flink 2.3	
2.2.1	Supported	Supported	Amazon Managed Service for Apache Flink 2.2	This is a major version with breaking changes. See Breaking changes and deprecations before proceedin g.
1.20.5	Supported	Supported	Amazon Managed Service for Apache Flink 1.20	

Apache Flink version	Status - Amazon Managed Service for Apache Flink	Status - Apache Flink community	Link	Note
1.19.1	Supported	Supported	Amazon Managed Service for Apache Flink 1.19	
1.18.1	Supported	Unsupported	Amazon Managed Service for Apache Flink 1.18	
1.15.2	Supported	Unsupported	Amazon Managed Service for Apache Flink 1.15	
1.13.1	Deprecating	Unsupported	Getting started: Flink 1.13.2	The support for this version in Amazon Managed Service for Apache Flink will end on October 16, 2025.

Apache Flink version	Status - Amazon Managed Service for Apache Flink	Status - Apache Flink community	Link	Note
1.11.1	Deprecating	Unsupported	Earlier version information for Managed Service for Apache Flink (This version won't be supported from February 2025)	We plan to end support for Apache Flink versions 1.6, 1.8, and 1.11 in Amazon Managed Service for Apache Flink. From July 14, 2025, we will place applications using these versions into a READY state. From July 28, 2025, you will not be able to START applications using these versions. We recommend that you now immediately upgrade your applications to Flink version 1.20 using the in-place version upgrades

Apache Flink version	Status - Amazon Managed Service for Apache Flink	Status - Apache Flink community	Link	Note
				feature in Amazon Managed Service for Apache Flink. For more information, see Use in-place version upgrades for Apache Flink.

Apache Flink version	Status - Amazon Managed Service for Apache Flink	Status - Apache Flink community	Link	Note
1.8.2	Deprecating	Unsupported	Earlier version information for Managed Service for Apache Flink (This version won't be supported from February 2025)	We plan to end support for Apache Flink versions 1.6, 1.8, and 1.11 in Amazon Managed Service for Apache Flink. From July 14, 2025, we will place applications using these versions into a READY state. From July 28, 2025, you will not be able to START applications using these versions. We recommend that you now immediately upgrade your applications to Flink version 1.20 using the in-place version upgrades

Apache Flink version	Status - Amazon Managed Service for Apache Flink	Status - Apache Flink community	Link	Note
				feature in Amazon Managed Service for Apache Flink. For more information, see Use in-place version upgrades for Apache Flink.

Apache Flink version	Status - Amazon Managed Service for Apache Flink	Status - Apache Flink community	Link	Note
1.6.2	Deprecating	Unsupported	Earlier version information for Managed Service for Apache Flink (This version won't be supported from February 2025)	We plan to end support for Apache Flink versions 1.6, 1.8, and 1.11 in Amazon Managed Service for Apache Flink. From July 14, 2025, we will place applications using these versions into a READY state. From July 28, 2025, you will not be able to START applications using these versions. We recommend that you now immediately upgrade your applications to Flink version 1.20 using the in-place version upgrades

Apache Flink version	Status - Amazon Managed Service for Apache Flink	Status - Apache Flink community	Link	Note
				feature in Amazon Managed Service for Apache Flink. For more information, see Use in-place version upgrades for Apache Flink.

Topics

- Amazon Managed Service for Apache Flink 2.3
- Amazon Managed Service for Apache Flink 2.2
- Amazon Managed Service for Apache Flink 1.20
- 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 2.3

Managed Service for Apache Flink now supports Apache Flink version 2.3.0. This section introduces you to the key new features and changes introduced with Managed Service for Apache Flink support of Apache Flink 2.3.0.

> **Note**
>
> If you are using an earlier supported version of Apache Flink and want to upgrade your existing applications to Apache Flink 2.3.0, you can do so using in-place Apache Flink

version upgrades. For more information, see Use 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.

Note

Upgrading from Flink 1.x to Flink 2.3 is a major version upgrade with breaking changes and state compatibility considerations. Before upgrading from 1.x, review Breaking changes and deprecations and State compatibility guide for Flink 2.2 upgrades.

Supported features

Apache Flink 2.3.0 introduces improvements in the SQL API, adaptive partition selection for skewed workloads, and the Flink dashboard.

Supported features and related documentation

Supported features	Description	Apache Flink documentation reference
Improvements to SinkUpsertMaterializer and changelog disorder	Improves update ordering when writing CDC or changelog streams to upsert sinks.	FLIP-558: Improvements to SinkUpsertMaterializer and changelog disorder
Application Management read APIs	New read-only endpoints for application-level observability. Write endpoints are disabled in Amazon Managed Service for Apache Flink.	FLIP-549: Application Management, FLIP-560: Application Management Capability Enhancement
Adaptive Partition Selection for StreamPartitioner	Opt-in throughput optimization for skewed `Rebalance` and `Rescale` partitioners. Disabled by default; enable	FLIP-339: Adaptive Partition Selection for StreamPartitioner

Supported features	Description	Apache Flink documentation reference
	with `taskmanager.network.adaptive-partitioner.enabled` .	

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 checkpoint during recovery	This is disabled by default as it is an experimental feature in open source. You can request to enable it for your application using a support case.	FLIP-547: Support checkpoint during recovery

For the Apache Flink 2.3.0 release documentation, see Apache Flink Documentation v2.3.0. For Flink 2.3 release notes, see Release notes - Flink 2.3.

Changes in Amazon Managed Service for Apache Flink with Apache Flink 2.3

Watermark alignment default preserved

Apache Flink 2.3 changes the default behavior of watermark alignment to prevent backlogged jobs from consuming all downstream resources. In Amazon Managed Service for Apache Flink, the default value of `pipeline.watermark-alignment.max-drift` is set to `0` to preserve backward compatibility with earlier supported Apache Flink versions and protect your existing applications from a behavioral regression on upgrade. If you want the new fairness behavior, you can opt in by

setting `pipeline.watermark-alignment.max-drift` programmatically in your application code. For more information, see FLINK-37399: Watermark alignment prevents backlogged jobs from using all resources.

Components

Components for Amazon Managed Service for Apache Flink 2.3

Component	Version
Java	17 (recommended)
Python	3.12
Kinesis Data Analytics Flink Runtime (aws-kinesisanalytics-runtime)	1.2.0
Connectors	For information about available connectors, see Apache Flink connectors.

Unsupported and experimental features

The following Apache Flink 2.3 features are not supported in Amazon Managed Service for Apache Flink:

- **Materialized Tables**: Continuously maintained, queryable table snapshots.
- **Custom Telemetry Changes**: Custom metric reporters and telemetry configurations.
- **Flink Native S3 FileSystem (FLIP-555)**: Native S3 File system implementation (experimental in open source)
- **ForSt State Backend**: Disaggregated state storage (experimental in open source).
- **Java 21**: Experimental support in open source, not supported in Managed Service for Apache Flink.

Amazon Managed Service for Apache Flink Studio

Flink 2.3 in Amazon Managed Service for Apache Flink does not support Studio applications. For more information, see Creating a Studio notebook.

Amazon Managed Service for Apache Flink 2.2

Amazon Managed Service for Apache Flink now supports Apache Flink version 2.2. This is the first major version upgrade for the service. This page covers the capabilities introduced in Flink 2.2, along with important considerations for upgrading from Flink 1.x.

> **Note**
>
> Flink 2.2 introduces breaking changes that require careful planning. Review the full list of breaking changes and deprecations below and the State compatibility guide for Flink 2.2 upgrades before upgrading from 1.x.

What's new in Amazon Managed Service for Apache Flink 2.2

Amazon Managed Service for Apache Flink 2.2 introduces behavioral changes that may break existing applications upon upgrade. Review these carefully alongside the Flink API changes in the next section.

Programmatic Configuration Handling

- MSF Flink 2.2 now reports an exception when customers attempt to modify configs not supported by MSF through `env.getConfig().set()` or similar APIs. See Programmatic Flink configuration properties.
- Customers can still request to change certain configs through support tickets (see Modifiable Settings)

Metrics Removal

- The `fullRestarts` metric has been removed in Flink 2.2. Use the `numRestarts` metric instead.
- The `bytesRequestedPerFetch` metric for KDS connector has been removed in Flink AWS connector version 6.0.0 (only connector version compatible with Flink 2.2).
- The `uptime` and `downtime` metrics are both marked as deprecated in Flink 2.2 and will be removed soon. Replace `uptime` with the new metric `runningTime`. Replace `downtime` with one or more of `restartingTime`, `cancellingTime`, and `failingTime`.
- See Metrics and Dimensions page for full list of supported metrics.

Non-Credential IMDS Calls Blocked

- These allowed endpoints are used by the AWS SDK's **DefaultCredentialsProvider** (`/latest/meta-data/iam/security-credentials/`) and **DefaultAwsRegionProviderChain** (`/latest/dynamic/instance-identity/document`) to automatically configure credentials and region for your application.
- Applications using AWS SDK functions that rely on non-credential IMDS calls (such as `EC2MetadataUtils.getInstanceId()`, `EC2MetadataUtils.getInstanceType()`, `EC2MetadataUtils.getLocalHostName()`, or `EC2MetadataUtils.getAvailabilityZone()`) will receive HTTP 4xx errors when attempting these calls.
- If your application uses IMDS for instance metadata or other information outside the allowed paths, refactor your code to use environment variables or application configuration instead.

Read-Only Root Filesystem

- To improve security, any dependency outside of `/tmp` which is the default flink working directory will result in: `java.io.FileNotFoundException: /{path}/{filename} (Read-only file system)`.
- Filesystem dependencies can originate directly from your code or indirectly from libraries included in your dependencies. Override direct filesystem dependencies to `/tmp/` in your code. For indirect filesystem dependencies from libraries, use library configuration overrides to redirect filesystem operations to `/tmp/`.

Breaking changes and deprecations

Below is a summary of breaking changes and deprecations introduced in Managed Service for Apache Flink 2.2. See Apache Flink 2.0 Release Notes for full release notes of Apache Flink 2.0 that introduces these breaking changes.

Flink API and Language Removals

DataSet API Removed

- The legacy DataSet API for batch processing has been completely removed in Flink 2.0+. All batch processing must now use the unified DataStream API.

- Applications using the DataSet API must be migrated to DataStream API before upgrading. See Apache Flink migration guide for DataSet to DataStream conversion

Java 11 and Python 3.8 Removed

- Java 11 support completely removed; Java 17 is the default and recommended runtime.
- Python 3.8 support removed; Python 3.12 is now the default.

Legacy Connector Classes Removed

- The legacy `SourceFunction` and `SinkFunction` interfaces have been superseded by the new unified Source (FLIP-27) and Sink (FLIP-143) APIs, which provide better support for bounded/unbounded duality, improved checkpoint coordination, and a cleaner programming model.
- For Kinesis Data Streams, use `KinesisStreamsSource` and `KinesisStreamsSink` from `flink-connector-aws-kinesis-streams:6.0.0-2.0`.

Scala API removed

- The Flink Scala API has been removed. Flink's Java API is now the single supported API for JVM-based applications.
- If your application is written in Scala, you can still use Flink's Java API from Scala code — the main change is that the Scala-specific wrappers and implicit conversions are no longer available. See Upgrading Applications and Flink Versions for details on updating your Scala applications.

State Compatibility Considerations

- Kryo serializer upgraded from version 2.24 to 5.6 may cause state compatibility issues.
- POJOs with collections (`HashMap`, `ArrayList`, `HashSet`) may have state compatibility issues.
- Avro and Protobuf serialization unaffected.
- See State compatibility guide for Flink 2.2 upgrades for detailed assessment to triage your application's risk level.

Apache Flink 2.2 features supported

Runtime and language support

Feature	Description	Documentation
Java 17 Runtime	Java 17 is now the default and recommended runtime; Java 11 support removed.	Java Compatibility
Python 3.12 Support	Python 3.12 now supported; Python 3.8 support removed.	PyFlink Documentation

State management and performance

Feature	Description	Documentation
RocksDB 8.10.0	Improved I/O performance with RocksDB upgrade.	State Backends
Serialization Improvements	Dedicated serializers for Map, List, Set; Kryo upgraded from 2.24 to 5.6.	Type Serialization

SQL and Table API features

Feature	Description	Documentation
VARIANT Data Type	Native support for semi-structured data (JSON) without repeated string parsing.	Data Types
Delta Join	Reduces state requirements for streaming joins by maintaining only the latest version of each key; requires customer-managed infrastructure (for example, Apache Fluss).	Joins

Feature	Description	Documentation
StreamingMultiJoinOperator	Executes multi-way joins as a single operator, eliminating intermediate materialization.	FLIP-516
ProcessTableFunction (PTF)	Enables stateful, event-driven logic directly in SQL with per-key state and timers.	User-Defined Functions
ML_PREDICT Function	Call registered ML models on streaming/batch tables directly from SQL. Requires customer to bundle a ModelProvider implement ation (e.g., `flink-model-openai`). ModelProvider libraries are not shipped by Managed Service for Apache Flink.	ML Predict
Model DDL	Define ML models as first-class catalog objects using CREATE MODEL statements.	CREATE Statements
Vector Search	Flink SQL API supports searching vector databases. No open source `VectorSearchTableSource` implementation is currently available; customers must provide their own implement ation.	Flink SQL

DataStream API features

Feature	Description	Documentation
FLIP-27 Source API	New unified source interface replacing legacy SourceFunction.	Sources
FLIP-143 Sink API	New unified sink interface replacing legacy SinkFunction.	Sinks
Async Python DataStream	Non-blocking I/O operations in Python DataStream API using AsyncFunction.	Async I/O

Connector availability

When upgrading to Flink 2.2, you also need to update your connector dependencies to versions that are compatible with the Flink 2.2 runtime. Flink connectors are released independently from the Flink runtime, and not all connectors have a Flink 2.2-compatible release yet. The following table summarizes the availability of commonly used connectors in Amazon Managed Service for Apache Flink:

Connector availability for Flink 2.2

Connector	Flink 1.20 Version	Flink 2.0+ Version	Notes
Apache Kafka	flink-connector-kafka 3.4.0-1.20	flink-connector-kafka 4.0.0-2.0	Recommended for Flink 2.2
Kinesis Data Streams (source)	flink-connector-kinesis 5.0.0-1.20	flink-connector-aws-kinesis-streams 6.0.0-2.0	Recommended for Flink 2.2
Kinesis Data Streams (sink)	flink-connector-aws-kinesis-streams 5.1.0-1.20	flink-connector-aws-kinesis-streams 6.0.0-2.0	Recommended for Flink 2.2

Connector	Flink 1.20 Version	Flink 2.0+ Version	Notes
Amazon Data Firehose	flink-connector-aws-kinesis-firehose 5.1.0-1.20	flink-connector-aws-kinesis-firehose 6.0.0-2.0	Compatible with Flink 2.0
Amazon DynamoDB	flink-connector-dynamodb 5.1.0-1.20	flink-connector-dynamodb 6.0.0-2.0	Compatible with Flink 2.0
Amazon SQS	flink-connector-sqs 5.1.0-1.20	flink-connector-sqs 6.0.0-2.0	Compatible with Flink 2.0
FileSystem (S3, HDFS)	Bundled with Flink	Bundled with Flink	Built into the Flink distribution — always available
JDBC	flink-connector-jdbc 3.3.0-1.20	Not yet released for 2.x	No Flink 2.x-compatible release available
OpenSearch	flink-connector-opensearch 1.2.0-1.19	Not yet released for 2.x	No Flink 2.x-compatible release available
Elasticsearch	Legacy connector only	Not yet released for 2.x	Consider migrating to the OpenSearch connector
Amazon Managed Service for Prometheus	flink-connector-prometheus 1.0.0-1.20	Not yet released for 2.x	No Flink 2.x-compatible release available

- If your application depends on a connector that does not yet have a Flink 2.x release, you have two options: wait for the connector to release a compatible version, or evaluate whether you can replace it with an alternative (for example, using the JDBC catalog or a custom sink).
- When updating connector versions, pay attention to artifact name changes — some connectors were renamed between major versions (for example, the Firehose connector changed from `flink-connector-aws-kinesis-firehose` to `flink-connector-aws-firehose` in some intermediate versions).

- Always check the Amazon Managed Service for Apache Flink connector documentation for the exact artifact names and versions supported in your target runtime.

Unsupported and experimental features

The following features are not supported in Amazon Managed Service for Apache Flink 2.2:

- **Materialized Tables**: Continuously maintained, queryable table snapshots.
- **Custom Telemetry Changes**: Custom metric reporters and telemetry configurations.
- **ForSt State Backend**: Disaggregated state storage (experimental in open source).
- **Java 21**: Experimental support in open source, not supported in Managed Service for Apache Flink.

Known issues

Amazon Managed Service for Apache Flink Studio

Flink 2.2 in Amazon Managed Service for Apache Flink does not support Studio applications. For more information, see Creating a Studio notebook.

Kinesis Connector EFO

- Applications using the `KinesisStreamsSource` with EFO (Enhanced Fan-Out / SubscribeToShard) path introduced in connector v5.0.0 and v6.0.0 may fail when Kinesis streams undergo resharding. This is a known issue in the community. For more information, see FLINK-37648.
- Applications using the `KinesisStreamsSource` with EFO (Enhanced Fan-Out / SubscribeToShard) path introduced in connector v5.0.0 and v6.0.0 together with `KinesisStreamsSink` may experience deadlocks if the Flink application is under backpressure, resulting in a complete stop of data processing in one or more TaskManagers. A force stop operation and a start application operation are needed to recover the application. This is a sub-case of the known issue in the community. For more information, see FLINK-34071.

Upgrade experience

Amazon Managed Service for Apache Flink supports in-place version upgrades that preserve your application configuration, logs, metrics, tags, and—if state and binaries are compatible—your application state. For step-by-step instructions, see Upgrading to Flink 2.2: Complete guide.

For guidance on assessing state compatibility risk and handling incompatible state during upgrades, see State compatibility guide for Flink 2.2 upgrades.

Next steps

- New to Flink 2.2? For detailed Apache Flink 2.2 documentation, see Apache Flink 2.2 Documentation.
- Planning an upgrade? See Upgrading to Flink 2.2: Complete guide
- State compatibility concerns? See State compatibility guide for Flink 2.2 upgrades

For questions or issues, see the Troubleshoot Managed Service for Apache Flink or contact AWS Support.

Amazon Managed Service for Apache Flink 1.20

Managed Service for Apache Flink now supports Apache Flink version 1.20.5. This section introduces you to the key new features and changes introduced with Managed Service for Apache Flink support of Apache Flink 1.20.5. Apache Flink 1.20 is expected to be the last 1.x release and a Flink long-term support (LTS) version. For more information, see FLIP-458: Long-Term Support for the Final Release of Apache Flink 1.x Line.

> **Note**
>
> If you are using an earlier supported version of Apache Flink and want to upgrade your existing applications to Apache Flink 1.20.5, you can do so using in-place Apache Flink version upgrades. For more information, see Use 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 1.20.5 introduces improvements in the SQL APIs, in the DataStream APIs, and in the Flink dashboard.

Supported features and related documentation

Supported features	Description	Apache Flink documentation reference
Add DISTRIBUTED BY clause	Many SQL engines expose the concepts of `Partitioning` , `Bucketing` , or `Clustering` . Flink 1.20 introduces the concept of `Bucketing` to Flink.	FLIP-376: Add DISTRIBUTED BY clause
DataStream API: Support Full Partition Proessing	Flink 1.20 introduces built-in support for aggregations on non-keyed streams through the `FullPartitionWindow` API.	FLIP-380: Support Full Partition Processing on Non-keyed DataStream
Show data skew score on Flink Dashboard	The Flink 1.20 dashboard now shows data skew infrmation. Each operator on the Flink job graph UI shows an additional data skew score.	FLIP-418: Show data skew score on Flink Dashboard

For the Apache Flink 1.20.5 release documentation, see Apache Flink Documentation v1.20.5 on the Apache Flink website. For Flink 1.20 release notes, see Release notes - Flink 1.20

Components

Flink 1.20 components

Component	Version
Java	11 (recommended)

Component	Version
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)	From version 2.70.0. For more information, see Flink Version Compatibility.

Known issues

Amazon Managed Service for Apache Flink Studio

Amazon Managed Service for Apache Flink 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 1.20. This work is scheduled with the Zeppelin community. We will update these notes when that work 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.

Backported bug fixes

Amazon Managed Service for Apache Flink backports fixes from the Flink community for critical issues. Following is a list of bug fixes that we have backported:

Backported bug fixes

Apache Flink JIRA link	Description
FLINK-35886	This fix addresses an issue causing incorrect accounting of watermark idleness timeouts when a subtask is backpressured/blocked.

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 Use 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

Supported features	Description	Apache Flink documentation reference
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 more information, see What's New in Python 3.11.	FLINK-33030: Add python 3.11 support
Provide metrics for TwoPhaseCommitting sink	Users can view statistic s around the status of committers in two phase committing sinks.	FLIP-371: Provide initializ ation context for Committer creation in TwoPhaseC ommittingSink
Trace Reporters for job restart and checkpointing	Users can now monitor traces around checkpoint duration and recvery trends. In Amazon Managed Service for Apache Flink, we enable Slf4j trace reporters by default, so users can monitor checkpoin t 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
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 the 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.

Backported bug fixes

Amazon Managed Service for Apache Flink backports fixes from the Flink community for critical issues. This means that the runtime differs from the Apache Flink 1.19.1 release. Following is a list of bug fixes that we have backported:

Backported bug fixes

Apache Flink JIRA link	Description
FLINK-35531	This fix addresses the performance regression introduced in 1.17.0 that causes slower writes to HDFS.
FLINK-35157	This fix addresses the issue of stuck Flink jobs when sources with watermark alignment encounter finished subtasks.
FLINK-34252	This fix addresses the issue in watermark generation that results in an erroneous IDLE watermark state.
FLINK-34252	This fix addresses the performance regressio n during watermark generation by reducing system calls.
FLINK-33936	This fix addresses the issue with duplicate records during mini-batch aggregation on Table API.
FLINK-35498	This fix addresses the issue with argument name conflicts when defining named parameters in Table API UDFs.
FLINK-33192	This fix addresses the issue of a state memory leak in window operators due to improper timer cleanup.
FLINK-35069	This fix addresses the issue when a Flink job gets stuck triggering a timer at the end of a window.

Apache Flink JIRA link	Description
FLINK-35832	This fix addresses the issue when IFNULL returns incorrect results.
FLINK-35886	This fix addresses the issue when backpress ured tasks are considered as idle.

Components

Component	Version
Java	11 (recommended)
Python	3.11
Kinesis Data Analytics Flink Runtime (aws-kine sisanalytics-runtime)	1.2.0
Connectors	For information about available connectors, see Apache Flink connectors.
Apache Beam (Beam applications only)	From version 2.61.0. For more information, see Flink Version Compatibility.

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 1.19. 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 Use in-place version upgrades for Apache Flink.

Supported features with Apache Flink documentation references

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

Supported Features	Description	Apache Flink documentation reference
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 alignment, and adds the related benchmark.	FLINK-32524: Watermark aggregation performance
Make watermark alignment ready for production use	Removes risk of large jobs overloading JobManager	FLINK-32548: Make watermark alignment ready
Configurable RateLimitingStratey 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.

Backported bug fixes

Amazon Managed Service for Apache Flink backports fixes from the Flink community for critical issues. This means that the runtime differs from the Apache Flink 1.18.1 release. Following is a list of bug fixes that we have backported:

Backported bug fixes

Apache Flink JIRA link	Description
FLINK-33863	This fix addresses the issue when a state restore fails for compressed snapshots.
FLINK-34063	This fix addresses the issue when source operators lose splits when snapshot compression is enabled. 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.
FLINK-35069	This fix addresses the issue when a Flink job gets stuck triggering a timer at the end of a window.
FLINK-35097	This fix addresses the pissue of duplicate records in a Table API Filesystem connector with the raw format.
FLINK-34379	This fix addresses the issue of an OutOfMemoryError when enabling dynamic table filtering.

Apache Flink JIRA link	Description
FLINK-28693	This fix addresses the issue of the Table API being unable to generate a graph if the watermark has a columnBy expression.
FLINK-35217	This fix addresses the issue of a corrupted checkpoint during a specific Flink job failure mode.

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)	From version 2.57.0. For more information, see Flink Version Compatibility.

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.

Incorrect watermark idleness when subtask is backpressured

There is a known issue in watermark generation when a subtask is backpressured, which has been fixed from Flink 1.19 and later. This can show up as a spike in the number of late records when a Flink job graph is backpressured. We recommend that you upgrade to the latest Flink version to pull in this fix. For more information, see Incorrect watermark idleness timeout accounting when subtask is backpressured/blocked.

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	FLIP-34: Terminate/Suspend Job with Savepoint.

Feature	Description	Apache FLIP reference
	exactly-once semantics for customers that rely on them.	
Scala Decoupling	Users can now leverage the Java API from any Scala version, including Scala 3. Customers will need to bundle the Scala standard library of their choice in their Scala applications.	FLIP-28: Long-term goal of making flink-table Scala-free.
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 `numRestarts` in parallel with `fullRestarts` 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 Use 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

Component	Version
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

Known issues

Kafka Commit on checkpointing fails repeatedly after a broker restart

There is a known open source Apache Flink issue with the Apache Kafka connector in Flink version 1.15 caused by a critical open source Kafka Client bug in Kafka Client 2.8.1. For more information, see Kafka Commit on checkpointing fails repeatedly after a broker restart and KafkaConsumer is unable to recover connection to group coordinator after commitOffsetAsync exception.

To avoid this issue, we recommend that you use Apache Flink 1.18 or later in Amazon Managed Service for Apache Flink.

Earlier version information for Managed Service for Apache Flink

Note

Apache Flink versions 1.6, 1.8, and 1.11 haven't been supported by the Apache Flink community for over three years. We issued notice of this change in June 2024 and October 2024 and will now end support for these versions in Amazon Managed Service for Apache Flink.

- On July 14, 2025, we'll stop your applications and place them into a READY state. You'll be able to re-start your applications at that time and continue to use your applications as normal, subject to service limits.
- From July 28, 2025, we'll disable the ability to START your applications. You won't be able to start or operate your Flink version 1.6 applications from this time.

We recommend that you immediately upgrade any existing applications using Apache Flink version 1.6, 1.8, or 1.11, to Apache Flink version 1.20. This is the most recent supported Flink version. You can upgrade your applications using the in-place version upgrades feature in Amazon Managed Service for Apache Flink. For more information, see Use in-place version upgrades for Apache Flink.
If you have further questions or concerns, you can contact AWS Support.

Note

Apache Flink version **1.13** has not been supported by the Apache Flink community for over three years. We now plan to end support for this version in Amazon Managed Service for Apache Flink on **October 16, 2025**. After this date, you will no longer be able to create, start, or run applications using Apache Flink version 1.13 in Amazon Managed Service for Apache Flink.
You can upgrade your applications statefully using the in-place version upgrades feature in Managed Service for Apache Flink. For more information, see Use in-place version upgrades for Apache Flink.

Version **1.15.2** is supported by Managed Service for Apache Flink, but is 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 Create an application.

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 Create an application.

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, CloudFormation, or the AWS Management Console. For more information, see Use 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 Add streaming data sources.
- **Operators:** The application processes data by using one or more *operators*. An operator can transform, enrich, or aggregate data. For more information, see 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 Write data using 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 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

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference 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
IAM	(Recommended) Use console credentials as 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. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
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. • For the AWS CLI, see Configuring the AWS

Which user needs programmatic access?	To	By
		CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
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 *IAM User Guide*.
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. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Next step

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 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-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 Use 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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

   JSON

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

```
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-
java-apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "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
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.

JSON

```
{
    "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 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 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 "Set up application logging in Managed Service for Apache Flink".

Update Environment Properties

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

To update environment properties for the application

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

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

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

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

Update the Application Code

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

> **Note**
>
> To load a new version of the application code with the same file name, you must specify the new object version. For more information about using Amazon S3 object versions, see Enabling or Disabling Versioning.

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

The following sample request for the UpdateApplication action reloads the application code and restarts the application. Update the CurrentApplicationVersionId to the current application version. You can check the current application version using the ListApplications or DescribeApplication actions. Update the bucket name suffix (*<username>*) with the suffix that you chose in the the section called “Create two Amazon Kinesis data streams” 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
- 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. Sign in to the AWS Management Console, and open the Amazon MSF 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:** Offical 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 Use in-place version upgrades for Apache Flink.

This topic contains a version of the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Add streaming data sources.
- **Operators:** The application processes data by using one or more *operators*. An operator can transform, enrich, or aggregate data. For more information, see 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 Write data using 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 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

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference 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
IAM	(Recommended) Use console credentials as 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. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
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. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in

Which user needs programmatic access?	To	By
		the *AWS SDKs and Tools Reference Guide*.
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 *IAM User Guide*.
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. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Next step

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 Use 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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

   JSON

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

```
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-
java-apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "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	flink.inputstream.initpos	LATEST
ProducerConfigProperties	aws.region	us-west-2
ProducerConfigProperties	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. 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.

JSON

```
{
    "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 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 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 "Set up application logging in Managed Service for Apache Flink".

Update environment properties

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

To update environment properties for the application

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

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

```
    }
}
```

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

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

Update the application code

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

> **Note**
>
> To load a new version of the application code with the same file name, you must specify the new object version. For more information about using Amazon S3 object versions, see Enabling or Disabling Versioning.

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

The following sample request for the UpdateApplication action reloads the application code and restarts the application. Update the CurrentApplicationVersionId to the current application version. You can check the current application version using the ListApplications or DescribeApplication actions. Update the bucket name suffix (*<username>*) with the suffix that you chose in the the section called “Create two Amazon Kinesis data streams” 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 rour 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. Sign in to the AWS Management Console, and open the Amazon MSF 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 rour 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:** Offical 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 Use in-place version upgrades for Apache Flink.

This topic contains a version of the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Add streaming data sources.
- **Operators:** The application processes data by using one or more *operators*. An operator can transform, enrich, or aggregate data. For more information, see 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 Write data using 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

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference 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
IAM	(Recommended) Use console credentials as 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. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
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. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests	Following the instructions in Using temporary credentia

Which user needs programmatic access?	To	By
	to the AWS CLI, AWS SDKs, or AWS APIs.	ls with AWS resources in the *IAM User Guide*.
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. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

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 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_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 Use 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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

   JSON

```
{
```

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

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

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.

JSON

```
{
    "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 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 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 "Set up application logging in Managed Service for Apache Flink".

Update environment properties

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

To update environment properties for the application

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

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

```
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
   }
}
```

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

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

Update the application code

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

> **Note**
>
> To load a new version of the application code with the same file name, you must specify the new object version. For more information about using Amazon S3 object versions, see Enabling or Disabling Versioning.

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

The following sample request for the UpdateApplication action reloads the application code and restarts the application. Update the CurrentApplicationVersionId to the current application version. You can check the current application version using the ListApplications or DescribeApplication actions. Update the bucket name suffix (*<username>*) with the suffix that you chose in the the section called “Create two Amazon Kinesis data streams” 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. Sign in to the AWS Management Console, and open the Amazon MSF 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 Use in-place version upgrades for Apache Flink.

This topic contains a version of the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Add streaming data sources.

- **Operators:** The application processes data by using one or more *operators*. An operator can transform, enrich, or aggregate data. For more information, see 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 Write data using 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

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference 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
IAM	(Recommended) Use console credentials as 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. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
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. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in

Which user needs programmatic access?	To	By
		the *AWS SDKs and Tools Reference Guide*.
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 *IAM User Guide*.
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. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

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 Use 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 "Specify 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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-
started-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "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.

JSON

```
{
    "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 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 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 "Set up application logging in Managed Service for Apache Flink".

Update environment properties

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

To update environment properties for the application

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

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

```
                }
            },
            {
                "PropertyGroupId": "ConsumerConfigProperties",
                "PropertyMap" : {
                     "aws.region" : "us-west-2"
                }
            }
         ]
      }
   }
}
```

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

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

Update the application code

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

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

The following sample request for the UpdateApplication action reloads the application code and restarts the application. Update the CurrentApplicationVersionId to the current application version. You can check the current application version using the ListApplications or DescribeApplication actions. Update the bucket name suffix (*<username>*) with the suffix that you chose in the the section called “Create two Amazon Kinesis data streams” 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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

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
- Tutorial: Get started using the DataStream API in Managed Service for Apache Flink

> **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 for creating and working with Managed Service for Apache Flink applications.

In this exercise, you create a Managed Service for Apache Flink application that aggregates data using a tumbling window. Aggregration 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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Complete the required prerequisites in the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

   JSON

```
{
    "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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

> **Note**
>
> To set up required prerequisites for this exercise, first complete the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Complete the required prerequisites in the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

   JSON

```
{
    "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. Sign in to the AWS Management Console, and open the Amazon MSF 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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Complete the required prerequisites in the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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. Sign in to the AWS Management Console, and open the Amazon MSF 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**
>
> Aggregration 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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink tutorial.

> **Note**
>
> If you deleted the Amazon S3 bucket from the Getting Started tutorial, follow the the section called "Upload the application code JAR file" 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. Sign in to the AWS Management Console, and open the Amazon MSF 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	*`The bootstrap server list you saved previously`*
`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	*The bootstrap server list you saved previously*
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 Use 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 Troubleshoot Managed Service for Apache Flink topic.

Example: Use an EFO consumer with a Kinesis data stream

> **Note**
>
> For current examples, see Examples for creating and working with Managed Service for Apache Flink applications.

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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Complete the required prerequisites in the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

JSON

```
{
    "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.recordpublisher	EFO
ConsumerConfigProperties	flink.stream.efo.consumername	basic-efo-flink-app
ConsumerConfigProperties	INPUT_STREAM	ExampleInputStream
ConsumerConfigProperties	flink.inputstream.initpos	LATEST
ConsumerConfigProperties	AWS_REGION	us-west-2

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

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

8. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
9. For **CloudWatch logging**, select the **Enable** check box.
10. 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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Complete the required prerequisites in the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-
started-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "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.

JSON

```
{
    "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 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 "Set up application logging in Managed Service for Apache Flink".

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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 kinesis create-stream \
--stream-name SourceAccountExampleInputStream \
--shard-count 1 \
--profile ka-source-stream-account-profile
```

Create and update IAM roles and policies

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

- Creating IAM Roles
- Creating IAM Policies

Sink account roles and policies

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

   > **Note**
   >
   > When you use the console to create your application, the console creates a policy called `kinesis-analytics-service-<application name>-<application`

region>, and a role called 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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AssumeRoleInSourceAccount",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::123456789012: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-2:123456789012:log-group:*"
            ]
        },
        {
```

```
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:123456789012:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:123456789012: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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:us-west-2:111122223333: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.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333: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 for creating and working with Managed Service for Apache Flink applications.

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 Kakfa 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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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 Tutorial: Get started using the DataStream API in Managed Service for Apache Flink tutorial.

> **Note**
>
> If you deleted the Amazon S3 bucket from the Getting Started tutorial, follow the the section called “Upload the application code JAR file” 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. Sign in to the AWS Management Console, and open the Amazon MSF 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	*The bootstrap server list you saved previously*
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

Group ID	Key	Value
KafkaSink	bootstrap.servers	*The bootstrap server list you saved previously*
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 Use 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 Troubleshoot Managed Service for Apache Flink 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 for creating and working with Managed Service for Apache Flink applications.

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 Tutorial: Get started using Python in Managed Service for Apache Flink 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. Sign in to the AWS Management Console, and open the Amazon MSF 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
`consumer.config.0`	`input.stream.name`	`ExampleInputStream`
`consumer.config.0`	`aws.region`	`us-west-2`
`consumer.config.0`	`scan.stream.initpos`	`LATEST`

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

Group ID	Key	Value
producer.config.0	output.stream.name	ExampleOutputStream
producer.config.0	aws.region	us-west-2
producer.config.0	shard.count	1

8. Under **Properties**, choose **Add group** again. For **Group ID**, enter **kinesis.analytics.flink.run.options**. This special property group tells your application where to find its code resources. For more information, see Specify your code files.
9. Enter the following:

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

10. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
11. For **CloudWatch logging**, select the **Enable** check box.
12. Choose **Update**.

Note

When you choose to enable CloudWatch logging, 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.

   JSON

```
{
    "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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

> **Note**
>
> To set up required prerequisites for this exercise, first complete the Tutorial: Get started using Python in Managed Service for Apache Flink 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. Sign in to the AWS Management Console, and open the Amazon MSF 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
`consumer.config.0`	`input.stream.name`	`ExampleInputStream`
`consumer.config.0`	`aws.region`	`us-west-2`
`consumer.config.0`	`scan.stream.initpos`	`LATEST`

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

Group ID	Key	Value
`producer.config.0`	`output.stream.name`	`ExampleOutputStream`
`producer.config.0`	`aws.region`	`us-west-2`
`producer.config.0`	`shard.count`	`1`

8. Under **Properties**, choose **Add group** again. For **Group ID**, enter **`kinesis.analytics.flink.run.options`**. This special property group tells your application where to find its code resources. For more information, see Specify your code files.
9. Enter the following application properties and values:

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

10. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
11. For **CloudWatch logging**, select the **Enable** check box.
12. Choose **Update**.

> **Note**
>
> When you choose to enable CloudWatch logging, 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.

JSON

```
{
    "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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

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 Tutorial: Get started using Python in Managed Service for Apache Flink 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. Sign in to the AWS Management Console, and open the Amazon MSF 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
`consumer.config.0`	`input.stream.name`	`ExampleInputStream`
`consumer.config.0`	`aws.region`	`us-west-2`
`consumer.config.0`	`scan.stream.initpos`	`LATEST`

   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 Specify your code files.
7. Enter the following application properties and values:

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

8. 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 Specify your code files.
9. Enter the following application properties and values: (replace *`bucket-name`* with the actual name of your Amazon S3 bucket.)

Group ID	Key	Value
`sink.config.0`	**`output.bucket.name`**	***`bucket-name`***

10. Under **Monitoring**, ensure that the **Monitoring metrics level** is set to **Application**.
11. For **CloudWatch logging**, select the **Enable** check box.
12. Choose **Update**.

> **Note**
>
> When you choose to enable CloudWatch logging, 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.

JSON

```
{
    "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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

> **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-seconds 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. Sign in to the AWS Management Console, and open the Amazon MSF 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	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**
ConsumerConfigProperties	**flink.stream.initpos**	**LATEST**

   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.

JSON

```
{
    "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-specfic 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 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": "tumbling_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": "tumbling_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": "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 kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

Update the application code

When you need to update your application code with a new version of your code package, you use the UpdateApplication 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": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

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-seconds 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. Sign in to the AWS Management Console, and open the Amazon MSF 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	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**

Group ID	Key	Value
ConsumerConfigProperties	flink.stream.initpos	LATEST

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.

   JSON

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

Update the application code

When you need to update your application code with a new version of your code package, you use the UpdateApplication 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": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

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. Sign in to the AWS Management Console, and open the Amazon MSF 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 for creating and working with Managed Service for Apache Flink applications.

> **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. Sign in to the AWS Management Console, and open the Amazon MSF 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-*<user-name>* /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.

   JSON

```
{
    "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.

JSON

```
{
    "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-2:123456789012:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
```

```
                "arn:aws:logs:us-west-2:123456789012:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:123456789012:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-
west-2:123456789012:stream/ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-
west-2:123456789012: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 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": "s3_sink",
       "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": "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 kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

Update the application code

When you need to update your application code with a new version of your code package, you use the UpdateApplication 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": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

Use 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 Tutorial: Create a Studio notebook in Managed Service for Apache Flink. 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:

- Use the correct Studio notebook Runtime version
- Create a Studio notebook
- Perform an interactive analysis of streaming data
- Deploy as an application with durable state

- Review IAM permissions for Studio notebooks
- Use connectors and dependencies
- Implement user-defined functions
- Enable checkpointing
- Upgrade Studio Runtime
- Work with AWS Glue
- Examples and tutorials for Studio notebooks in Managed Service for Apache Flink
- Troubleshoot Studio notebooks for Managed Service for Apache Flink
- Create custom IAM policies for Managed Service for Apache Flink Studio notebooks

Use the correct Studio notebook Runtime version

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 Upgrade Studio Runtime.

Studio Runtime

Apache Flink version	Apache Zeppelin version	Python version	
1.15	0.1	3.8	Recommended
1.13	0.9	3.8	Supported until October 16, 2024

Apache Flink version	Apache Zeppelin version	Python version	
1.11	0.9	3.7	Deprecating on February 24, 2025

Create 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 Work 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**.

Perform an 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 for Studio notebooks in Managed Service for Apache Flink 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
%flink	FlinkInterpreter	Creates ExecutionEnvironment/StreamExecution

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

For more information about Flink interpreters, see Flink interpreter for Apache Zeppelin.

If you are using `%flink.pyflink` or `%flink.ipyflink` 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
`senv`	`StreamExecutionEnvironment`
`stenv`	`StreamTableEnvironment for blink planner`

You access Python table environment resources with the following variables:

Variable	Resource
`s_env`	`StreamExecutionEnvironment`
`st_env`	`StreamTableEnvironment for blink planner`

For more information about using table environments, see Concepts and Common API in the Apache Flink documentation.

Deploy 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 Deploy 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 (`senv`, `stenv` for Scala; `s_env`, `st_env` for Python) and not the older "Flink" planner (`stenv_2` for Scala, `st_env_2` for Python). The Apache Flink project recommends the use of the Blink planner for production use cases, and this is the default planner in Zeppelin and in Flink.
- Your Python paragraphs must not use shell invocations/assignments using `!` or IPython magic commands like `%timeit` or `%conda` in notes meant to be deployed as applications.

- You can't use Scala case classes as parameters of functions passed to higher-order dataflow operators like `map` and `filter`. For information about Scala case classes, see CASE CLASSES in the Scala documentation.

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.

Review 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
Amazon EC2	List
AWS Glue	Read, Write
Managed Service for Apache Flink	Read
Managed Service for Apache Flink V2	Read
Amazon S3	Read, Write

Use 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.

Add 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 you 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. ssql(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.

Enable checkpointing

You enable checkpointing by using environment settings. For information about checkpointing, see Fault Tolerance in the Managed Service for Apache Flink Developer Guide.

Set 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"
)
```

Set 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"
)
```

Upgrade 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.

Upgrade your notebook to a new Studio Runtime

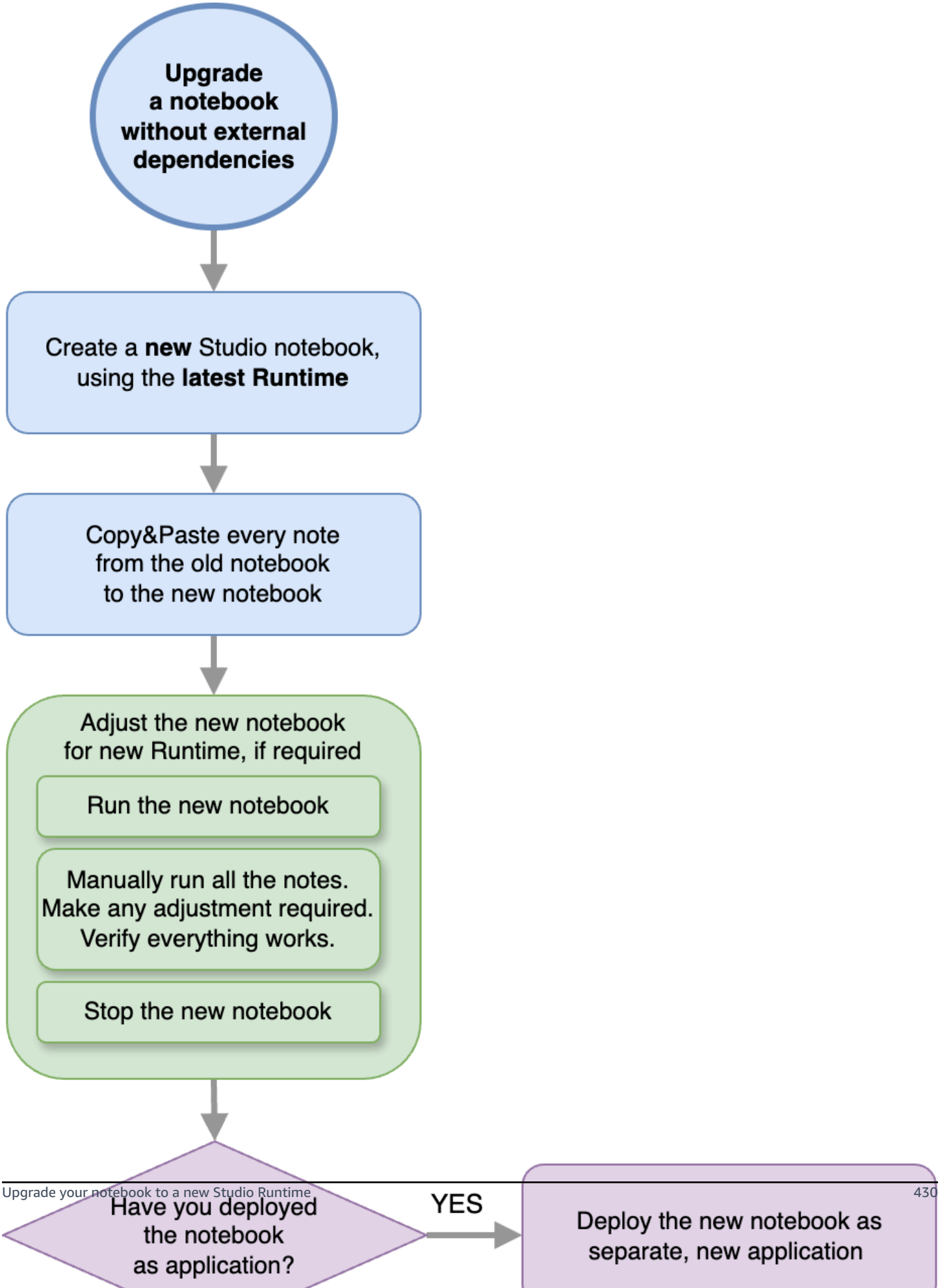
Depending on how you use Studio, the steps to upgrade your Runtime differ. Select the option that fits your use case.

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, reardless 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

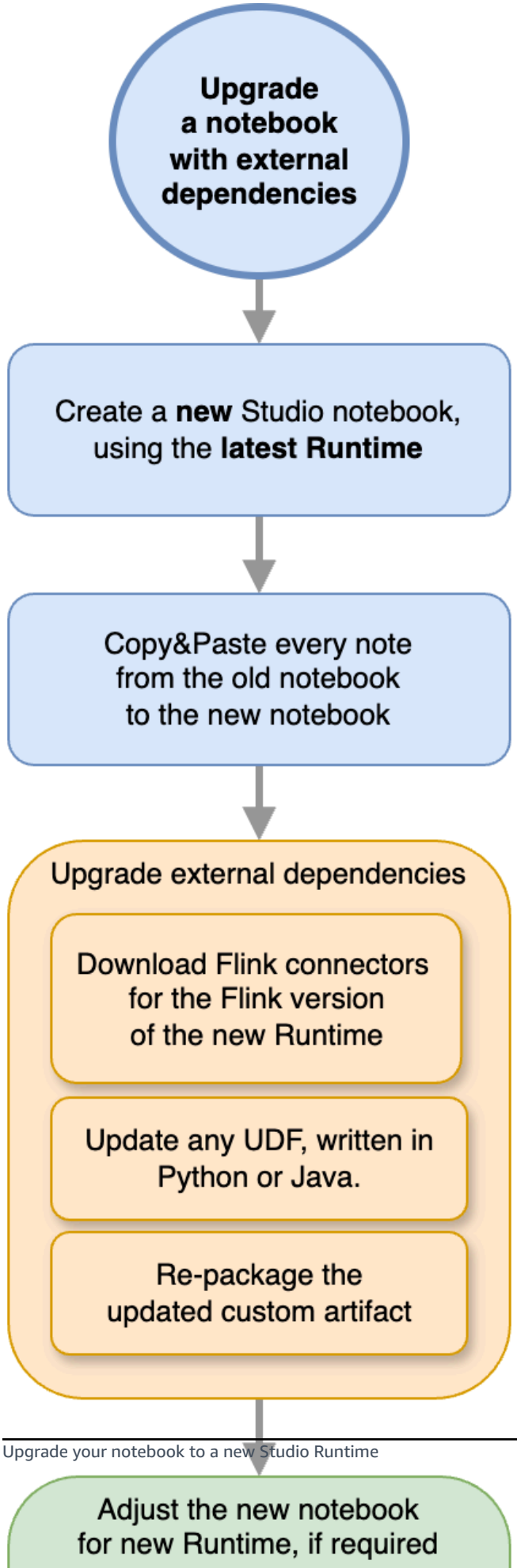

Upgrade a notebook without external dependencies
Create a new Studio notebook, using the latest Runtime
Copy&Paste every note from the old notebook to the new notebook
Adjust the new notebook for new Runtime, if required
Run the new notebook
Manually run all the notes. Make any adjustment required. Verify everything works.
Stop the new notebook
Have you deployed the notebook as application?
YES
Deploy the new notebook as separate, new application

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


Upgrade a notebook with external dependencies
Create a new Studio notebook, using the latest Runtime
Copy&Paste every note from the old notebook to the new notebook
Upgrade external dependencies
Download Flink connectors for the Flink version of the new Runtime
Update any UDF, written in Python or Java.
Re-package the updated custom artifact
Adjust the new notebook for new Runtime, if required

Work 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 Tutorial: Create a Studio notebook in Managed Service for Apache Flink 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:

- Define Apache Flink time values: These properties define how Managed Service for Apache Flink emits Apache Flink internal data processing time values.
- Use 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**.

Define Apache Flink time values

Apache Flink provides time values that describe when stream processing events occured, 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.*column_name*.milliseconds	The watermark interval in milliseconds

Use 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`.

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

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

Examples and tutorials for Studio notebooks in Managed Service for Apache Flink

Topics

- Tutorial: Create a Studio notebook in Managed Service for Apache Flink
- Tutorial: Deploy a Studio notebook as a Managed Service for Apache Flink application with durable state
- View example queries to analyza data in a Studio notebook

Tutorial: Create 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:

- Complete the prerequisites
- Create an AWS Glue database
- Next steps: Create a Studio notebook with Kinesis Data Streams or Amazon MSK
- Create a Studio notebook with Kinesis Data Streams
- Create a Studio notebook with Amazon MSK
- Clean up your application and dependent resources

Complete the prerequisites

Make sure 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: Create a Studio notebook with Kinesis Data Streams or Amazon MSK

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

- Create a Studio notebook with 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.

- Create a Studio notebook with 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.

Create 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:

- Complete the prerequisites
- Create an AWS Glue table
- Create a Studio notebook with Kinesis Data Streams
- Send data to your Kinesis data stream
- Test your Studio notebook

Complete the prerequisites

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 kinesisanalyticsv2 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/ZeppelinRole",
...
```

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 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 CloudFormation stack to finish being created. After the stack is complete, open the **Kinesis-Data-Generator-Cognito-User** stack in the 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.

Create 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:

- Set up an Amazon MSK cluster
- 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

Set up an Amazon MSK cluster

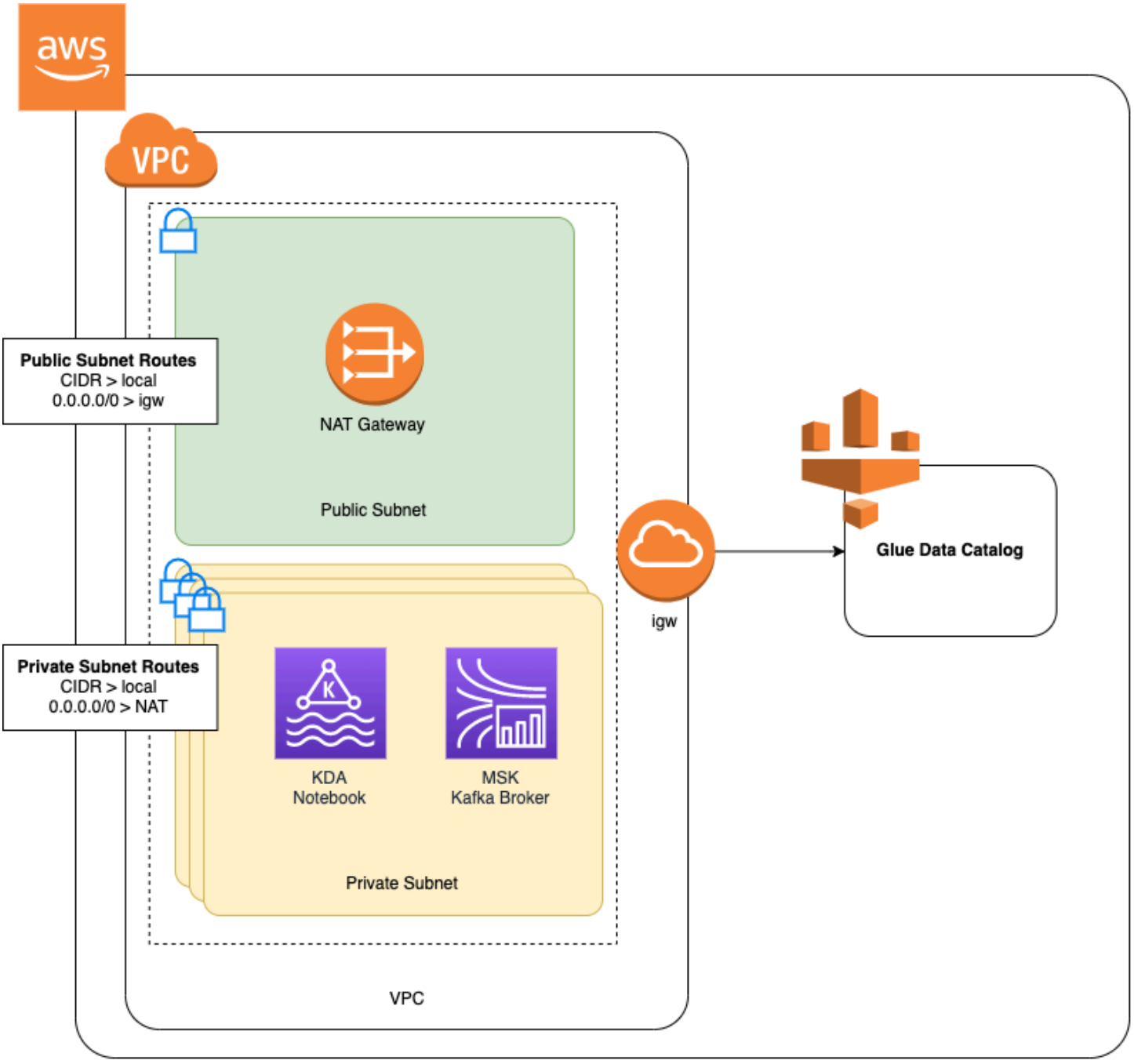
For this tutorial, you need an Amazon MSK cluster that allows plaintext access. If you don't have an Amazon MSK cluster set up already, follow the Getting Started Using Amazon MSK tutorial to create an Amazon VPC, an Amazon MSK cluster, a topic, and an Amazon EC2 client instance.

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 Elasic 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 the following properties:

   - key: `managed-flink.proctime`, value: `proctime`
   - key: `flink.properties.group.id`, value: `test-consumer-group`
   - key: `flink.properties.auto.offset.reset`, value: `latest`
   - key: `classification`, value: `json`

Without these key/value pairs, the Flink notebook runs into an error.

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

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

```
{
    "ApplicationDetail": {
        "ApplicationARN": "arn:aws:kinesisanalyticsus-
east-1:012345678901:application/MyNotebook",
        "ApplicationName": "MyNotebook",
        "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
        "ApplicationMode": "INTERACTIVE",
        "ServiceExecutionRole": "arn:aws:iam::012345678901:role/ZeppelinRole",
...
```

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

```
aws kinesisanalyticsv2 start-application --application-arn
 arn:aws:kinesisanalyticsus-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.

Clean 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: Deploy a Studio notebook as a Managed Service for Apache Flink 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:

- Complete prerequisites
- Deploy an application with durable state using the AWS Management Console
- Deploy an application with durable state using the AWS CLI

Complete prerequisites

Create a new Studio notebook by following the Tutorial: Create a Studio notebook in Managed Service for Apache Flink, 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 Review 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 Complete the prerequisites.

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

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

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

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

View example queries to analyza data in a Studio notebook

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

- Create tables with Amazon MSK/Apache Kafka

- Create tables with Kinesis
- Query a tumbling window
- Query a sliding window
- Use interactive SQL
- Use the BlackHole SQL connector
- Use Scala to generate sample data
- Use interactive Scala
- Use interactive Python
- Use a combination of interactive Python, SQL, and Scala
- Use a 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.

Create 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.

Create 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.

Query a 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)
```

Query a 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)
```

Use 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 “Use Scala to generate sample data” 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`;
```

Use the 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
```

Use Scala to generate sample data

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`
```

Use interactive Scala

This is the Scala translation of the the section called "Use 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`
```

Use interactive Python

This is the Python translation of the the section called “Use 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`
```

Use a combination of 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.ssql
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.ssql(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!")
```

Use a 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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {}
        }
    ]
}
```

Use the following paragraph for the create table statement.

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

```
)
```

Troubleshoot Studio notebooks for Managed Service for Apache Flink

This section contains troubleshooting information for Studio notebooks.

Stop 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.

Deploy 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: 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
```

Cancel 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)
```

Restart 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**.

Create custom IAM policies for Managed Service for Apache Flink Studio notebooks

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.

JSON

```
{
    "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:us-east-1:123456789012:connection/*",
                "arn:aws:glue:us-east-1:123456789012:table/<database-name>/*",
                "arn:aws:glue:us-east-1:123456789012:database/<database-name>",
                "arn:aws:glue:us-east-1:123456789012:database/hive",
                "arn:aws:glue:us-east-1:123456789012: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:

JSON

```
{
    "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:us-east-1:123456789012:stream/<stream-
name>"
        },
        {
            "Sid": "KinesisEfoConsumer",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStreamConsumer",
                "kinesis:SubscribeToShard"
            ],
            "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/<stream-
name>/consumer/*"
        }
    ]
}
```

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

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "KinesisStreamSink",
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords",
                "kinesis:DescribeStreamSummary",
                "kinesis:DescribeStream"
            ],
            "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/<stream-name>"
        }
    ]
}
```

If your application accesses an encypted 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.

Get 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 implementing an application in Java using 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
- Set up an AWS account and create an administrator user
- Set up the AWS Command Line Interface (AWS CLI)
- Create and run a Managed Service for Apache Flink application
- Clean up AWS resources
- Explore additional resources

Review the components of the Managed Service for Apache Flink application

> **Note**
>
> Amazon Managed Service for Apache Flink supports all Apache Flink APIs and potentially all JVM languages. For more information, see Flink's APIs.
> Depending on the API you choose, the structure of the application and the implementation is slightly different. This Getting Started tutorial covers the implementation of the applications using the DataStream API in Java.

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

A typical Managed Service for Apache Flink application has the following components:

- **Runtime properties:** You can use *runtime properties* to pass configuration parameters to your application to change them without modifying and republishing the code.
- **Sources:** The application consumes data from one or more *sources*. A source uses a connector to read data from an external system, such as a Kinesis data stream, or a Kafka bucket. For more information, see Add streaming data sources.
- **Operators:** The application processes data by using one or more *operators*. An operator can transform, enrich, or aggregate data. For more information, see Operators.
- **Sinks:** The application sends data to external sources through *sinks*. A sink uses a connector to send data to a Kinesis data stream, a Kafka topic, Amazon S3, or a relational database. You can also use a special connector to print the output for development purposes only. For more information, see Write data using sinks.

Your application requires some *external dependencies*, such as the Flink connectors that your application uses, or potentially a Java library. To run in Amazon Managed Service for Apache Flink, the application must be packaged along with dependencies in a *fat-jar* and uploaded to an Amazon S3 bucket. You then create a Managed Service for Apache Flink application. You pass the location of the code package, along with any other runtime configuration parameter.

This tutorial demonstrates how to use Apache Maven to package the application, and how to run the application locally in the IDE of your choice.

Fulfill the prerequisites for completing the exercises

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

- Git client. Install the Git client, if you haven't already.
- Java Development Kit (JDK) version 11 . Install a Java JDK 11 and set the `JAVA_HOME` environment variable to point to your JDK install location. If you don't have a JDK 11, you can use Amazon Coretto 11 or any other standard JDK of your choice.
  - To verify that you have the JDK installed correctly, run the following command. The output will be different if you are using a JDK other than Amazon Corretto. Make sure that the version is 11.x.

```
$ java --version

openjdk 11.0.23 2024-04-16 LTS
```

```
OpenJDK Runtime Environment Corretto-11.0.23.9.1 (build 11.0.23+9-LTS)
OpenJDK 64-Bit Server VM Corretto-11.0.23.9.1 (build 11.0.23+9-LTS, mixed mode)
```

- Apache Maven. Install Apache Maven if you haven't already. To learn how to install it, see Installing Apache Maven.
  - To test your Apache Maven installation, enter the following:

```
$ mvn -version
```

- IDE for local development. We recommend that you use a development environment such as Eclipse Java Neon or IntelliJ IDEA to develop and compile your application.
  - To test your Apache Maven installation, enter the following:

```
$ mvn -version
```

To get started, go to Set up an AWS account and create an administrator user.

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

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference Guide*.

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-east-1 US East (N. Virginia) 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

Create and run a Managed Service for Apache Flink application

Create and run a Managed Service for Apache Flink application

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

This section contains the following steps:

- Create dependent resources
- Set up your local development environment
- Download and examine the Apache Flink streaming Java code
- Write sample records to the input stream
- Run your application locally
- Observe input and output data in Kinesis streams
- Stop your application running locally
- Compile and package your application code
- Upload the application code JAR file
- Create and configure the Managed Service for Apache Flink application
- Next step

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 for input and output
- An Amazon S3 bucket to store the application's code

> **Note**
>
> This tutorial assumes that you are deploying your application in the us-east-1 US East (N. Virginia) Region. If you use another Region, adapt all steps accordingly.

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 streams using the AWS CLI, use the following commands, adjusting to the Region you use for your application.

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-east-1 \
```

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-east-1 \
```

Create an Amazon S3 bucket for the application code

You can create the Amazon S3 bucket using the console. To learn how to create an Amazon S3 bucket using the console, see Creating a bucket in the Amazon S3 User Guide. Name the Amazon S3 bucket using a globally unique name, for example by appending your login name.

> **Note**
>
> Make sure that you create the bucket in the Region you use for this tutorial (us-east-1).

Other resources

When you create your application, Managed Service for Apache Flink automatically creates the following Amazon CloudWatch resources if they don't already exist:

- A log group called `/AWS/KinesisAnalytics-java/<my-application>`
- A log stream called `kinesis-analytics-log-stream`

Set up your local development environment

For development and debugging, you can run the Apache Flink application on your machine directly from your IDE of choice. Any Apache Flink dependencies are handled like regular Java dependencies using Apache Maven.

> **Note**
>
> On your development machine, you must have Java JDK 11, Maven, and Git installed. We recommend that you use a development environment such as Eclipse Java Neon or IntelliJ IDEA. To verify that you meet all prerequisites, see Fulfill the prerequisites for completing the exercises. You **do not** need to install an Apache Flink cluster on your machine.

Authenticate your AWS session

The application uses Kinesis data streams to publish data. When running locally, you must have a valid AWS authenticated session with permissions to write to the Kinesis data stream. Use the following steps to authenticate your session:

1. If you don't have the AWS CLI and a named profile with valid credential configured, see Set up the AWS Command Line Interface (AWS CLI).
2. Verify that your AWS CLI is correctly configured and your users have permissions to write to the Kinesis data stream by publishing the following test record:

```
$ aws kinesis put-record --stream-name ExampleOutputStream --data TEST --partition-key TEST
```

3. If your IDE has a plugin to integrate with AWS, you can use it to pass the credentials to the application running in the IDE. For more information, see AWS Toolkit for IntelliJ IDEA and AWS Toolkit for Eclipse.

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.

Review application components

The application is entirely implemented in the `com.amazonaws.services.msf.BasicStreamingJob` class. The `main()` method defines the data flow to process the streaming data and to run it.

> **Note**
>
> For an optimized developer experience, the application is designed to run without any code changes both on Amazon Managed Service for Apache Flink and locally, for development in your IDE.

- To read the runtime configuration so it will work when running in Amazon Managed Service for Apache Flink and in your IDE, the application automatically detects if it's running standalone locally in the IDE. In that case, the application loads the runtime configuration differently:
  1. When the application detects that it's running in standalone mode in your IDE, form the `application_properties.json` file included in the **resources** folder of the project. The content of the file follows.
  2. When the application runs in Amazon Managed Service for Apache Flink, the default behavior loads the application configuration from the runtime properties you will define in the Amazon Managed Service for Apache Flink application. See Create and configure the Managed Service for Apache Flink application.

```
private static Map<String, Properties>
 loadApplicationProperties(StreamExecutionEnvironment env) throws IOException {
    if (env instanceof LocalStreamEnvironment) {
        LOGGER.info("Loading application properties from '{}'",
 LOCAL_APPLICATION_PROPERTIES_RESOURCE);
        return KinesisAnalyticsRuntime.getApplicationProperties(
                BasicStreamingJob.class.getClassLoader()

 .getResource(LOCAL_APPLICATION_PROPERTIES_RESOURCE).getPath());
    } else {
        LOGGER.info("Loading application properties from Amazon Managed Service for
 Apache Flink");
        return KinesisAnalyticsRuntime.getApplicationProperties();
    }
}
```

- The `main()` method defines the application data flow and runs it.
  - Initializes the default streaming environments. In this example, we show how to create both the `StreamExecutionEnvironment` to be used with the DataSteam API and the

`StreamTableEnvironment` to be used with SQL and the Table API. The two environment objects are two separate references to the same runtime environment, to use different APIs.

```
StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();
```

- Load the application configuration parameters. This will automatically load them from the correct place, depending on where the application is running:

```
Map<String, Properties> applicationParameters = loadApplicationProperties(env);
```

- The application defines a source using the Kinesis Consumer connector to read data from the input stream. The configuration of the input stream is defined in the `PropertyGroupId=InputStream0`. The name and Region of the stream are in the properties named `stream.name` and `aws.region` respectively. For simplicity, this source reads the records as a string.

```
private static FlinkKinesisConsumer<String> createSource(Properties
 inputProperties) {
    String inputStreamName = inputProperties.getProperty("stream.name");
    return new FlinkKinesisConsumer<>(inputStreamName, new SimpleStringSchema(),
 inputProperties);
}
...

public static void main(String[] args) throws Exception {
   ...
   SourceFunction<String> source =
 createSource(applicationParameters.get("InputStream0"));
   DataStream<String> input = env.addSource(source, "Kinesis Source");
   ...
}
```

- The application then defines a sink using the Kinesis Streams Sink connector to send data to the output stream. Output stream name and Region are defined in the `PropertyGroupId=OutputStream0`, similar to the input stream. The sink is connected directly to the internal `DataStream` that is getting data from the source. In a real application, you have some transformation between source and sink.

```
private static KinesisStreamsSink<String> createSink(Properties outputProperties) {
    String outputStreamName = outputProperties.getProperty("stream.name");
```

```
    return KinesisStreamsSink.<String>builder()
            .setKinesisClientProperties(outputProperties)
            .setSerializationSchema(new SimpleStringSchema())
            .setStreamName(outputStreamName)
            .setPartitionKeyGenerator(element ->
 String.valueOf(element.hashCode()))
            .build();
}
...
public static void main(String[] args) throws Exception {
   ...
   Sink<String> sink = createSink(applicationParameters.get("OutputStream0"));
   input.sinkTo(sink);
   ...
}
```

- Finally, you run the data flow that you just defined. This must be the last instruction of the `main()` method, after you defined all the operators the data flow requires:

```
env.execute("Flink streaming Java API skeleton");
```

Use the pom.xml file

The pom.xml file defines all dependencies required by the application and sets up the Maven Shade plugin to build the fat-jar that contains all dependencies required by Flink.

- Some dependencies have `provided` scope. These dependencies are automatically available when the application runs in Amazon Managed Service for Apache Flink. They are required to compile the application, or to run the application locally in your IDE. For more information, see Run your application locally. Make sure that you are using the same Flink version as the runtime you will use in Amazon Managed Service for Apache Flink.

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-clients</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-streaming-java</artifactId>
```

```
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
```

- You must add additional Apache Flink dependencies to the pom with the default scope, such as the Kinesis connector used by this application. For more information, see Use Apache Flink connectors. You can also add any additional Java dependencies required by your application.

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-kinesis</artifactId>
    <version>${aws.connector.version}</version>
</dependency>
```

- The Maven Java Compiler plugin makes sure that the code is compiled against Java 11, the JDK version currently supported by Apache Flink.
- The Maven Shade plugin packages the fat-jar, excluding some libraries that are provided by the runtime. It also specifies two transformers: `ServicesResourceTransformer` and `ManifestResourceTransformer`. The latter configures the class containing the `main` method to start the application. If you rename the main class, don't forget to update this transformer.
-

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    ...
        <transformer
 implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
            <mainClass>com.amazonaws.services.msf.BasicStreamingJob</mainClass>
        </transformer>
    ...
</plugin>
```

Write sample records to the input stream

In this section, you will send sample records to the stream for the application to process. You have two options for generating sample data, either using a Python script or the Kinesis Data Generator.

Generate sample data using a Python script

You can use a Python script to send sample records to the stream.

> **Note**
>
> To run this Python script, you must use Python 3.x and have the AWS SDK for Python (Boto) library installed.

To start sending test data to the Kinesis input stream:

1. Download the data generator `stock.py` Python script from the Data generator GitHub repository.
2. Run the `stock.py` script:

```
$ python stock.py
```

Keep the script running while you complete the rest of the tutorial. You can now run your Apache Flink application.

Generate sample data using Kinesis Data Generator

Alternatively to using the Python script, you can use Kinesis Data Generator, also available in a hosted version, to send random sample data to the stream. Kinesis Data Generator runs in your browser, and you don't need to install anything on your machine.

To set up and run Kinesis Data Generator:

1. Follow the instructions in the Kinesis Data Generator documentation to set up access to the tool. You will run an CloudFormation template that sets up a user and password.
2. Access Kinesis Data Generator through the URL generated by the CloudFormation template. You can find the URL in the **Output** tab after the CloudFormation template is completed.
3. Configure the data generator:
   - **Region:** Select the Region that you are using for this tutorial: us-east-1
   - **Stream/delivery stream:** Select the input stream that the application will use: `ExampleInputStream`

- **Records per second:** 100
- **Record template:** Copy and paste the following template:

```
{
  "event_time" : "{{date.now("YYYY-MM-DDTkk:mm:ss.SSSSS")}},
  "ticker" : "{{random.arrayElement(
        ["AAPL", "AMZN", "MSFT", "INTC", "TBV"]
    )}}",
  "price" : {{random.number(100)}}
}
```

4. Test the template: Choose **Test template** and verify that the generated record is similar to the following:

```
{ "event_time" : "2024-06-12T15:08:32.04800, "ticker" : "INTC", "price" : 7 }
```

5. Start the data generator: Choose **Select Send Data**.

Kinesis Data Generator is now sending data to the `ExampleInputStream`.

Run your application locally

You can run and debug your Flink application locally in your IDE.

> **Note**
>
> Before you continue, verify that the input and output streams are available. See Create two Amazon Kinesis data streams. Also, verify that you have permission to read and write from both streams. See Authenticate your AWS session.
> Setting up the local development environment requires Java 11 JDK, Apache Maven, and and IDE for Java development. Verify you meet the required prerequisites. See Fulfill the prerequisites for completing the exercises.

Import the Java project into your IDE

To start working on the application in your IDE, you must import it as a Java project.

The repository you cloned contains multiple examples. Each example is a separate project. For this tutorial, import the content in the `./java/GettingStarted` subdirectory into your IDE.

Insert the code as an existing Java project using Maven.

> **Note**
>
> The exact process to import a new Java project varies depending on the IDE you are using.

Check the local application configuration

When running locally, the application uses the configuration in the `application_properties.json` file in the resources folder of the project under `./src/main/resources`. You can edit this file to use different Kinesis stream names or Regions.

```
[
  {
    "PropertyGroupId": "InputStream0",
    "PropertyMap": {
      "stream.name": "ExampleInputStream",
      "flink.stream.initpos": "LATEST",
      "aws.region": "us-east-1"
    }
  },
  {
    "PropertyGroupId": "OutputStream0",
    "PropertyMap": {
      "stream.name": "ExampleOutputStream",
      "aws.region": "us-east-1"
    }
  }
]
```

Set up your IDE run configuration

You can run and debug the Flink application from your IDE directly by running the main class `com.amazonaws.services.msf.BasicStreamingJob`, as you would run any Java application. Before running the application, you must set up the Run configuration. The setup depends on the IDE you are using. For example, see Run/debug configurations in the IntelliJ IDEA documentation. In particular, you must set up the following:

1. **Add the `provided` dependencies to the classpath**. This is required to make sure that the dependencies with `provided` scope are passed to the application when running locally. Without this set up, the application displays a `class not found` error immediately.
2. **Pass the AWS credentials to access the Kinesis streams to the application**. The fastest way is to use AWS Toolkit for IntelliJ IDEA. Using this IDE plugin in the Run configuration, you can select a specific AWS profile. AWS authentication happens using this profile. You don't need to pass AWS credentials directly.
3. Verify that the IDE runs the application using **JDK 11**.

Run the application in your IDE

After you set up the Run configuration for the `BasicStreamingJob`, you can run or debug it like a regular Java application.

> **Note**
>
> You can't run the fat-jar generated by Maven directly with `java -jar ...` from the command line. This jar does not contain the Flink core dependencies required to run the application standalone.

When the application starts successfully, it logs some information about the standalone minicluster and the initialization of the connectors. This is followed by a number of INFO and some WARN logs that Flink normally emits when the application starts.

```
13:43:31,405 INFO  com.amazonaws.services.msf.BasicStreamingJob                [] -
 Loading application properties from 'flink-application-properties-dev.json'
13:43:31,549 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Flink Kinesis Consumer is going to read the following streams:
 ExampleInputStream,
13:43:31,676 INFO  org.apache.flink.runtime.taskexecutor.TaskExecutorResourceUtils []
 - The configuration option taskmanager.cpu.cores required for local execution is not
 set, setting it to the maximal possible value.
13:43:31,676 INFO  org.apache.flink.runtime.taskexecutor.TaskExecutorResourceUtils
 [] - The configuration option taskmanager.memory.task.heap.size required for local
 execution is not set, setting it to the maximal possible value.
13:43:31,676 INFO  org.apache.flink.runtime.taskexecutor.TaskExecutorResourceUtils []
 - The configuration option taskmanager.memory.task.off-heap.size required for local
 execution is not set, setting it to the maximal possible value.
```

```
13:43:31,676 INFO  org.apache.flink.runtime.taskexecutor.TaskExecutorResourceUtils []
 - The configuration option taskmanager.memory.network.min required for local execution
 is not set, setting it to its default value 64 mb.
13:43:31,676 INFO  org.apache.flink.runtime.taskexecutor.TaskExecutorResourceUtils []
 - The configuration option taskmanager.memory.network.max required for local execution
 is not set, setting it to its default value 64 mb.
13:43:31,676 INFO  org.apache.flink.runtime.taskexecutor.TaskExecutorResourceUtils [] -
 The configuration option taskmanager.memory.managed.size required for local execution
 is not set, setting it to its default value 128 mb.
13:43:31,677 INFO  org.apache.flink.runtime.minicluster.MiniCluster             [] -
 Starting Flink Mini Cluster
....
```

After the initialization is complete, the application doesn't emit any further log entries. **While data is flowing, no log is emitted.**

To verify if the application is correctly processing data, you can inspect the input and output Kinesis streams, as described in the following section.

> **Note**
>
> Not emitting logs about flowing data is the normal behavior for a Flink application. Emitting logs on every record might be convenient for debugging, but can add considerable overhead when running in production.

Observe input and output data in Kinesis streams

You can observe records sent to the input stream by the (generating sample Python) or the Kinesis Data Generator (link) by using the **Data Viewer** in the Amazon Kinesis console.

To observe records

1. Open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Verify that the Region is the same where you are running this tutorial, which is us-east-1 US East (N. Virginia) by default. Change the Region if it does not match.
3. Choose **Data Streams**.
4. Select the stream that you want to observe, either `ExampleInputStream` or `ExampleOutputStream`.

5. Choose the **Data viewer** tab.
6. Choose any **Shard**, keep **Latest** as **Starting position**, and then choose **Get records**. You might see a "No record found for this request" error. If so, choose **Retry getting records**. The newest records published to the stream display.
7. Choose the value in the Data column to inspect the content of the record in JSON format.

Stop your application running locally

Stop the application running in your IDE. The IDE usually provides a "stop" option. The exact location and method depends on the IDE you're using.

Compile and package your application code

In this section, you use Apache Maven to compile the Java code and package it into a JAR file. You can compile and package your code using the Maven command line tool or your IDE.

To compile and package using the Maven command line:

Move to the directory containing the Java GettingStarted project and run the following command:

```
$ mvn package
```

To compile and package using your IDE:

Run `mvn package` from your IDE Maven integration.

In both cases, the following JAR file is created: `target/amazon-msf-java-stream-app-1.0.jar`.

> **Note**
>
> Running a "build project" from your IDE might not create the JAR file.

Upload the application code JAR file

In this section, you upload the JAR file you created in the previous section to the Amazon Simple Storage Service (Amazon S3) bucket you created at the beginning of this tutorial. If you have not completed this step, see (link).

To upload the application code JAR file

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the bucket you previously created for the application code.
3. Choose **Upload**.
4. Choose **Add files**.
5. Navigate to the JAR file generated in the previous step: `target/amazon-msf-java-stream-app-1.0.jar`.
6. Choose **Upload** without changing any other settings.

> **Warning**
>
> Make sure that you select the correct JAR file in `<repo-dir>/java/GettingStarted/target/amazon-msf-java-stream-app-1.0.jar`.
> The `target` directory also contains other JAR files that you don't need to upload.

Create and configure 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. For this tutorial, you will use the console.

> **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 the application
- Edit the IAM policy
- Configure the application
- Run the application
- Observe the metrics of the running application

- Observe output data in Kinesis streams
- Stop the application

Create the application

To create the application

1. Sign in to the AWS Management Console, and open the Amazon MSF console at https://console.aws.amazon.com/flink.
2. Verify that the correct Region is selected: us-east-1 US East (N. Virginia)
3. Open the menu on the right and choose **Apache Flink applications** and then **Create streaming application**. Alternatively, choose **Create streaming application** in the Get started container of the initial page.
4. On the **Create streaming application** page:
   - **Choose a method to set up the stream processing application:** choose **Create from scratch**.
   - **Apache Flink configuration, Application Flink version:** choose **Apache Flink 1.20**.
5. Configure your application
   - **Application name:** enter **`MyApplication`**.
   - **Description:** enter **`My java test app`**.
   - **Access to application resources:** choose **Create / update IAM role `kinesis-analytics-MyApplication-us-east-1` with required policies**.
6. Configure your **Template for application settings**
   - **Templates:** choose **Development**.
7. Choose **Create streaming application** at the bottom of the page.

> **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-east-1`
- Role: `kinesisanalytics-MyApplication-us-east-1`

Amazon Managed Service for Apache Flink was formerly known as Kinesis Data Analytics. The name of the resources that are automatically created is prefixeed with `kinesis-analytics-` for backward compatibility.

Edit the IAM policy

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

To edit the policy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Policies**. Choose the **kinesis-analytics-service-MyApplication-us-east-1** policy that the console created for you in the previous section.
3. Choose **Edit** and then 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.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/kinesis-analytics-placeholder-s3-
object"
            ]
        },
```

```
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-east-1:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
```

```
            "Resource": "arn:aws:kinesis:us-east-1:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

5. Choose **Next** at the bottom of the page and then choose **Save changes**.

Configure the application

Edit the application configuration to set the application code artifact.

To edit the configuration

1. On the **MyApplication** page, choose **Configure**.
2. In the **Application code location** section:
   - For **Amazon S3 bucket**, select the bucket you previously created for the application code. Choose **Browse** and select the correct bucket, and then select **Choose**. Do not click on the bucket name.
   - For **Path to Amazon S3 object**, enter **`amazon-msf-java-stream-app-1.0.jar`**.
3. For **Access permissions**, choose **Create / update IAM role `kinesis-analytics-MyApplication-us-east-1` with required policies**.
4. In the **Runtime properties** section, add the following properties.
5. Choose **Add new item** and add each of the following parameters:

Group ID	Key	Value
`InputStream0`	`stream.name`	`ExampleInputStream`
`InputStream0`	`aws.region`	`us-east-1`
`OutputStream0`	`stream.name`	`ExampleOutputStream`
`OutputStream0`	`aws.region`	`us-east-1`

6. Do not modify any of the other sections.
7. Choose **Save changes**.

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 application is now configured and ready to run.

To run the application

1. On the console for Amazon Managed Service for Apache Flink, choose **My Application** and choose **Run**.
2. On the next page, the Application restore configuration page, choose **Run with latest snapshot** and then choose **Run**.

   The **Status** in **Application details** transitions from `Ready` to `Starting` and then to `Running` when the application has started.

When the application is in the `Running` status, you can now open the Flink dashboard.

To open the dashboard

1. Choose **Open Apache Flink dashboard**. The dashboard opens on a new page.
2. In the **Runing jobs** list, choose the single job that you can see.

   Note

   If you set the Runtime properties or edited the IAM policies incorrectly, the application status might turn into `Running`, but the Flink dashboard shows that the job is continuously restarting. This is a common failure scenario if the application is misconfigured or lacks permissions to access the external resources.

When this happens, check the **Exceptions** tab in the Flink dashboard to see the cause of the problem.

Observe the metrics of the running application

On the **MyApplication** page, in the **Amazon CloudWatch metrics** section, you can see some of the fundamental metrics from the running application.

To view the metrics

1. Next to the **Refresh** button, select **10 seconds** from the dropdown list.
2. When the application is running and healthy, you can see the **uptime** metric continuously increasing.
3. The **fullrestarts** metric should be zero. If it is increasing, the configuration might have issues. To investigate the issue, review the **Exceptions** tab on the Flink dashboard.
4. The **Number of failed checkpoints** metric should be zero in a healthy application.

Note

This dashboard displays a fixed set of metrics with a granularity of 5 minutes. You can create a custom application dashboard with any metrics in the CloudWatch dashboard.

Observe output data in Kinesis streams

Make sure you are still publishing data to the input, either using the Python script or the Kinesis Data Generator.

You can now observe the output of the application running on Managed Service for Apache Flink by using the Data Viewer in the https://console.aws.amazon.com/kinesis/, similarly to what you already did earlier.

To view the output

1. Open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Verify that the Region is the same as the one you are using to run this tutorial. By default, it is us-east-1US East (N. Virginia). Change the Region if necessary.

3. Choose **Data Streams**.
4. Select the stream that you want to observe. For this tutorial, use `ExampleOutputStream`.
5. Choose the **Data viewer** tab.
6. Select any **Shard**, keep **Latest** as **Starting position**, and then choose **Get records**. You might see a "no record found for this request" error. If so, choose **Retry getting records**. The newest records published to the stream display.
7. Select the value in the Data column to inspect the content of the record in JSON format.

Stop the application

To stop the applicatio, go to the console page of the Managed Service for Apache Flink application named `MyApplication`.

To stop the application

1. From the **Action** dropdown list, choose **Stop**.
2. The **Status** in **Application details** transitions from `Running` to `Stopping`, and then to `Ready` when the application is completely stopped.

> **Note**
>
> Don't forget to also stop sending data to the input stream from the Python script or the Kinesis Data Generator.

Next step

Clean up AWS resources

Clean up AWS resources

This section includes procedures for cleaning up AWS resources created in this Getting Started (DataStream API) 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
- Explore additional resources for Apache Flink

Delete your Managed Service for Apache Flink application

Use the following procedure 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. From the **Actions** dropdown list, choose **Delete** and then confirm the deletion.

Delete your Kinesis data streams

1. Sign in to the AWS Management Console, and open the Amazon MSF console at https://console.aws.amazon.com/flink..
2. Choose **Data streams**.
3. Select the two streams that you created, `ExampleInputStream` and `ExampleOutputStream`.
4. From the **Actions** dropdown list, choose **Delete**, and then confirm the deletion.

Delete your Amazon S3 objects and bucket

Use the following procedures to delete your Amazon S3 objects and bucket.

To delete the object from the S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Select the S3 bucket that you created for the application artifact.
3. Select the application artifact you uploaded, named `amazon-msf-java-stream-app-1.0.jar`.
4. Choose **Delete** and confirm the deletion.

To delete the S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Select the bucket that you created for the artifacts.
3. Choose **Delete** and confirm the deletion.

> **Note**
>
> The S3 bucket must be empty to delete it.

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-east-1** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-east-1** 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.

Explore additional resources for Apache Flink

Explore additional resources

Explore additional resources

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.

- **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.
- **Examples for creating and working with Managed Service for Apache Flink applications:** 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:** Offical introductory Apache Flink training that gets you started writing scalable streaming ETL, analytics, and event-driven applications.

Get 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 implementing an application in Java using the Table API and SQL. It demonstrates how to switch between different APIs within the same application, and 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
- Complete the required prerequisites
- Create and run a Managed Service for Apache Flink application
- Next step
- Clean up AWS resources
- Explore additional resources

Review the components of the Managed Service for Apache Flink application

> **Note**
>
> Managed Service for Apache Flink supports all Apache Flink APIs and potentially all JVM languages. Depending on the API you choose, the structure of the application and the implementation is slightly different. This tutorial covers the implementation of applications using the Table API and SQL, and the integration with the DataStream API, implemented in Java.

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

A typical Apache Flink application has the following components:

- **Runtime properties:** You can use *runtime properties* to pass configuration parameters to your application without modifying and republishing the code.
- **Sources:** The application consumes data from one or more *sources*. A source uses a connector to read data from and external system, such as a Kinesis data stream or an Amazon MSK topic. For development or testing, you can also have sources random[ly generate test data. For more information, see Add streaming data sources to Managed Service for Apache Flink. With SQL or Table API, sources are defined as *source tables*.
- **Transformations:** The application processes data through one or more transformations that can filter, enrich, or aggregate data. When using SQL or Table API, transformations are defined as *queries over tables* or *views*.
- **Sinks:** The application sends data to external systems through *sinks*. A sink uses a connector to send data to an external system, such as a Kinesis data stream, an Amazon MSK topic, an Amazon S3 bucket, or a relational database. You can also use a special connector to print the output for development purposes only. When using SQL or Table API, sinks are defined as *sink tables* where you will insert results. For more information, see Write data using sinks in Managed Service for Apache Flink.

Your application requires some **external dependencies**, such as Flink connectors your application uses, or potentially a Java library. To run in Amazon Managed Service for Apache Flink, you must package the application along with dependencies in a *fat-JAR* and upload it to an Amazon S3 bucket. You then create a Managed Service for Apache Flink application. You pass the code package location, along with other runtime configuration parameters. This tutorial demonstrates how to use Apache Maven to package the application and how to run the application locally in the IDE of your choice.

Complete the required prerequisites

Before starting this tutorial, complete the first two steps of the Get started with Amazon Managed Service for Apache Flink (DataStream API):

- Fulfill the prerequisites for completing the exercises
- 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 Kinesis data streams as a source and sink.

This section contains the following steps.

- Create dependent resources
- Set up your local development environment
- Download and examine the Apache Flink streaming Java code
- Run your application locally
- Observe the application writing data to an S3 bucket
- Stop your application running locally
- Compile and package your application code
- Upload the application code JAR file
- Create and configure the Managed Service for Apache Flink application

Create dependent resources

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

- An Amazon S3 bucket to store the application's code and to write the application output.

> **Note**
>
> This tutorial assumes that you are deploying your application in the us-east-1 Region. If you use another Region, you must adapt all steps accordingly.

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.

> **Note**
>
> Make sure that you create the bucket in the Region you use for this tutorial. The default for the tutorial is us-east-1.

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/<my-application>`.
- A log stream called `kinesis-analytics-log-stream`.

Set up your local development environment

For development and debugging, you can run the Apache Flink application on your machine, directly from your IDE of choice. Any Apache Flink dependencies are handled as normal Java dependencies using Maven.

> **Note**
>
> On your development machine, you must have Java JDK 11, Maven, and Git installed. We recommend that you use a development environment such as Eclipse Java Neon or IntelliJ IDEA. To verify that you meet all prerequisites, see Fulfill the prerequisites for completing the exercises. You **do not** need to install an Apache Flink cluster on your machine.

Authenticate your AWS session

The application uses Kinesis data streams to publish data. When running locally, you must have a valid AWS authenticated session with permissions to write to the Kinesis data stream. Use the following steps to authenticate your session:

1. If you don't have the AWS CLI and a named profile with valid credential configured, see Set up the AWS Command Line Interface (AWS CLI).

2. If your IDE has a plugin to integrate with AWS, you can use it to pass the credentials to the application running in the IDE. For more information, see AWS Toolkit for IntelliJ IDEA and AWS Toolkit for compiling the application or running Eclipse.

Download and examine the Apache Flink streaming Java code

The 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-managed-service-for-apache-flink-examples.git
```

2. Navigate to the `./java/GettingStartedTable` directory.

Review application components

The application is entirely implemented in the `com.amazonaws.services.msf.BasicTableJob` class. The `main()` method defines sources, transformations, and sinks. The execution is initiated by an execution statement at the end of this method.

> **Note**
>
> For an optimal developer experience, the application is designed to run without any code changes both on Amazon Managed Service for Apache Flink and locally, for development in your IDE.

- To read the runtime configuration so that it will work when running in Amazon Managed Service for Apache Flink and in your IDE, the application automatically detects if it's running standalone locally in the IDE. In that case, the application loads the runtime configuration differently:
  1. When the application detects that it's running in standalone mode in your IDE, form the `application_properties.json` file included in the **resources** folder of the project. The content of the file follows.

2. When the application runs in Amazon Managed Service for Apache Flink, the default behavior loads the application configuration from the runtime properties you will define in the Amazon Managed Service for Apache Flink application. See Create and configure the Managed Service for Apache Flink application.

```
private static Map<String, Properties>
 loadApplicationProperties(StreamExecutionEnvironment env) throws IOException {
    if (env instanceof LocalStreamEnvironment) {
        LOGGER.info("Loading application properties from '{}'",
 LOCAL_APPLICATION_PROPERTIES_RESOURCE);
        return KinesisAnalyticsRuntime.getApplicationProperties(
                BasicStreamingJob.class.getClassLoader()

 .getResource(LOCAL_APPLICATION_PROPERTIES_RESOURCE).getPath());
    } else {
        LOGGER.info("Loading application properties from Amazon Managed Service for
 Apache Flink");
        return KinesisAnalyticsRuntime.getApplicationProperties();
    }
}
```

- The `main()` method defines the application data flow and runs it.
  - Initializes the default streaming environments. In this example, we show how to create both the `StreamExecutionEnvironment` to use with the DataStream API, and the `StreamTableEnvironment` to use with SQL and the Table API. The two environment objects are two separate references to the same runtime environment, to use different APIs.

```
StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();
StreamTableEnvironment tableEnv = StreamTableEnvironment.create(env,
 EnvironmentSettings.newInstance().build());
```

  - Load the application configuration parameters. This will automatically load them from the correct place, depending on where the application is running:

```
Map<String, Properties> applicationParameters = loadApplicationProperties(env);
```

  - The FileSystem sink connector that the application uses to write results to Amazon S3 output files when Flink completes a checkpoint. You must enable checkpoints to write files to the

destination. When the application is running in Amazon Managed Service for Apache Flink, the application configuration controls the checkpoint and enables it by default. Conversely, when running locally, checkpoints are disabled by default. The application detects that it runs locally and configures checkpointing every 5,000 ms.

```
if (env instanceof LocalStreamEnvironment) {
   env.enableCheckpointing(5000);
}
```

- This application does not receive data from an actual external source. It generates random data to process through the DataGen connector. This connector is available for DataStream API, SQL, and Table API. To demonstrate the integration between APIs, the application uses the DataStram API version because it provides more flexibility. Each record is generated by a generator function called `StockPriceGeneratorFunction` in this case, where you can put custom logic.

```
DataGeneratorSource<StockPrice> source = new DataGeneratorSource<>(
        new StockPriceGeneratorFunction(),
        Long.MAX_VALUE,
        RateLimiterStrategy.perSecond(recordPerSecond),
        TypeInformation.of(StockPrice.class));
```

- In the DataStream API, records can have custom classes. Classes must follow specific rules so that Flink can use them as record. For more information, see Supported Data Types. In this example, the `StockPrice` class is a POJO.
- The source is then attached to the execution environment, generating a `DataStream` of `StockPrice`. This application doesn't use event-time semantics and doesn't generate a watermark. Run the DataGenerator source with a parallelism of 1, independent of the parallelism of the rest of the application.

```
DataStream<StockPrice> stockPrices = env.fromSource(
        source,
        WatermarkStrategy.noWatermarks(),
        "data-generator"
    ).setParallelism(1);
```

- What follows in the data processing flow is defined using the Table API and SQL. To do so, we convert the DataStream of StockPrices into a table. The schema of the table is automatically inferred from the `StockPrice` class.

```
Table stockPricesTable = tableEnv.fromDataStream(stockPrices);
```

- The following snippet of code shows how to define a view and a query using the programmatic Table API:

```
Table filteredStockPricesTable = stockPricesTable.
        select(
                $("eventTime").as("event_time"),
                $("ticker"),
                $("price"),
                dateFormat($("eventTime"), "yyyy-MM-dd").as("dt"),
                dateFormat($("eventTime"), "HH").as("hr")
        ).where($("price").isGreater(50));

tableEnv.createTemporaryView("filtered_stock_prices", filteredStockPricesTable);
```

- A sink table is defined to write the results to an Amazon S3 bucket as JSON files. To illustrate the difference with defining a view programmatically, with the Table API the sink table is defined using SQL.

```
tableEnv.executeSql("CREATE TABLE s3_sink (" +
            "eventTime TIMESTAMP(3)," +
            "ticker STRING," +
            "price DOUBLE," +
            "dt STRING," +
            "hr STRING" +
        ") PARTITIONED BY ( dt, hr ) WITH (" +
        "'connector' = 'filesystem'," +
        "'fmat' = 'json'," +
        "'path' = 's3a://"  + s3Path + "'" +
        ")");
```

- The last step of the is an `executeInsert()` that inserts the filtered stock prices view into the sink table. This method initiates the execution of the data flow we have defined so far.

```
filteredStockPricesTable.executeInsert("s3_sink");
```

Use the pom.xml file

The pom.xml file defines all dependencies required by the application and sets up the Maven Shade plugin to build the fat-jar that contains all dependencies required by Flink.

- Some dependencies have `provided` scope. These dependencies are automatically available when the application runs in Amazon Managed Service for Apache Flink. They are required for application or to the application locally in your IDE. For more information, see (update to TableAPI) Run your application locally. Make sure that you are using the same Flink version as the runtime you will use in Amazon Managed Service for Apache Flink. To use the TableAPI and SQL, you must include the `flink-table-planner-loader` and `flink-table-runtime-dependencies`, both with `provided` scope.

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-streaming-java</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-clients</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table-planner-loader</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table-runtime</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
```

- You must add additional Apache Flink dependencies to the pom with the default scope. For example, the DataGen connector, the FileSystem SQL connector, and the JSON format.

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-datagen</artifactId>
    <version>${flink.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-files</artifactId>
    <version>${flink.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-json</artifactId>
    <version>${flink.version}</version>
</dependency>
```

- To write to Amazon S3 when running locally, the S3 Hadoop File System is also included wit `provided` scope.

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-s3-fs-hadoop</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
```

- The Maven Java Compiler plugin makes sure that the code is compiled against Java 11, the JDK version currently supported by Apache Flink.
- The Maven Shade plugin packages the fat-jar, excluding some libraries that are provided by the runtime. It also specifies two transformers: `ServicesResourceTransformer` and `ManifestResourceTransformer`. The latter configures the class containing the `main` method to start the application. If you rename the main class, don't forget update this transformer.
-

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    ...
        <transformer
 implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
            <mainClass>com.amazonaws.services.msf.BasicStreamingJob</mainClass>
```

```
        </transformer>
    ...
</plugin>
```

Run your application locally

You can run and debug your Flink application locally in your IDE.

> **Note**
>
> Before you continue, verify that the input and output streams are available. See Create two Amazon Kinesis data streams. Also, verify that you have permission to read and write from both streams. See Authenticate your AWS session.
> Setting up the local development environment requires Java 11 JDK, Apache Maven, and an IDE for Java development. Verify you meet the required prerequisites. See Fulfill the prerequisites for completing the exercises.

Import the Java project into your IDE

To start working on the application in your IDE, you must import it as a Java project.

The repository you cloned contains multiple examples. Each example is a separate project. For this tutorial, import the content in the `./jave/GettingStartedTable` subdirectory into your IDE .

Insert the code as an existing Java project using Maven.

> **Note**
>
> The exact process to import a new Java project varies depending on the IDE you are using.

Modify the local application configuration

When running locally, the application uses the configuration in the `application_properties.json` file in the resources folder of the project under `./src/main/resources`. For this tutorial application, the configuration parameters are the name of the bucket and the path where the data will be written.

Edit the configuration and modify the name of the Amazon S3 bucket to match the bucket that you created at the beginning of this tutorial.

```
[
 {
 "PropertyGroupId": "bucket",
 "PropertyMap": {
 "name": "<bucket-name>",
 "path": "output"
 }
 }
]
```

> **Note**
>
> The configuration property `name` must contain only the bucket name, for example `my-bucket-name`. Don't include any prefix such as `s3://` or a trailing slash.
> If you modify the path, omit any leading or trailing slashes.

Set up your IDE run configuration

You can run and debug the Flink application from your IDE directly by running the main class `com.amazonaws.services.msf.BasicTableJob`, as you would run any Java application. Before running the application, you must set up the Run configuration. The setup depends on the IDE that you are using. For example, see Run/debug configurations in the IntelliJ IDEA documentation. In particular, you must set up the following:

1. **Add the `provided` dependencies to the classpath**. This is required to make sure that the dependencies with `provided` scope are passed to the application when running locally. Without this set up, the application displays a `class not found` error immediately.
2. **Pass the AWS credentials to access the Kinesis streams to the application**. The fastest way is to use AWS Toolkit for IntelliJ IDEA. Using this IDE plugin in the Run configuration, you can select a specific AWS profile. AWS authentication happens using this profile. You don't need to pass AWS credentials directly.
3. Verify that the IDE runs the application using **JDK 11**.

Run the application in your IDE

After you set up the Run configuration for the `BasicTableJob`, you can run or debug it like a regular Java application.

> **Note**
>
> You can't run the fat-jar generated by Maven directly with `java -jar ...` from the command line. This jar does not contain the Flink core dependencies required to run the application standalone.

When the application starts successfully, it logs some information about the standalone minicluster and the initialization of the connectors. This is followed by a number of INFO and some WARN logs that Flink normally emits when the application starts.

```
21:28:34,982 INFO  com.amazonaws.services.msf.BasicTableJob
               [] - Loading application properties from 'flink-application-properties-
dev.json'
21:28:35,149 INFO  com.amazonaws.services.msf.BasicTableJob
[] - s3Path is ExampleBucket/my-output-bucket
...
```

After the initialization is complete, the application doesn't emit any further log entries. **While data is flowing, no log is emitted.**

To verify if the application is correctly processing data, you can inspect the content of the output bucket, as described in the following section.

> **Note**
>
> Not emitting logs about flowing data is the normal behavior for a Flink application. Emitting logs on every record might be convenient for debugging, but can add considerable overhead when running in production.

Observe the application writing data to an S3 bucket

This example application generates random data internally and writes this data to the destination S3 bucket you configured. Unless you modified the default configuration path, the data will be

written to the `output` path followed by data and hour partitioning, in the format `./output/<yyyy-MM-dd>/<HH>`.

The FileSystem sink connector creates new files on the Flink checkpoint. When running locally, the application runs a checkpoint every 5 seconds (5,000 milliseconds), as specified in the code.

```
if (env instanceof LocalStreamEnvironment) {
   env.enableCheckpointing(5000);
}
```

To browse the S3 bucket and observe the file written by the application

1. 1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the bucket you previously created.
3. Navigate to the `output` path, and then to the date and hour folders that correspond to the current time in the UTC time zone.
4. Periodically refresh to observe new files appearing every 5 seconds.
5. Select and download one file to observe the content.

> **Note**
>
> By default, the files have no extensions. The content is formatted as JSON. You can open the files with any text editor to inspect the content.

Stop your application running locally

Stop the application running in your IDE. The IDE usually provides a "stop" option. The exact location and method depends on the IDE.

Compile and package your application code

In this section, you use Apache Maven to compile the Java code and package it into a JAR file. You can compile and package your code using the Maven command line tool or your IDE.

To compile and package using the Maven command line

Move to the directory that contains the Jave GettingStarted project and run the following command:

```
$ mvn package
```

To compile and package using your IDE

Run `mvn package` from your IDE Maven integration.

In both cases, the JAR file `target/amazon-msf-java-table-app-1.0.jar` is created.

> **Note**
>
> Running a *build project* from your IDE might not create the JAR file.

Upload the application code JAR file

In this section, you upload the JAR file you created in the previous section to the Amazon S3 bucket you created at the beginning of this tutorial. If you have done it yet, complete Create an Amazon S3 bucket.

To upload the application code

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the bucket you previously created for the application code.
3. Choose **Upload** field.
4. Choose **Add files**.
5. Navigate to the JAR file generated in the previous section: `target/amazon-msf-java-table-app-1.0.jar`.
6. Choose **Upload** without changing any other settings.

> **Warning**
>
> Make sure that you select the correct JAR file in `<repo-dir>/java/GettingStarted/target/`**`amazon/msf-java-table-app-1.0.jar`**.
> The target directory also contains other JAR files that you don't need to upload.

Create and configure the Managed Service for Apache Flink application

You can create and configure a Managed Service for Apache Flink application using either the console or the AWS CLI. For this tutorial, you will use the console.

> **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 must create these resources separately.

Create the application

1. Sign in to the AWS Management Console, and open the Amazon MSF console at https://console.aws.amazon.com/flink.
2. Verify that the correct Region is selected: US East (N. Virginia) us-east-1.
3. On the right menu, choose **Apache Flink applications** and then choose **Create streaming application**. Alternatively, choose **Create streaming application** in the **Get started** section of the initial page.
4. On the **Create streaming application** page, complete the following:
   - For **Choose a method to set up the stream processing application**, choose **Create from scratch**.
   - For **Apache Flink configuration, Application Flink version**, choose **Apache Flink 1.19**.
   - In the **Application configuration** section, complete the following:
     - For **Application name**, enter **`MyApplication`**.
     - For **Description**, enter **`My Java Table API test app`**.
     - For **Access to application resources**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-east-1 with required policies**.
   - In **Template for application settings**, complete the following:
     - For **Templates**, choose **Develoment**.
5. Choose **Create streaming 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-east-1`
- Role: `kinesisanalytics-MyApplication-us-east-1`

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-east-1** policy that the console created for you in the previous section.
3. Choose **Edit** and then choose the **JSON** tab.
4. Add the highlighted section of the following policy example to the policy. Replace the sample account ID (*012345678901*) with your account ID and *<bucket-name>* with the name of the S3 bucket that you created.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::amzn-s3-demo-bucket/kinesis-analytics-
placeholder-s3-object"
            ]
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "WriteOutputBucket",
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket2"
            ]
       }
```

```
    ]
}
```

5. Choose **Next** and then choose **Save changes**.

Configure the application

Edit the application to set the application code artifact.

To configure the application

1. On the **MyApplication** page, choose **Configure**.
2. In the **Aplication code location** section, choose **Configure**.
   - For **Amazon S3 bucket**, select the bucket you previously created for the application code. Choose **Browse** and select the correct bucket, and then choose **Choose**. Don't click on the bucket name.
   - For **Path to Amazon S3 object**, enter **`amazon-msf-java-table-app-1.0.jar`**.
3. For **Access permissions**, choose **Create / update IAM role `kinesis-analytics-MyApplication-us-east-1`**.
4. In the **Runtime properties** section, add the following properties.
5. Choose **Add new item** and add each of the following parameters:

Group ID	Key	Value
`bucket`	`name`	**`your-bucket-name`**
`bucket`	`path`	`output`

6. Don't modify any other setting.
7. Choose **Save changes**.

> **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 application is now configured and ready to run.

To run the application

1. Return to the console page in Amazon Managed Service for Apache Flink and choose **MyApplication**.
2. Choose **Run** to start the application.
3. On the **Application restore configuration**, choose **Run with latest snapshot**.
4. Choose **Run**.
5. The **Status** in **Application details** transitions from `Ready` to `Starting` and then to `Running` after the application has started.

When the application is in `Running` status, you can open the Flink dashboard.

To open the dashboard and view the job

1. Choose **Open Apache Flink dashbard**. The dashboard opens in a new page.
2. In the **Running Jobs** list, choose the single job you can see.

> **Note**
>
> If you set the runtime properties or edited the IAM policies incorrectly, the application status might change to `Running`, but the Flink dashboard shows the job continuously restarting. This is a common failure scenario when the application is misconfigured or lacks the permissions to access the external resources.
> When this happens, check the **Exceptions** tab in the Flink dashboard to investigate the cause of the problem.

Observe the metrics of the running application

On the **MyApplication** page, in the **Amazon CloudWatch metrics** section, you can see some of the fundamental metrics from the running application.

To view the metrics

1. Next to the **Refresh** button, select **10 seconds** from the dropdown list.
2. When the application is running and healthy, you can see the **uptime** metric continuously increasing.
3. The **fullrestarts** metric should be zero. If it is increasing, the configuration might have issues. Review the **Exceptions** tab on the Flink dashboard to investigate the issue.
4. The **Number of failed checkpoints** metric should be zero in a healthy application.

> **Note**
>
> This dashboard displays a fixed set of metrics with a granularity of 5 minutes. You can create a custom application dashboard with any metrics in the CloudWatch dashboard.

Observe the application writing data to the destination bucket

You can now observe the application running in Amazon Managed Service for Apache Flink writing files to Amazon S3.

To observe the files, follow the same process you used to check the files being written when the application was running locally. See Observe the application writing data to an S3 bucket.

Remember that the application writes new files on the Flink checkpoint. When running on Amazon Managed Service for Apache Flink, checkpoints are enabled by default and run every 60 seconds. The application creates new files approximately every 1 minute.

Stop the application

To stop the applicatio, go to the console page of the Managed Service for Apache Flink application named `MyApplication`.

To stop the application

1. From the **Action** dropdown list, choose **Stop**.

2. The **Status** in **Application details** transitions from `Running` to `Stopping`, and then to `Ready` when the application is completely stopped.

> **Note**
>
> Don't forget to also stop sending data to the input stream from the Python script or the Kinesis Data Generator.

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 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. From the **Actions** dropdown list, 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 the application object from the S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Select the S3 bucket that you created.
3. Select the application artifact that you uploaded named `amazon-msf-java-table-app-1.0.jar`, choose **Delete**, and then confirm the deletion.

To delete all output files written by the application

1. Choose the `output` folder.
2. Choose **Delete**.
3. Confirm that you want to permanently delete the content.

To delete the S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Select the S3 bucket you created.
3. Choose **Delete** and confirm the 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-east-1** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.

7. Choose the **kinesis-analytics-MyApplication-us-east-1** 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

Explore additional resources

Explore additional resources

Now that you've created and run a Managed Service for Apache Flink application that uses the Table API, see Explore additional resources in the Get started with Amazon Managed Service for Apache Flink (DataStream API).

Get 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

- Review the components of a Managed Service for Apache Flink application
- Fulfill the prerequisites
- Create and run a Managed Service for Apache Flink for Python application
- Clean up AWS resources

Review the components of a Managed Service for Apache Flink application

> **Note**
>
> Amazon Managed Service for Apache Flink supports all Apache Flink APIs. Depending on the API you choose, the structure of the application is slightly different. One popular approach when developing an Apache Flink application in Python is to define the application flow using SQL embedded in Python code. This is the approach that we follow in the following Gettgin Started tutorial.

To process data, your Managed Service for Apache Flink application uses a Python script to define the data flow that processes input and produces output using the Apache Flink runtime.

A typical 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.
- **Sources:** The application consumes data from one or more *sources*. A source uses a connector to read data from an external system such as a Kinesis data stream, or an Amazon MSK topic. You

can also use special connectors to generate data from within the application. When you use SQL, the application defines sources as *source tables*.

- **Transformations:** The application processes data by using one or more *transformations* that can filter, enrich, or aggregate data. When you use SQL, the application defines transformations as SQL queries.
- **Sinks:** The application sends data to external sources through *sinks*. A sink uses a connector to send data to an external system such as a Kinesis data stream, an Amazon MSK topic, an Amazon S3 bucket, or a relational database. You can also use a special connector to print the output for development purposes. When you use SQL, the application defines sinks as *sink tables* into which you insert results. For more information, see Write data using sinks in Managed Service for Apache Flink.

Your Python application might also require external dependencies, such as additional Python libraries or any Flink connector your application uses. When you package your application, you must include every dependency that your application requires. This tutorial demonstrates how to include connector dependencies and how to package the application for deployment on Amazon Managed Service for Apache Flink.

Fulfill the prerequisites

To complete this tutorial, you must have the following:

- **Python 3.11**, preferably using a standalone environment like VirtualEnv (venv), Conda, or Miniconda.
- Git client - install the Git client if you have not already.
- Java Development Kit (JDK) version 11 - install a Java JDK 11 and set the `JAVA_HOME` environment variable to point to your install location. If you don't have a JDK 11, you can use Amazon Corretto or any standard JDK of our choice.
  - To verify that you have the JDK correctly installed, run the following command. The output will be different if you are using a JDK other than Amazon Corretto 11. Make sure that the version is 11.x.

```
$ java --version

openjdk 11.0.23 2024-04-16 LTS
OpenJDK Runtime Environment Corretto-11.0.23.9.1 (build 11.0.23+9-LTS)
```

```
OpenJDK 64-Bit Server VM Corretto-11.0.23.9.1 (build 11.0.23+9-LTS, mixed mode)
```

- Apache Maven - install Apache Maven if you have not done so already. For more information, see Installing Apache Maven.
  - To test your Apache Maven installation, use the following command:

```
$ mvn -version
```

Note

Although your application is written in Python, Apache Flink runs in the Java Virtual Machine (JVM). It distributes most of the dependencies, such as the Kinesis connector, as JAR files. To manage these dependencies and to package the application in a ZIP file, use Apache Maven. This tutorial explains how to do so.

Warning

We recommend that you use Python 3.11 for local development. This is the same Python version used by Amazon Managed Service for Apache Flink with the Flink runtime 1.19. Installing the Python Flink library 1.19 on Python 3.12 might fail.
If you have another Python version installed by default on your machine, we recommend that you create a standalone environment such as VirtualEnv using Python 3.11.

IDE for local development

We recommend that you use a development environment such as PyCharm or Visual Studio Code to develop and compile your application.

Then, complete the first two steps of the Get started with Amazon Managed Service for Apache Flink (DataStream API):

- Set up an AWS account and create an administrator user
- 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 section, 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
- Set up your local development environment
- Download and examine the Apache Flink streaming Python code
- Manage JAR dependencies
- Write sample records to the input stream
- Run your application locally
- Observe input and output data in Kinesis streams
- Stop your application running locally
- Package your application code
- Upload the application package to an Amazon S3 bucket
- Create and configure 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.

> **Note**
>
> This tutorial assumes that you are deploying your application in the us-east-1 Region. If you use another Region, you must adapt all steps accordingly.

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`) in the same Region you will use to deploy your application (us-east-1 in this example). 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-east-1
```

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-east-1
```

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, for example by appending your login name.

> **Note**
>
> Make sure that you create the S3 bucket in the Region you use for this tutorial (us-east-1).

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/<my-application>`.
- A log stream called `kinesis-analytics-log-stream`.

Set up your local development environment

For development and debugging, you can run the Python Flink application on your machine. You can start the application from the command line with `python main.py` or in a Python IDE of your choice.

> **Note**
>
> On your development machine, you must have Python 3.10 or 3.11, Java 11, Apache Maven, and Git installed. We recommend that you use an IDE such as PyCharm or Visual Studio Code. To verify that you meet all prerequisites, see Fulfill the prerequisites for completing the exercises before you proceed.

Install the PyFlink library

To develop your application and run it locally, you must install the Flink Python library.

1. Create a standalone Python environment using VirtualEnv, Conda, or any similar Python tool.

2. Install the PyFlink library in that environment. Use the same Apache Flink runtime version that you will use in Amazon Managed Service for Apache Flink. Currently, the recommended runtime is 1.19.1.

```
$ pip install apache-flink==1.19.1
```

3. Make sure that the environment is active when you run your application. If you run the application in the IDE, make sure that the IDE is using the environment as runtime. The process depends on the IDE that you are using.

> **Note**
>
> You only need to install the PyFlink library. You **do not** need to install an Apache Flink cluster on your machine.

Authenticate your AWS session

The application uses Kinesis data streams to publish data. When running locally, you must have a valid AWS authenticated session with permissions to write to the Kinesis data stream. Use the following steps to authenticate your session:

1. If you don't have the AWS CLI and a named profile with valid credential configured, see Set up the AWS Command Line Interface (AWS CLI).
2. Verify that your AWS CLI is correctly configured and your users have permissions to write to the Kinesis data stream by publishing the following test record:

```
$ aws kinesis put-record --stream-name ExampleOutputStream --data TEST --partition-
key TEST
```

3. If your IDE has a plugin to integrate with AWS, you can use it to pass the credentials to the application running in the IDE. For more information, see AWS Toolkit for PyCharm, AWS Toolkit for Visual Studio Code, and AWS Toolkit for IntelliJ IDEA.

Download 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. 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 `./python/GettingStarted` directory.

Review application components

The application code is located in `main.py`. We use SQL embedded in Python to define the flow of the application.

> **Note**
>
> For an optimized developer experience, the application is designed to run without any code changes both on Amazon Managed Service for Apache Flink and locally, for development on your machine. The application uses the environment variable `IS_LOCAL = true` to detect when it is running locally. You must set the environment variable `IS_LOCAL = true` either on your shell or in the run configuration of your IDE.

- The application sets up the execution environment and reads the runtime configuration. To work both on Amazon Managed Service for Apache Flink and locally, the application checks the `IS_LOCAL` variable.
  - The following is the default behavior when the application runs in Amazon Managed Service for Apache Flink:
    1. Load dependencies packaged with the application. For more information, see (link)
    2. Load the configuration from the Runtime properties you define in the Amazon Managed Service for Apache Flink application. For more information, see (link)
  - When the application detects `IS_LOCAL = true` when you run your application locally:
    1. Loads external dependencies from the project.
    2. Loads the configuration from the `application_properties.json` file included in the project.

```
...
APPLICATION_PROPERTIES_FILE_PATH = "/etc/flink/application_properties.json"
...
```

```
is_local = (
    True if os.environ.get("IS_LOCAL") else False
)
...
if is_local:
    APPLICATION_PROPERTIES_FILE_PATH = "application_properties.json"
    CURRENT_DIR = os.path.dirname(os.path.realpath(__file__))
    table_env.get_config().get_configuration().set_string(
        "pipeline.jars",
        "file:///" + CURRENT_DIR + "/target/pyflink-dependencies.jar",
    )
```

- The application defines a source table with a CREATE TABLE statement, using the Kinesis Connector. This table reads data from the input Kinesis stream. The application takes the name of the stream, the Region, and initial position from the runtime configuration.

```
table_env.execute_sql(f"""
        CREATE TABLE prices (
                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' = '{input_stream_name}',
                'aws.region' = '{input_stream_region}',
                'format' = 'json',
                'json.timestamp-format.standard' = 'ISO-8601'
              ) """)
```

- The application also defines a sink table using the Kinesis Connector in this example. This tale sends data to the output Kinesis stream.

```
table_env.execute_sql(f"""
            CREATE TABLE output (
                ticker VARCHAR(6),
                price DOUBLE,
                event_time TIMESTAMP(3)
              )
              PARTITIONED BY (ticker)
              WITH (
```

```
                'connector' = 'kinesis',
                'stream' = '{output_stream_name}',
                'aws.region' = '{output_stream_region}',
                'sink.partitioner-field-delimiter' = ';',
                'sink.batch.max-size' = '100',
                'format' = 'json',
                'json.timestamp-format.standard' = 'ISO-8601'
              )""")
```

- Finally, the application executes a SQL that `INSERT INTO...` the sink table from the source table. In a more complex application, you likely have additional steps transforming data before writing to the sink.

```
table_result = table_env.execute_sql("""INSERT INTO output
        SELECT ticker, price, event_time FROM prices""")
```

- You must add another step at the end of the `main()` function to run the application locally:

```
if is_local:
    table_result.wait()
```

  Without this statement, the application terminates immediately when you run it locally. You must not execute this statement when you run your application in Amazon Managed Service for Apache Flink.

Manage JAR dependencies

A PyFlink application usually requires one or more connectors. The application in this tutorial uses the Kinesis Connector. Because Apache Flink runs in the Java JVM, connectors are distributed as JAR files, regardless if you implement your application in Python. You must package these dependencies with the application when you deploy it on Amazon Managed Service for Apache Flink.

In this example, we show how to use Apache Maven to fetch the dependencies and package the application to run on Managed Service for Apache Flink.

> **Note**
>
> There are alternative ways to fetch and package dependencies. This example demonstrates a method that works correctly with one or more connectors. It also lets you run the

application locally, for development, and on Managed Service for Apache Flink without code changes.

Use the pom.xml file

Apache Maven uses the `pom.xml` file to control dependencies and application packaging.

Any JAR dependencies are specified in the `pom.xml` file in the `<dependencies>...</dependencies>` block.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    ...
    <dependencies>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-kinesis</artifactId>
            <version>4.3.0-1.19</version>
        </dependency>
    </dependencies>
    ...
```

To find the correct artifact and version of connector to use, see Use Apache Flink connectors with Managed Service for Apache Flink. Make sure that you refer to the version of Apache Flink you are using. For this example, we use the Kinesis connector. For Apache Flink 1.19, the connector version is `4.3.0-1.19`.

> **Note**
>
> If you are using Apache Flink 1.19, there is no connector version released specifically for this version. Use the connectors released for 1.18.

Download and package dependencies

Use Maven to download the dependencies defined in the `pom.xml` file and package them for the Python Flink application.

1. Navigate to the directory that contains the Python Getting Started project called `python/GettingStarted`.
2. Run the following command:

```
$ mvn package
```

Maven creates a new file called `./target/pyflink-dependencies.jar`. When you are developing locally on your machine, the Python application looks for this file.

> **Note**
>
> If you forget to run this command, when you try to run your application, it will fail with the error: **Could not find any factory for identifier "kinesis**.

Write sample records to the input stream

In this section, you will send sample records to the stream for the application to process. You have two options for generating sample data, either using a Python script or the Kinesis Data Generator.

Generate sample data using a Python script

You can use a Python script to send sample records to the stream.

> **Note**
>
> To run this Python script, you must use Python 3.x and have the AWS SDK for Python (Boto) library installed.

To start sending test data to the Kinesis input stream:

1. Download the data generator `stock.py` Python script from the Data generator GitHub repository.
2. Run the `stock.py` script:

```
$ python stock.py
```

Keep the script running while you complete the rest of the tutorial. You can now run your Apache Flink application.

Generate sample data using Kinesis Data Generator

Alternatively to using the Python script, you can use Kinesis Data Generator, also available in a hosted version, to send random sample data to the stream. Kinesis Data Generator runs in your browser, and you don't need to install anything on your machine.

To set up and run Kinesis Data Generator:

1. Follow the instructions in the Kinesis Data Generator documentation to set up access to the tool. You will run an CloudFormation template that sets up a user and password.
2. Access Kinesis Data Generator through the URL generated by the CloudFormation template. You can find the URL in the **Output** tab after the CloudFormation template is completed.
3. Configure the data generator:
   - **Region:** Select the Region that you are using for this tutorial: us-east-1
   - **Stream/delivery stream:** Select the input stream that the application will use: `ExampleInputStream`
   - **Records per second:** 100
   - **Record template:** Copy and paste the following template:

```
{
  "event_time" : "{{date.now("YYYY-MM-DDTkk:mm:ss.SSSSS")}},
  "ticker" : "{{random.arrayElement(
        ["AAPL", "AMZN", "MSFT", "INTC", "TBV"]
    )}}",
  "price" : {{random.number(100)}}
}
```

4. Test the template: Choose **Test template** and verify that the generated record is similar to the following:

```
{ "event_time" : "2024-06-12T15:08:32.04800, "ticker" : "INTC", "price" : 7 }
```

5. Start the data generator: Choose **Select Send Data**.

Kinesis Data Generator is now sending data to the `ExampleInputStream`.

Run your application locally

You can test the application locally, running from the command line with `python main.py` or from your IDE.

To run your application locally, you must have the correct version of the PyFlink library installed as described in the previous section. For more information, see (link)

> **Note**
>
> Before you continue, verify that the input and output streams are available. See Create two Amazon Kinesis data streams. Also, verify that you have permission to read and write from both streams. See Authenticate your AWS session.

Import the Python project into your IDE

To start working on the application in your IDE, you must import it as a Python project.

The repository you cloned contains multiple examples. Each example is a separate project. For this tutorial, import the content in the `./python/GettingStarted` subdirectory into your IDE.

Import the code as an existing Python project.

> **Note**
>
> The exact process to import a new Python project varies depending on the IDE you are using.

Check the local application configuration

When running locally, the application uses the configuration in the `application_properties.json` file in the resources folder of the project under `./src/main/resources`. You can edit this file to use different Kinesis stream names or Regions.

```
[
  {
    "PropertyGroupId": "InputStream0",
    "PropertyMap": {
      "stream.name": "ExampleInputStream",
```

```
      "flink.stream.initpos": "LATEST",
      "aws.region": "us-east-1"
    }
  },
  {
    "PropertyGroupId": "OutputStream0",
    "PropertyMap": {
      "stream.name": "ExampleOutputStream",
      "aws.region": "us-east-1"
    }
  }
]
```

Run your Python application locally

You can run your application locally, either from the command line as a regular Python script, or from the IDE.

To run your application from the command line

1. Make sure that the standalone Python environment such as Conda or VirtualEnv where you installed the Python Flink library is currently active.
2. Make sure that you ran `mvn package` at least one time.
3. Set the `IS_LOCAL = true` environment variable:

   ```
   $ export IS_LOCAL=true
   ```

4. Run the application as a regular Python script.

   ```
   $python main.py
   ```

To run the application from within the IDE

1. Configure your IDE to run the `main.py` script with the following configuration:

   1. Use the standalone Python environment such as Conda or VirtualEnv where you installed the PyFlink library.
   2. Use the AWS credentials to access the input and output Kinesis data streams.
   3. Set `IS_LOCAL = true`.

2. The exact process to set the run configuration depends on your IDE and varies.
3. When you have set up your IDE, run the Python script and use the tooling provided by your IDE while the application is running.

Inspect application logs locally

When running locally, the application does not show any log in the console, aside from a few lines printed and displayed when the application starts. PyFlink writes logs to a file in the directory where the Python Flink library is installed. The application prints the location of the logs when it starts. You can also run the following command to find the logs:

```
$ python -c "import pyflink;import
 os;print(os.path.dirname(os.path.abspath(pyflink.__file__))+'/log')"
```

1. List the files in the logging directory. You usually find a single `.log` file.
2. Tail the file while the application is running: `tail -f <log-path>/<log-file>.log`.

Observe input and output data in Kinesis streams

You can observe records sent to the input stream by the (generating sample Python) or the Kinesis Data Generator (link) by using the **Data Viewer** in the Amazon Kinesis console.

To observe records:

Stop your application running locally

Stop the application running in your IDE. The IDE usually provides a "stop" option. The exact location and method depends on the IDE.

Package your application code

In this section, you use Apache Maven to package the application code and all required dependencies in a .zip file.

Run the Maven package command again:

```
$ mvn package
```

This command generates the file `target/managed-flink-pyflink-getting-started-1.0.0.zip`.

Upload the application package to an Amazon S3 bucket

In this section, you upload the .zip file you created in the previous section to the Amazon Simple Storage Service (Amazon S3) bucket you created at the beginning of this tutorial. If you have not completed this step, see (link).

To upload the application code JAR file

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose the bucket you previously created for the application code.
3. Choose **Upload**.
4. Choose **Add files**.
5. Navigate to the .zip file generated in the previous step: `target/managed-flink-pyflink-getting-started-1.0.0.zip`.
6. Choose **Upload** without changing any other settings.

Create and configure the Managed Service for Apache Flink application

You can create and configure a Managed Service for Apache Flink application using either the console or the AWS CLI. For this tutorial, we will use the console.

Create the application

1. Sign in to the AWS Management Console, and open the Amazon MSF console at https://console.aws.amazon.com/flink.
2. Verify that the correct Region is selected: US East (N. Virginia)us-east-1.
3. Open the right-side menu and choose **Apache Flink applications** and then **Create streaming application**. Alternatively, choose **Create streaming application** from the **Get started** section of the initial page.
4. On the **Create streaming applications** page:
   - For **Chose a method to set up the stream processing application**, choose **Create from scratch**.

- For **Apache Flink configuration, Application Flink version**, choose **Apache Flink 1.19**.
- For **Application configuration**:
  - For **Application name**, enter **`MyApplication`**.
  - For **Description**, enter **`My Python test app`**.
  - In **Access to application resources**, choose **Create / update IAM role kinesis-analytics-MyApplication-us-east-1 with required policies**.
- For **Template for applications settings**:
  - For **Templates**, choose **Development**.
- Choose **Create streaming 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`*
>
> Amazon Managed Service for Apache Flink was formerly known as *Kinesis Data Analytics*. The name of the resources that are generated automatically is prefixed with `kinesis-analytics` for backward compatibility.

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-east-1`** policy that the console created for you in the previous section.
3. Choose **Edit** and then 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.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/kinesis-analytics-placeholder-s3-
object"
            ]
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
```

```
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-east-1:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-east-1:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

5. Choose **Next** and then choose **Save changes**.

Configure the application

Edit the application configuration to set the application code artifact.

To configure the application

1. On the **MyApplication** page, choose **Configure**.
2. In the **Application code location** section:
   - For **Amazon S3 bucket**, select the bucket you previously created for the application code. Choose **Browse** and select the correct bucket, then choose **Choose**. Don't select on the bucket name.

- For **Path to Amazon S3 object**, enter **`managed-flink-pyflink-getting-started-1.0.0.zip`**.

3. For **Access permissions**, choose **Create / update IAM role `kinesis-analytics-MyApplication-us-east-1` with required policies**.
4. Move to **Runtime properties** and keep the default values for all other settings.
5. Choose **Add new item** and add each of the following parameters:

Group ID	Key	Value
`InputStream0`	`stream.name`	`ExampleInputStream`
`InputStream0`	`flink.stream.initpos`	`LATEST`
`InputStream0`	`aws.region`	`us-east-1`
`OutputStream0`	`stream.name`	`ExampleOutputStream`
`OutputStream0`	`aws.region`	`us-east-1`
`kinesis.analytics.flink.run.options`	`python`	`main.py`
`kinesis.analytics.flink.run.options`	`jarfile`	`lib/pyflink-dependencies.jar`

6. Do not modify any of the other sections and choose **Save changes**.

> **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 application is now configured and ready to run.

To run the application

1. On the console for Amazon Managed Service for Apache Flink, choose **My Application** and choose **Run**.
2. On the next page, the Application restore configuration page, choose **Run with latest snapshot** and then choose **Run**.

   The **Status** in **Application details** transitions from `Ready` to `Starting` and then to `Running` when the application has started.

When the application is in the `Running` status, you can now open the Flink dashboard.

To open the dashboard

1. Choose **Open Apache Flink dashboard**. The dashboard opens on a new page.
2. In the **Runing jobs** list, choose the single job that you can see.

> **Note**
>
> If you set the Runtime properties or edited the IAM policies incorrectly, the application status might turn into `Running`, but the Flink dashboard shows that the job is continuously restarting. This is a common failure scenario if the application is misconfigured or lacks permissions to access the external resources.
> When this happens, check the **Exceptions** tab in the Flink dashboard to see the cause of the problem.

Observe the metrics of the running application

On the **MyApplication** page, in the **Amazon CloudWatch metrics** section, you can see some of the fundamental metrics from the running application.

To view the metrics

1. Next to the **Refresh** button, select **10 seconds** from the dropdown list.

2. When the application is running and healthy, you can see the **uptime** metric continuously increasing.
3. The **fullrestarts** metric should be zero. If it is increasing, the configuration might have issues. To investigate the issue, review the **Exceptions** tab on the Flink dashboard.
4. The **Number of failed checkpoints** metric should be zero in a healthy application.

> **Note**
>
> This dashboard displays a fixed set of metrics with a granularity of 5 minutes. You can create a custom application dashboard with any metrics in the CloudWatch dashboard.

Observe output data in Kinesis streams

Make sure you are still publishing data to the input, either using the Python script or the Kinesis Data Generator.

You can now observe the output of the application running on Managed Service for Apache Flink by using the Data Viewer in the https://console.aws.amazon.com/kinesis/, similarly to what you already did earlier.

To view the output

1. Open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Verify that the Region is the same as the one you are using to run this tutorial. By default, it is us-east-1US East (N. Virginia). Change the Region if necessary.
3. Choose **Data Streams**.
4. Select the stream that you want to observe. For this tutorial, use `ExampleOutputStream`.
5. Choose the **Data viewer** tab.
6. Select any **Shard**, keep **Latest** as **Starting position**, and then choose **Get records**. You might see a "no record found for this request" error. If so, choose **Retry getting records**. The newest records published to the stream display.
7. Select the value in the Data column to inspect the content of the record in JSON format.

Stop the application

To stop the applicatio, go to the console page of the Managed Service for Apache Flink application named `MyApplication`.

To stop the application

1. From the **Action** dropdown list, choose **Stop**.
2. The **Status** in **Application details** transitions from `Running` to `Stopping`, and then to `Ready` when the application is completely stopped.

> **Note**
>
> Don't forget to also stop sending data to the input stream from the Python script or the Kinesis Data Generator.

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. From the **Actions** dropdown list, choose **Delete** and then confirm the deletion.

Delete your Kinesis data streams

1. Sign in to the AWS Management Console, and open the Amazon MSF console at https://console.aws.amazon.com/flink.
2. Choose **Data streams**.
3. Select the two streams that you created, `ExampleInputStream` and `ExampleOutputStream`.
4. From the **Actions** dropdown list, 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 the object from the S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Select the S3 bucket that you created for the application artifact.
3. Select the application artifact you uploaded, named `amazon-msf-java-stream-app-1.0.jar`.
4. Choose **Delete** and confirm the deletion.

To delete the S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Select the bucket that you created for the artifacts.
3. Choose **Delete** and confirm the deletion.

> **Note**
>
> The S3 bucket must be empty to delete it.

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-east-1** policy.
5. Choose **Policy Actions** and then choose **Delete**.
6. In the navigation bar, choose **Roles**.
7. Choose the **kinesis-analytics-MyApplication-us-east-1** 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.

Get 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 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. Sign in to the AWS Management Console, and open the Amazon MSF 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**.

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.

   JSON

```
{
    "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.

JSON

```
{
    "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-2:123456789012:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:123456789012:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:123456789012:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
```

```
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:123456789012:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:123456789012: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 kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

Update the application code

When you need to update your application code with a new version of your code package, you use the UpdateApplication 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": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

Use Apache Beam with Managed Service for Apache Flink applications

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 Use CloudFormation.

This topic contains the following sections:

- Limitations of Apache Flink runner with Managed Service for Apache Flink
- Apache Beam capabilities with Managed Service for Apache Flink
- Create an application using Apache Beam

Limitations of Apache Flink runner 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.**

Apache Beam capabilities with Managed Service for Apache Flink

Managed Service for Apache Flink supports the same Apache Beam capabilties 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.

Create 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 Use Apache Beam with Managed Service for Apache Flink applications.

> **Note**
>
> To set up required prerequisites for this exercise, first complete the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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\n** 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 Complete the required prerequisites in the Tutorial: Get started using the DataStream API in Managed Service for Apache Flink 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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

   JSON

```
{
    "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. Sign in to the AWS Management Console, and open the Amazon MSF 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.

Amazon Managed Service for Apache Flink workshop.

Develop Apache Flink applications locally before deploying to Managed Service for Apache Flink

This workshop demonstrates 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.

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

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 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 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.

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.

Clickstream Lab

Set up custom scaling using Application Auto Scaling

Two samples that show you how to automatically scale your Managed Service for Apache Flink applications using Application Auto Scaling. This lets you set up custom scaling policies and custom scaling attributes.

- 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.

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 CloudFormation templates where data flows through producers, streaming storage, consumers, and destinations.

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

Use practical utilities for Managed Service for Apache Flink

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 initiate 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

Examples for creating and working with Managed Service for Apache Flink applications

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
- Tutorial: Get started using the DataStream API in Managed Service for Apache Flink

> **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 for Managed Service for Apache Flink
- Python examples for Managed Service for Apache Flink
- Scala examples for Managed Service for Apache Flink

Java examples for Managed Service for Apache Flink

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.

Improve serialization performance defining custom TypeInfo

This example illustrates how to define custom TypeInfo on your record or state object to prevent serialization falling back to the less efficient Kryo serialization. This is required, for example, when your objects contain a `List` or `Map`. For more information, see Data Types & Serialization in the Apache Flink documentation. The example also shows how to test whether the serialization of your object falls back to the less efficient Kryo serialization.

Code example: CustomTypeInfo

Get 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

Get 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

Use 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

Use 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

Use 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

Use the Prometheus sink connector

This example demonstrates the use of the Prometheus sink connector to write time-series data to Prometheus.

Code example: PrometheusSink

Use 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

Use custom metrics

This example shows how to add custom metrics to your Flink application and send them to CloudWatch metrics.

Code example: CustomMetrics

Use Kafka Configuration Providers to fetch custom keystore and truststore for mTLS at runtime

This example illustrates how you can use Kafka Configuration Providers to set up a custom keystore and truststore with certificates for mTLS authentication for the Kafka connector. This technique

lets you load the required custom certificates from Amazon S3 and the secrets from AWS Secrets Manager when the application starts.

Code example: Kafka-mTLS-Keystore-ConfigProviders

Use Kafka Configuration Providers to fetch secrets for SASL/SCRAM authentication at runtime

This example illustrates how you can use Kafka Configuration Providers to fetch credentials from AWS Secrets Manager and download the truststore from Amazon S3 to set up SASL/SCRAM authentication on a Kafka connector. This technique lets you load the required custom certificates from Amazon S3 and the secrets from AWS Secrets Manager when the application starts.

Code example: Kafka-SASL_SSL-ConfigProviders

Use Kafka Configuration Providers to fetch custom keystore and truststore for mTLS at runtime with Table API/SQL

This example illustrates how you can use Kafka Configuration Providers in Table API /SQL to set up a custom keystore and truststore with certificates for mTLS authentication for the Kafka connector. This technique lets you load the required custom certificates from Amazon S3 and the secrets from AWS Secrets Manager when the application starts.

Code example: Kafka-mTLS-Keystore-Sql-ConfigProviders

Use Side Outputs to split a stream

This example illustrates how to leverage Side Outputs in Apache Flink for splitting a stream on specified attributes. This pattern is particularly useful when trying to implement the concept of Dead Letter Queues (DLQ) in streaming applications.

Code example: SideOutputs

Use Async I/O to call an external endpoint

This example illustrates how to use Apache Flink Async I/O to call an external endpoint in a non-blocking way, with retries on recoverable errors.

Code example: AsyncIO

Python examples for Managed Service for Apache Flink

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 as 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

Get 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

Add Python dependencies

This example shows how to add Python dependencies to your PyFlink application in the most general way. This method works for simple dependencies, like Boto3, or complex dependencies containing C libraries such as PyArrow.

Code example: PythonDependencies

Use 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

Use 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

Use 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

Use an Amazon Data Firehose sink

This example demonstrates how to send data to Amazon Data Firehose using SQL.

Code example: FirehoseSink

Scala examples for Managed Service for Apache Flink

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

Set up a 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
- Key management in Amazon Managed Service for Apache Flink
- Identity and Access Management for Amazon 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. Amazon MSF 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 Amazon MSF:

- 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).
- Amazon MSF 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

Amazon MSF encrypts all data in transit. Encryption in transit is enabled for all Amazon MSF applications and cannot be disabled.

Amazon MSF encrypts data in transit in the following scenarios:

- Data in transit from Kinesis Data Streams to Amazon MSF.
- Data in transit between internal components within Amazon MSF.
- Data in transit between Amazon MSF and Firehose.

Key management

In Amazon MSF, you can use either service managed or your own customer managed keys to encrypt data. For more information, see Key management in Amazon Managed Service for Apache Flink.

Key management in Amazon Managed Service for Apache Flink

In Amazon MSF, you can choose to use either AWS managed keys or your own customer managed keys (CMKs) to encrypt data. CMKs in AWS Key Management Service (AWS KMS) are encryption keys that you create, own, and manage yourself.

On this page

- Transparent encryption in Amazon MSF
- Customer managed keys in Amazon MSF
- Using customer managed keys in Amazon MSF
- Managing CMK using AWS Management Console
- Managing CMK using APIs

Transparent encryption in Amazon MSF

By default, Amazon MSF uses AWS owned keys (AOKs) to encrypt your data in ephemeral (running application storage) and durable (durable application storage) storage. This means all data subject to a Flink checkpoint or snapshot will be encrypted by default. AOKs are the default encryption method in Amazon MSF and no additional set up is required. To encrypt data in transit, Amazon MSF uses TLS and HTTP+SSL by default and requires no additional set up or configuration.

Customer managed keys in Amazon MSF

In Amazon MSF, CMK is a feature where you can encrypt your application's data with a key that you create, own, and manage on AWS KMS.

In this section

- What is encrypted with CMKs?
- What isn't encrypted with CMKs?
- Supported KMS key types

- KMS key permissions
- KMS encryption context and constraints
- Key rotation policy
- Least-privileged key policy statements
- Example AWS CloudTrail log entries

What is encrypted with CMKs?

In an Amazon MSF application, data subject to a Flink checkpoint or snapshot will be encrypted with a CMK you define for that application. Consquently, your CMK will encrypt data stored in either running application storage or durable application storage. The following sections describe the procedure to set up CMKs for your Amazon MSF applications.

> **Key rotation policy**
>
> Amazon MSF doesn't manage the key rotation policy for your CMKs. You're responsible for your own key rotation. This is because you create and maintain CMKs. For information about how to use your key rotation policy with CMK in Amazon MSF, see Key rotation policy.

What isn't encrypted with CMKs?

Sources and sinks

Encryption of data sources and sinks isn't managed by Amazon MSF. It's managed by your source or sink configuration or application connector configuration.

Retroactive application of encryption

CMK in Amazon MSF doesn't provide support to retroactively apply CMKs to an existing historic snapshot.

Log encryption

Currently, Amazon MSF doesn't support log encryption using KMS CMK for logs generated by your application code jar. You'll need to make sure logs don't contain data that require CMK encryption.

Encryption of data in transit

You can't use CMK to encrypt data in transit. By default, Amazon MSF encrypts all data in transit using TLS or HTTP and SSL.

Supported KMS key types

CMK in Amazon MSF supports symmetric keys.

KMS key permissions

CMK in Amazon MSF requires permission to perform the following KMS actions. These permissions are necessary to validate access, create CMK encrypted running application storage, and store CMK encrypted application state in durable application storage.

- **kms:DescribeKey**

  Grants permission to resolve a KMS key alias to the key ARN.
- **kms:Decrypt**

  Grants permission to accesses durable application state and provision running application storage.
- **kms:GenerateDataKey**

  Grants permission to store durable application state.
- **kms:GenerateDataKeyWithoutPlaintext**

  Grants permission to provision running application storage.
- **kms:CreateGrant**

  Grants permission to access running application storage.

KMS encryption context and constraints

CMK in Amazon MSF provides encryption context when accessing keys to read or write encrypted data, that is, `kms:EncryptionContext:aws:kinesisanalytics:arn`. In addition to encryption context, source contexts aws:SourceArn and aws:SourceAccount are provided when reading or writing durable application storage.

When creating grant to provision encrypted running application storage, Amazon MSF CMK creates grants with constraint type EncryptionContextSubset ensuring that only Decrypt operation is allowed through `"kms:GrantOperations": "Decrypt"`.

Key rotation policy

Amazon MSF doesn't manage the key rotation policy for your CMKs. You're responsible for your own key rotation because you create and maintain CMKs.

In KMS you use either automatic or manual key rotation to create new cryptographic material for your CMKs. For information about how to rotate your keys, see Rotate AWS KMS keys in the *AWS Key Management Service Developer Guide*.

When you rotate keys for CMKs in Amazon MSF, you must make sure that the operator (API caller) has permissions for both the previous and new key.

> **Note**
>
> An application can start from a snapshot which was encrypted with AOK after it's configured to use CMK. An application can also start from a snapshot which was encrypted with an older CMK. To start an application from a snapshot, the operator (API caller) must have permissions for both the old and new key.

In Amazon MSF, we recommend that you stop and restart your applications using CMK encryption. This ensures the new rotation master key is applied to all data in running application storage and durable application storage. If you don't stop and restart your application, the new key material will only be applied to durable application storage. Running application storage will continue to be encrypted using the previous rotation key material.

If you're changing the AWS KMS key ARN used for CMK you should use UpdateApplication in Amazon MSF. This will ensure your Flink application will restart as part of `UpdateApplication` applying the CMK changes.

> **Note**
>
> When you provide an alias or alias ARN, Amazon MSF resolves the alias to key ARN and stores the key ARN as the configured key for the application.

Least-privileged key policy statements

For information about key policy statements, see Create a KMS key policy and Application lifecycle operator (API caller) permissions.

Example AWS CloudTrail log entries

When Amazon MSF uses CMKs in AWS KMS, AWS CloudTrail automatically logs all AWS KMS API calls and related details. These logs contain information, such as AWS service making the request, KMS key ARN, API actions performed, and timestamps excluding the encrypted data. These logs provide essential audit trails for compliance, security monitoring, and troubleshooting by showing which services accessed your keys and when.

Example 1: AWS KMS Decrypt API call using an assumed role in Amazon MSF

The following CloudTrail log shows Amazon MSF performing a test kms:Decrypt operation on a CMK. Amazon MSF makes this request using an **`Operator`** role while using the CreateApplication API. The following log includes essential details, such as the target KMS key ARN, associated Amazon MSF application (*`MyCmkApplication`*), and timestamp of the operation.

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "REDACTED",
        "arn": "arn:aws:sts::123456789012:assumed-role/Operator/CmkTestingSession",
        "accountId": "123456789012",
        "accessKeyId": "REDACTED",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "REDACTED",
                "arn": "arn:aws:iam::123456789012:role/Operator",
                "accountId": "123456789012",
                "userName": "Operator"
            },
            "attributes": {
                "creationDate": "2025-08-07T13:29:28Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "kinesisanalytics.amazonaws.com"
    },
    "eventTime": "2025-08-07T13:45:45Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
```

```
    "sourceIPAddress": "kinesisanalytics.amazonaws.com",
    "userAgent": "kinesisanalytics.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is set.",
    "requestParameters": {
        "encryptionContext": {
            "aws:kinesisanalytics:arn": "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication"
        },
        "keyId": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "dryRun": true
    },
    "responseElements": null,
    "additionalEventData": {
        "keyMaterialId": "REDACTED"
    },
    "requestID": "56764d19-1eb1-48f1-8044-594aa7dd05c4",
    "eventID": "1371b402-f1dc-4c47-8f3a-1004e4803c5a",
    "readOnly": true,
    "resources": [
        {
            "accountId": "123456789012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```


Example 2: AWS KMS Decrypt API call in Amazon MSF with direct service authentication

The following CloudTrail log shows Amazon MSF performing a test kms:Decrypt operation on a CMK. Amazon MSF makes this request through direct AWS service-to-service authentication instead of assuming a role. The following log includes essential details, such as the target KMS key ARN, associated Amazon MSF application (*MyCmkApplication*), and a shared event ID of the operation.

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "kinesisanalytics.amazonaws.com"
    },
    "eventTime": "2025-08-07T13:45:45Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "kinesisanalytics.amazonaws.com",
    "userAgent": "kinesisanalytics.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is set.",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionContext": {
            "aws:kinesisanalytics:arn": "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication"
        },
        "dryRun": true
    },
    "responseElements": null,
    "additionalEventData": {
        "keyMaterialId": "REDACTED"
    },
    "requestID": "5fe45ada-7519-4608-be2f-5a9b8ddd62b2",
    "eventID": "6206b08f-ce04-3011-9ec2-55951d357b2c",
    "readOnly": true,
    "resources": [
        {
            "accountId": "123456789012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "Application-account-ID",
    "sharedEventID": "acbe4a39-ced9-4f53-9f3c-21ef7e89dc37",
```

```
    "eventCategory": "Management"
}
```

Using customer managed keys in Amazon MSF

You need to consider the following factors when establishing, managing, and operating Amazon MSF applications subject to a CMK policy.

Customer managed key

This is the key policy and key material. You'll need to create a key which is used to encrypt your application state in running application storage and durable application storage.

Application lifecycle operator (API caller)

This is the **`Operator`** IAM user or role. The Operator can be a human or an automation, such as a CI/CD pipeline that will create, deploy, and run the Amazon MSF application. The application lifecycle Operator can either be an IAM role or user.

> **Note**
>
> It's possible that the key administrator and operator are the same person. In this case, we recommend that you always use separate roles or users.

Application

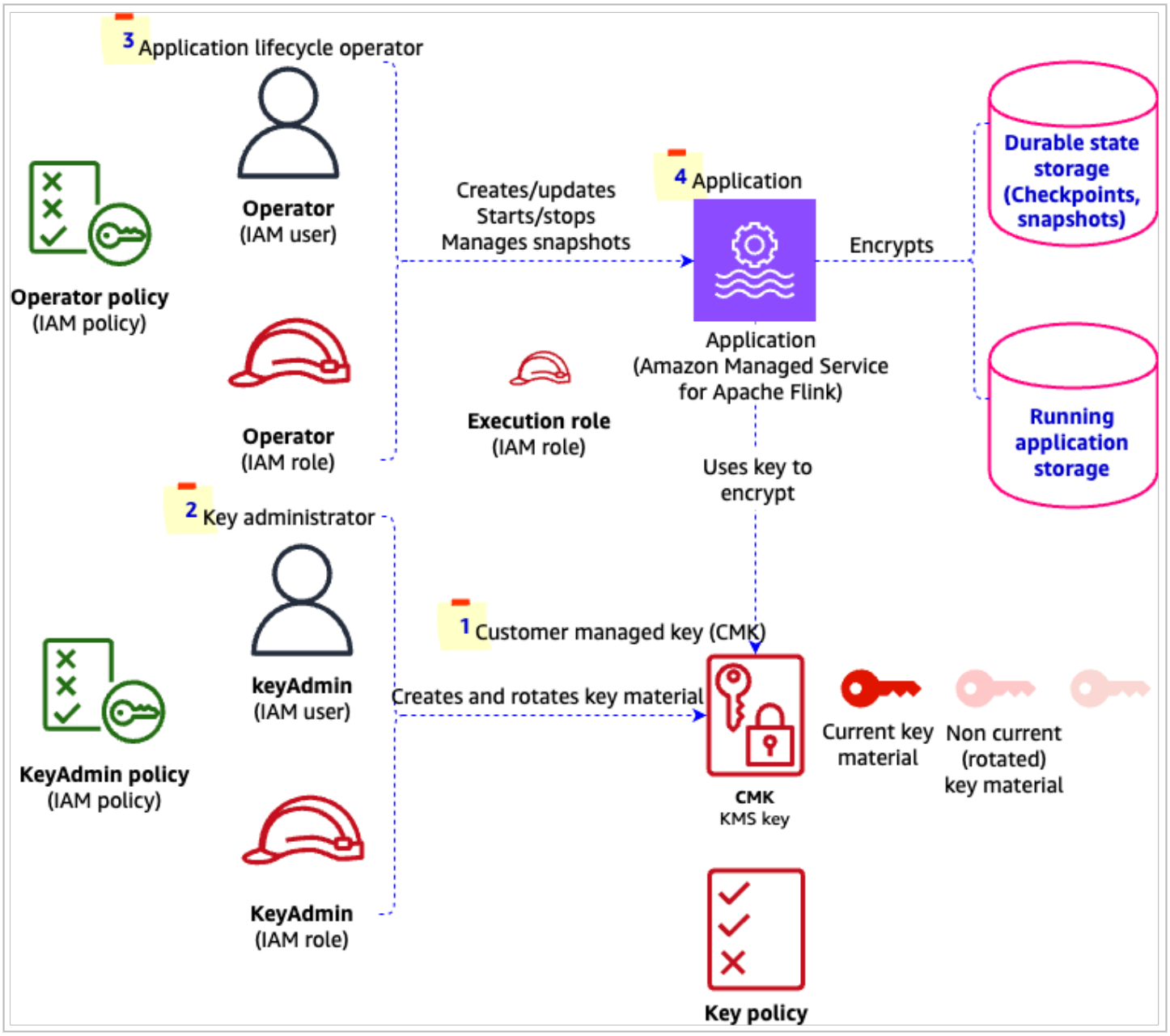
This is the Amazon MSF application you create. The application execution (IAM) role requires no changes to use CMK. For more information about IAM in Amazon MSF, see Identity and Access Management for Amazon Managed Service for Apache Flink.

> **Dependencies between policies**
>
> There are interdependencies between the key policy assigned to the CMK, and the IAM policy defining the permissions of the application lifecycle operator. You might want to create them in the following order:
>
> - Create the Operator IAM user or role without IAM policy defining permissions for CMK. The Operator creates the application with AOK.

- Create the key administrator with permissions to manage KMS keys. The key administrator creates the CMK. The key policy references to the Operator and administrator role ARNs, and to the application ARN. For more information, see Create a KMS key policy.
- Create an IAM policy for the Operator allowing to manage CMK for the application. For more information, see Application lifecycle operator (API caller) permissions. Attach the new IAM policy to the Operator. The Operator updates the application enabling CMK. For more information, see Update an existing application to use CMK.

If the application doesn't exist, create the application without CMK.

The following illustration shows how CMK is implemented in Amazon MSF.



1. **Customer managed key (CMK)**: Comprises key policy and key material.
2. **Key administrator**: The `KeyAdmin` IAM user or role.
3. **Application lifecycle operator (API caller)**: The operator IAM user or role.
4. **Application**: Has an execution (IAM) role attached.

Managing CMK using AWS Management Console

This topic describes how to create and update your KMS CMKs using the AWS Management Console. To follow the procedures described in this topic, you must have permission to manage the KMS key and the Amazon MSF application. The procedures in this topic use a permissive key policy, which is for demonstration and testing purposes only. We **don't recommend** using such a permissive key policy for production workloads. For production workloads, you can use the console, but in real-life scenarios, roles, permissions, and workflows are isolated.

Before you start, create a KMS key. For information about creating a KMS key, see Create a KMS key in the *AWS Key Management Service Developer Guide*.

Create and assign KMS keys

1. Sign in to the AWS Management Console, and open the Amazon MSF console at https://console.aws.amazon.com/flink.
2. On the **Streaming applications** page, choose **Create streaming application**.
3. For **Apache Flink version**, make sure that you choose **Apache Flink 1.20**.
4. For **Encryption**, choose **Use customer managed key**.
5. If you don't have a KMS key, choose **Create an AWS KMS key**, and create a KMS key. For information about how to create the key, see Using the AWS KMS console in the *AWS Key Management Service Developer Guide*.
6. If you don't have a KMS key, choose **Create an AWS KMS key**, and create a KMS key. For information about how to create the key using console, see Create a symmetric encryption KMS key.
7. Choose the key in the selector you want to use. Remember only the key with **Enabled** status is allowed.

Update an existing application to use CMK

1. Sign in to the AWS Management Console, and open the Amazon MSF console at https://console.aws.amazon.com/flink.
2. On the **Streaming applications** page, choose an application with Flink version 1.20.
3. Choose **Configure**.
4. For **Encryption**, choose **Use customer managed key**.

5. If you don't have a KMS key, choose **Create an AWS KMS key**, and create a KMS key. For information about how to create the key using console, see Create a symmetric encryption KMS key.
6. Choose the key in the selector you want to use. Remember only the key with **Enabled** status is allowed.

Switch from CMK to an AWS owned key

1. Sign in to the AWS Management Console, and open the Amazon MSF console at https://console.aws.amazon.com/flink.
2. On the **Streaming applications** page, choose an application with Flink version 1.20.
3. Choose **Configure**.
4. For **Encryption**, choose **Use AWS owned key**.

Managing CMK using APIs

This topic describes how to create, and update your KMS CMKs using Amazon MSF APIs. To follow the procedures described in this topic, you must have permission to manage the KMS key and the Amazon MSF application. The procedures in this topic use a permissive key policy, which is for demonstration and testing purposes only. We **don't recommend** using such a permissive key policy for production workloads. In real-life scenarios for production workloads, roles, permissions, and workflows are isolated.

On this page

- Create and assign KMS keys
- Update an existing application to use CMK
- Revert from CMK to AWS owned key

Create and assign KMS keys

Before you start, create a KMS key. For information about creating a KMS key, see Create a KMS key in the *AWS Key Management Service Developer Guide*.

In this section

- Create a KMS key policy

- Application lifecycle operator (API caller) permissions

Create a KMS key policy

To use CMK in Amazon MSF, you must add the following service principals to your key policy: `kinesisanalytics.amazonaws.com` and `infrastructure.kinesisanalytics.amazonaws.com`. Amazon MSF uses these service principals for validation and resource access. If you don't include these service principals, Amazon MSF rejects the request.

The following KMS key policy enables Amazon MSF to use a CMK for the application, *MyCmkApplication*. This policy grants necessary permissions to both the **Operator** role and Amazon MSF service principals, `kinesisanalytics.amazonaws.com` and `infrastructure.kinesisanalytics.amazonaws.com`, to perform the following operations:

- Describe the CMK
- Encrypt the application data
- Decrypt the application data
- Create grants for the key

The following example uses IAM roles. You can create the key policy for the KMS key using the following example as template, but make sure to do the following:

- Replace `arn:aws:iam::`*123456789012*`:role/Operator` with the **Operator** role. You must create the **Operator** role or user before creating the key policy. Failing to do this will cause the failure of your request.
- Replace `arn:aws:kinesisanalytics:us-east-1:`*123456789012*`:application/`*MyCmkApplication* with your application's ARN.
- Replace `kinesisanalytics.`*us-east-1*`.amazonaws.com` with a service value for the corresponding Region.
- Replace *123456789012* with your account idKey policy for CMK.
- Add additional policy statements to allow key administrators to administer the KMS key. Failing to do this will cause loss of access to manage the key.

The following key policy statements are large because they are intended to be explicit and show the conditions that each action requires.

```
{
    "Version":"2012-10-17",
    "Id": "MyMsfCmkApplicationKeyPolicy",
    "Statement": [
        {
            "Sid": "AllowOperatorToDescribeKey",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/Operator"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "kinesisanalytics.us-east-1.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AllowOperatorToConfigureAppToUseKeyForApplicationState",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/Operator"
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:kinesisanalytics:arn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication",
                    "kms:ViaService": "kinesisanalytics.us-east-1.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AllowOperatorToConfigureAppToCreateGrantForRunningState",
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": "arn:aws:iam::123456789012:role/Operator"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:kinesisanalytics:arn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication",
                    "kms:ViaService": "kinesisanalytics.us-east-1.amazonaws.com",
                    "kms:GrantConstraintType": "EncryptionContextSubset"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": "Decrypt"
                }
            }
        },
        {
            "Sid": "AllowMSFServiceToDescribeKey",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "kinesisanalytics.amazonaws.com",
                    "infrastructure.kinesisanalytics.amazonaws.com"
                ]
            },
            "Action": "kms:DescribeKey",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication",
                    "aws:SourceAccount": "123456789012"
                }
            }
        },
        {
            "Sid": "AllowMSFServiceToGenerateDataKeyForDurableState",
            "Effect": "Allow",
            "Principal": {
                "Service": "kinesisanalytics.amazonaws.com"
            },
            "Action": [
```

```
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication",
                    "kms:EncryptionContext:aws:kinesisanalytics:arn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication",
                    "aws:SourceAccount": "123456789012"
                }
            }
        },
        {
            "Sid": "AllowMSFServiceToDecryptForDurableState",
            "Effect": "Allow",
            "Principal": {
                "Service": "kinesisanalytics.amazonaws.com"
            },
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:kinesisanalytics:arn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication"
                }
            }
        },
        {
            "Sid": "AllowMSFServiceToUseKeyForRunningState",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "infrastructure.kinesisanalytics.amazonaws.com"
                ]
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKeyWithoutPlaintext"
```

```
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:kinesisanalytics:arn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication"
                }
            }
        },
        {
            "Sid": "AllowMSFServiceToCreateGrantForRunningState",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "infrastructure.kinesisanalytics.amazonaws.com"
                ]
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:kinesisanalytics:arn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication",
                    "kms:GrantConstraintType": "EncryptionContextSubset"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": "Decrypt"
                }
            }
        }
    ]
}
```

Application lifecycle operator (API caller) permissions

The following IAM policy ensures that the application lifecycle operator has the necessary permissions to assign a KMS key to the application, *MyCmkApplication*.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowMSFAPICalls",
            "Effect": "Allow",
            "Action": "kinesisanalytics:*",
            "Resource": "*"
        },
        {
            "Sid": "AllowPassingServiceExecutionRole",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::123456789012:role/MyCmkApplicationRole"
        },
        {
            "Sid": "AllowDescribeKey",
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "kinesisanalytics.us-east-1.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AllowMyCmkApplicationKeyOperationsForDurableState",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "kinesisanalytics.us-east-1.amazonaws.com",
                    "kms:EncryptionContext:aws:kinesisanalytics:arn":
```

```
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication"
                }
            }
        },
        {
            "Sid": "AllowMyCmkApplicationKeyOperationsForRunningState",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "kinesisanalytics.us-east-1.amazonaws.com",
                    "kms:EncryptionContext:aws:kinesisanalytics:arn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication"
                }
            }
        },
        {
            "Sid": "AllowMyCmkApplicationCreateGrantForRunningState",
            "Effect": "Allow",
            "Action": "kms:CreateGrant",
            "Resource": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": "Decrypt"
                },
                "StringEquals": {
                    "kms:ViaService": "kinesisanalytics.us-east-1.amazonaws.com",
                    "kms:EncryptionContext:aws:kinesisanalytics:arn":
                        "arn:aws:kinesisanalytics:us-
east-1:123456789012:application/MyCmkApplication",
                    "kms:GrantConstraintType": "EncryptionContextSubset"
                }
            }
        }
    ]
```

```
}
```

Update an existing application to use CMK

In Amazon MSF, you can apply a CMK policy to an existing application that uses AWS owned keys (AOKs).

By default, Amazon MSF uses AOKs to encrypt all your data in ephemeral (running application storage) and durable (durable application storage) storage. This means all data subject to a Flink checkpoint or snapshot are encrypted using AOKs by default. When you replace the AOK with a CMK, new checkpoints and snapshots are encrypted with CMK. However, historic snapshots will remain encrypted with the AOK.

To update an existing application to use CMK

1. Create a JSON file with the following configuration.

   Make sure that you replace the value of `CurrentApplicationVersionId` to the current version number of the application. You can get the current version number of your application, using DescribeApplication.

   In this JSON configuration, remember to replace the *sample* values with the actual values.

```
{
    "ApplicationName": "MyCmkApplication",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationEncryptionConfigurationUpdate": {
            "KeyTypeUpdate": "CUSTOMER_MANAGED_KEY",
            "KeyIdUpdate": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    }
}
```

2. Save this file. For example, save it with the name **`enable-cmk.json`**.
3. Run the update-application AWS CLI command as shown in the following example. In this command, provide the JSON configuration file you created in the previous steps as the file argument.

```
aws kinesisanalyticsv2 update-application \
    --cli-input-json file://enable-cmk.json
```

The preceding configuration is accepted to update the application for using CMK only if the following conditions are met:

- API caller has a policy statement that allows access to the key.
- Key policy has a policy statement that allows API caller access to the key.
- Key policy has a policy statement that allows the Amazon MSF service principal, for example, `kinesisanalytics.amazonaws.com` access to the key.

Revert from CMK to AWS owned key

To revert from CMK to an AOK

1. Create a JSON file with the following configuration.

   In this JSON configuration, remember to replace the *sample* values with the actual values.

   ```
   {
       "ApplicationName": "MyCmkApplication",
       "CurrentApplicationVersionId": 1,
       "ApplicationConfigurationUpdate": {
           "ApplicationEncryptionConfigurationUpdate": {
               "KeyTypeUpdate": "AWS_OWNED_KEY"
           }
       }
   }
   ```

2. Save this file. For example, save it with the name **disable-cmk.json**.
3. Run the update-application AWS CLI command as shown in the following example. In this command, provide the JSON configuration file you created in the previous steps as the file argument.

   ```
   aws kinesisanalyticsv2 update-application \
       --cli-input-json file://disable-cmk.json
   ```

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 based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting Amazon Managed Service for Apache Flink identity and access)
- **Service administrator** - determine user access and submit permission requests (see How Amazon Managed Service for Apache Flink works with IAM)
- **IAM administrator** - write policies to manage access (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 as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose 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 include 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. You must specify a principal in a resource-based policy.

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.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. 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

In Amazon MSF, you use IAM in the following different contexts:

- Application permissions: Control access by the application to external resources, such as Amazon S3, Amazon Kinesis Data Streams, or Amazon DynamoDB, that use IAM authentication.
- Application management and lifecycle control permissions: Control use of Amazon MSF API actions, such as CreateApplication, StartApplication, and UpdateApplication, which control the

application lifecycle. For a complete list of all Amazon MSF API actions that you can specify in the `Action` element of an IAM policy statement, see Actions defined by Amazon Kinesis Analytics V2 in the *Service Authorization Reference*.

Topics

- Application permissions
- Application management and lifecycle control permissions
- Identity-based policies for Managed Service for Apache Flink
- Resource-based policies within Managed Service for Apache Flink
- Access control lists (ACLs) in Managed Service for Apache Flink
- Service roles for Managed Service for Apache Flink
- Service-linked roles for Managed Service for Apache Flink

Application permissions

You control IAM permissions of an Amazon MSF application with the IAM role assigned to the application, as part of the application configuration. This IAM role determines application's permissions to access other services, such as Amazon S3, Kinesis Data Streams, or DynamoDB, which use IAM for authorization.

> **Warning**
>
> Changing the permissions for a service role might break Amazon MSF functionality. Make sure you don't remove permissions for the application to download the application code from the Amazon S3 bucket, and send logs to Amazon CloudWatch.

Assigning permissions to the application using resource-based policies isn't supported. You can't specify an Amazon MSF application as principal in a policy attached to the resource to be accessed.

Topics

- Permissions to access the application code and application logs
- Cross-service confused deputy prevention

Permissions to access the application code and application logs

Amazon MSF also uses the application IAM role to access the application code uploaded in an Amazon S3 bucket, and to write the application logs to Amazon CloudWatch Logs.

When you create or update the application using the AWS Management Console, choose **Create / update IAM role <role-name> with required policies** in the Application configuration, Amazon MSF automatically creates and modifies the IAM role assigning the required permissions to Amazon S3 and CloudWatch Logs.

If you create the IAM role manually or if you create and manage the application using automation tools, you must add the following permissions to the application IAM role.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/path-to-application-code"
            ]
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
```

```
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:/aws/kinesis-
analytics/application-name:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:/aws/kinesis-
analytics/application-name:log-stream:kinesis-analytics-log-stream"
            ]
        }
    ]
}
```

Cross-service confused deputy prevention

When an Amazon MSF application calls a different AWS service, you can provide more granular access permissions. For example, if an IAM role is reused across multiple applications, an application may get access to a resource it should not be have access to. This is known as the confused deputy problem. For information about how the accessed resource can restrict access to a specific Amazon MSF application, see Cross-service confused deputy prevention.

Application management and lifecycle control permissions

Actions to manage the application and its lifecycle, such as CreateApplication, StartApplication, and UpdateApplication, are controlled through identity-based policies associated to the resource performing the action, such as an IAM user, IAM group, or a resource such as AWS Lambda calling the Amazon MSF API.

> **Note**
>
> The API and SDK controlling Amazon MSF application lifecycle is called Amazon Kinesis Analytics V2, for backward compatibility reasons.

Assigning permissions for application lifecycle actions using resource-based policies attached to the Amazon MSF application isn't supported. The application IAM role isn't used to control access to the application lifecycle actions. You should not add application lifecycle permissions to the application role.

The following table lists the IAM features you can use with Amazon MSF application lifecycle actions.

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	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Cross-service principal permissions	Yes
Service roles	No
Service-linked roles	No

- For 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*.

- For information about the service-specific resources, actions, and condition context keys that you can use in IAM permission policies, see Actions, resources, and condition keys for Amazon Kinesis Analytics V2 in the *Service Authorization Reference*.

Topics

- Application lifecycle policy actions
- Application lifecycle policy resources
- Application lifecycle policy condition keys
- Attribute-based access control (ABAC) with Managed Service for Apache Flink
- Using temporary credentials
- Cross-service principal permissions

Application lifecycle policy actions

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. Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon MSF use the `kinesisanalytics` prefix before the action. Amazon MSF APIs and SDKs use the `Amazon Kinesis Analytics V2` prefix.

To specify multiple actions in a single statement, separate them with commas. The following example shows the syntax for specifying Amazon MSF policy actions.

```
"Action" : [
   "kinesisanalytics:action1",
   "kinesisanalytics:action2"
]
```

You can also specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action.

```
"Action": "kinesisanalytics:Describe*"
```

To see a complete list of all Amazon MSF API actions that you can specify in the `Action` element of an IAM policy statement, see Actions defined by Amazon Kinesis Analytics V2.

To view examples of Amazon MSF identity-based policies, see Identity-based policy examples.

Application lifecycle policy resources

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. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Permissions for Amazon MSF application lifecycle actions are defined for **each application**. The `Resource` JSON element in an IAM policy defines the Amazon MSF application to which the permissions apply.

You can assign permission to a single application by specifying the application ARN, or a group of application by using wildcards. The following example shows the syntax of the `Resource` element.

```
"Resouce" : "arn:partition:kinesisanalytics:Region:account:application/application-name
```

You can also assign permissions to control a subset of applications using wildcards. For example, you can assign permissions to control all applications whose name starts with a specific prefix.

```
"Resouce" : "arn:partition:kinesisanalytics:Region:account:application/application-
name-prefix*
```

Application lifecycle policy condition keys

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 specifies when statements execute based on defined criteria. 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. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

You can use condition keys to control permissions to Amazon MSF application lifecycle actions. 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.

Attribute-based access control (ABAC) with Managed Service for Apache Flink

Supports ABAC (tags in policies): Yes

Using condition keys, you can implement attribute-based access control (ABAC), which 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 Define permissions based on attributes with ABAC authorization.
- To view a tutorial with the steps for setting up ABAC, see IAM tutorial: Define permissions to access AWS resources based on tags.

Using temporary credentials

Supports temporary credentials: Yes

Amazon MSF application lifecycle actions support temporary credentials.

You're 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 Switch from a user to an IAM role (console).

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. We recommend 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

Supports forward access sessions (FAS): Yes

Amazon MSF application lifecycle actions support cross-service principal permissions.

When you use an IAM user or role to perform actions in AWS, you're considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. Forward access sessions (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.

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 Define custom IAM permissions with customer managed 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. 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

Amazon Managed Service for Apache Flink currently does not support resource-based access control.

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.

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 Create 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. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) 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 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 Validate policies with IAM Access Analyzer 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 Secure API access with MFA 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:ListGroupsForUser",
                "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**.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"kinesisanalytics.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"Account ID"
            },
            "ArnEquals":{
               "aws:SourceArn":"arn:aws:kinesisanalytics:us-
east-1:123456789012:application/my-app"
            }
         }
      }
   ]
}
```

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 CSPM – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

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 JobMananger 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 Implement 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 “Log Managed Service for Apache Flink API calls with 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 in Managed Service for Apache Flink
- Monitoring in Managed Service for Apache Flink
- Set up application logging in Managed Service for Apache Flink
- Analyze logs with CloudWatch Logs Insights
- Metrics and dimensions in Managed Service for Apache Flink
- Write custom messages to CloudWatch Logs
- Log Managed Service for Apache Flink API calls with AWS CloudTrail

Logging in Managed Service for Apache Flink

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 throughout 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.

Query 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 in Managed Service for Apache Flink

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 Use 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 considering 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.

Set up application logging in Managed Service for Apache Flink

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.

Set 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/kinesis-analytics/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 "Control application monitoring levels".

Set up CloudWatch logging using the CLI

To add a CloudWatch logging option using the AWS CLI, you complete 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.

Create 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.

Work 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.

Add 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:::amzn-s3-demo-bucket",
                            "FileKey": "myflink.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        }
    },
    "CloudWatchLoggingOptions": [{
      "LogStreamARN": "<Amazon Resource Name (ARN) of the CloudWatch log stream to add
 to the new application>"
 }]
}
```

Add 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>
}
```

Update 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>
}
```

Delete 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>
}
```

Set 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 “Control 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:::amzn-s3-demo-bucket",
          "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"
         }
      }
   }
}
```

Add 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.

JSON

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

Permissions policy

To grant permissions to an application to write log events to CloudWatch from a 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.

JSON

```
{
    "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:*"
            ]
        }
    ]
}
```

Control 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 "Implement application scaling".
- **Operator**: Metrics are scoped to each operator. For information about operators, see the section called "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 "Implement application 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.

Apply 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.

Perform 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 “Add 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.

Use CloudWatch Logs Insights

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 "Analyze logs with CloudWatch Logs Insights".

Analyze 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.

Review 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 Implement application 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 “Use 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:kinesisanalytics us-
west-2:012345678901:application\/YourApplication/
| filter @message like /AccessDenied/
| sort @timestamp desc
```

Analyze errors: Source or sink not found

The following CloudWatch Logs Insights query returns `ResourceNotFound` logs. `ResourceNotFound` logs result if a Kinesis source or sink is not found.

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalytics us-
west-2:012345678901:application\/YourApplication/
| filter @message like /ResourceNotFoundException/
| sort @timestamp desc
```

Analyze errors: Application task-related failures

The following CloudWatch Logs Insights query returns an application's task-related failure logs. These logs result if an application's status switches from `RUNNING` to `RESTARTING`.

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalytics us-
west-2:012345678901:application\/YourApplication/
```

```
| filter @message like /switched from RUNNING to RESTARTING/
| sort @timestamp desc
```

For applications using Apache Flink version 1.8.2 and prior, task-related failures will result in the application status switching from `RUNNING` to `FAILED` instead. When using Apache Flink 1.8.2 and prior, use the following query to search for application task-related failures:

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalytics us-
west-2:012345678901:application\/YourApplication/
| filter @message like /switched from RUNNING to FAILED/
| sort @timestamp desc
```

Metrics and dimensions in 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.

Flink 2.2 metric changes

Flink 2.2 introduces metric changes that may affect your monitoring and alarms. Review the following changes before upgrading:

- The `fullRestarts` metric has been removed. Use `numRestarts` instead.
- The `uptime` and `downtime` metrics are deprecated and will be removed in a future release. Migrate to the new state-specific metrics.
- The `bytesRequestedPerFetch` metric for Kinesis Data Streams connector 6.0.0 has been removed.

Application metrics

Metric	Unit	Description	Level	Usage Notes
`backPressuredTimeM`	Milliseconds	The time (in milliseconds) this task or	Task, Operator, Parallelism	*Available for Managed Service for

Metric	Unit	Description	Level	Usage Notes
sPerSecond*		operator is back pressured per second.		Apache Flink applications running Flink version 1.13 only. These metrics can be useful in identifying bottlenecks in an application.
busyTimeMsPerSecond*	Milliseconds	The time (in milliseconds) this task or operator is busy (neither idle nor back pressured) per second. Can be NaN, if the value could not be calculated.	Task, Operator, Parallelism	*Available for Managed Service for Apache Flink applications running Flink version 1.13 only. These metrics can be useful in identifying bottlenecks in an application.

Metric	Unit	Description	Level	Usage Notes
`cpuUtilization`	Percentage	Overall percentage of CPU 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 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
`containerCPUUtilization`	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	It is calculated per container as: *Total CPU time (in seconds) consumed by container * 100 / Container CPU limit (in CPUs/seconds)* 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 `containerCPUUtilization` metric gives you a

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	It is calculated per container as: *Container memory usage (bytes) * 100 / Container memory limit as per pod deployment spec (in bytes)* The `HeapMemoryUtilization` and `ManagedMemoryUtilzations` 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

Metric	Unit	Description	Level	Usage Notes
				RocksDB State Backend). The `containerMemoryUtilization` 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
`containerDiskUtilization`	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	It is calculated per container as: *Disk usage in bytes * 100 / Disk Limit for container in bytes* For containers, it represents utilization of the filesystem on which root volume of the container is set up.

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` [DEPRECATED]	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. **Deprecated in Flink 2.2.** Use `restartingTime`, `cancellingTime`, and/or `failingTime` instead.

Metric	Unit	Description	Level	Usage Notes
`failingTime`	Milliseconds	The time (in milliseconds) that the application has spent in a failing state. Use this metric to monitor application failures and trigger alerts.	Application, Flow	Available from Flink 2.2. Replaces part of the deprecated `downtime` metric.
`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	*Available for Managed Service for Apache Flink applications running Flink version 1.13 only. These metrics can be useful in identifying bottlenecks in an application.
`lastCheckpointSize`	Bytes	The total size of the last checkpoint	Application	You can use this metric to determine running application storage utilization. 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.

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	*Available for Managed Service for Apache Flink applications running Flink version 1.13 only. 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.

Metric	Unit	Description	Level	Usage Notes
managedMemoryTotal *	Bytes	The total amount of managed memory.	Application, Operator, Task, Parallelism	*Available for Managed Service for Apache Flink applications running Flink version 1.13 only. 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

Metric	Unit	Description	Level	Usage Notes
				RocksDB State Backend)
managedMemoryUtilization*	Percentage	Derived by managedMemoryUsed/ managedMemoryTotal	Application, Operator, Task, Parallelism	*Available for Managed Service for Apache Flink applications running Flink version 1.13 only. 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.

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	*To apply the SUM statistic over a period of time (second/minute): • Select the metric at the correct Level. If you're tracking the metric for an Operator, you need to select the correspon ding 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
				statistic over a period (second/minute) 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.

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	*To apply the SUM statistic over a period of time (second/minute): • 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
				statistic over a period (second/minute) The metric's Level specifies whether this metric measures the total number of records the entire application, a specific operator, or a specific task has received per second.

Metric	Unit	Description	Level	Usage Notes
numRecordsOut*	Count	The total number of records this application, operator or task has emitted.	Application, Operator, Task, Parallelism	*To apply the SUM statistic over a period of time (second/minute): • Select the metric at the correct Level. If you're tracking the metric for an Operator, you need to select the correspon ding 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
				statistic over a period (second/minute) 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.

Metric	Unit	Description	Level	Usage Notes	
numLateRecordsDropped*	Count	Application, Operator, Task, Parallelism		*To apply the SUM statistic over a period of time (second/minute): • Select the metric at the correct Level. If you're tracking the metric for an Operator, you need to select the correspon ding 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
				statistic over a period (second/minute) The number of records this operator or task has dropped due to arriving late.

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	*To apply the SUM statistic over a period of time (second/minute): • 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	
				statistic over a period (second/minute) 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.	
`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.
`threadsCount`	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.
`cancellingTime`	Milliseconds	The time (in milliseconds) that the application has spent in a cancelling state. Use this metric to monitor application cancellation operations.	Application, Flow	Available from Flink 2.2. Replaces part of the deprecated `downtime` metric.

Metric	Unit	Description	Level	Usage Notes
`restartingTime`	Milliseconds	The time (in milliseconds) that the application has spent in a restarting state. Use this metric to monitor application restart behavior.	Application, Flow	Available from Flink 2.2. Replaces part of the deprecated `downtime` metric.
`runningTime`	Milliseconds	The time (in milliseconds) that the application has been running without interruption. Replaces the deprecated `uptime` metric.	Application, Flow	Available from Flink 2.2. Use as a direct replacement for the deprecated `uptime` metric.

Metric	Unit	Description	Level	Usage Notes
`uptime` [DEPRECATED]	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. **Deprecated in Flink 2.2.** Use `runningTime` instead.
`jobmanagerFileDescriptorsMax`	Count	The maximum number of file descriptors available to the JobManager.	Application, Flow, Host	Use this metric to monitor file descriptor capacity.
`jobmanagerFileDescriptorsOpen`	Count	The current number of open file descriptors for the JobManager.	Application, Flow, Host	Use this metric to monitor file descriptor usage and detect potential resource exhaustion.

Metric	Unit	Description	Level	Usage Notes
taskmanagerFileDescriptorsMax	Count	The maximum number of file descriptors available to each TaskManager.	Application, Flow, Host, tm_id	Use this metric to monitor file descriptor capacity.
taskmanagerFileDescriptorsOpen	Count	The current number of open file descriptors for each TaskManager.	Application, Flow, Host, tm_id	Use this metric to monitor file descriptor usage and detect potential resource exhaustion.

Metric	Unit	Description	Level	Usage Notes
KPUs*	Count	The total number of KPUs used by the applicati on.	Application	*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. The KPU count includes the `orchestra tion` KPU. For more information, see Managed Service for Apache Flink Pricing.

Flink 2.2 metric migration guidance

Migration from fullRestarts: The `fullRestarts` metric has been removed in Flink 2.2. Use the `numRestarts` metric instead. The `numRestarts` metric provides equivalent functionality and can be used as a direct replacement in CloudWatch alarms without requiring threshold adjustments.

Migration from uptime: The `uptime` metric is deprecated in Flink 2.2 and will be removed in a future release. Use the `runningTime` metric instead. The `runningTime` metric

provides equivalent functionality and can be used as a direct replacement in CloudWatch alarms without requiring threshold adjustments.
Migration from downtime: The `downtime` metric is deprecated in Flink 2.2 and will be removed in a future release. Depending on what you want to monitor, use one or more of the following metrics:

- `restartingTime`: Monitor time spent restarting the application
- `cancellingTime`: Monitor time spent cancelling the application
- `failingTime`: Monitor time spent in a failing state

Kinesis Data Streams connector metrics

AWS emits all records for Kinesis Data Streams in addition to the following:

Metric	Unit	Description	Level	Usage Notes
`millisbehindLatest`	Milliseconds	The number of milliseconds the consumer is behind the head of the stream, indicating how far behind current time the consumer is.	Application (for Stream), Parallelism (for ShardId)	• A value of 0 indicates that record processing is caught up, and there are no new records to process at this moment. A particular shard's metric can be specified by stream name and shard id. • A value of -1 indicates that the service has not yet

Metric	Unit	Description	Level	Usage Notes
				reported a value for the metric.

> **Note**
>
> The `bytesRequestedPerFetch` metric has been removed in Flink AWS connector version 6.0.0 (the only connector version compatible with Flink 2.2). The only Kinesis Data Streams connector metric available in Flink 2.2 is `millisBehindLatest`.

Amazon MSK connector metrics

AWS emits all records for Amazon MSK in addition to the following:

Metric	Unit	Description	Level	Usage Notes
`currentoffsets`	N/A	The consumer's current read offset, for each partition. A particular partition's metric can be specified by topic name and partition id.	Application (for Topic), Parallelism (for 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

Metric	Unit	Description	Level	Usage Notes
				does not affect the integrity of Flink's checkpointed partition offsets.
`commitsSucceeded`	N/A	The total number of successful offset commits to Kafka, if offset committing and checkpointing are enabled.	Application, Operator, Task, Parallelism	
`committedoffsets`	N/A	The last successfully committed offsets to Kafka, for 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	

Metric	Unit	Description	Level	Usage Notes
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

Metric	Unit	Description	Prometheus name
zeppelinServerUptime	Seconds	The total time that the server has been up and running.	process_uptime_seconds

View 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
```

Set 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.

Use 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
- View examples for creating a mapping class
- View 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 according to Prometheus naming rules.
- 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.

View examples for creating a mapping class

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.

View custom metrics

Custom metrics for your application appear in the CloudWatch Metrics console in the **AWS/KinesisAnalytics** dashboard, under the **Application** metric group.

Use 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.

Review 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
`downtime` > 0	Average	0	A downtime greater than zero indicates that the applicati on has failed. If the value is larger than 0, the application is

Metric Expression	Statistic	Threshold	Description
			not processing any data. Recommend ed for all applicati ons. 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.

Metric Expression	Statistic	Threshold	Description
`RATE (numberOfFailedCheckpoints) > 0`	Average	0	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. 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

Metric Expression	Statistic	Threshold	Description
			processing the input data. For troublesh ooting, see Checkpoin ting is timing out.
`Operator.numRecordsOutPerSecond` < **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
`records_lag_max|millisbehindLatest` > **threshold**	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 `records_lag_max` metric for a Kafka source, or the `millisbehindLatest` 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
`lastCheckpointDuration` > **threshold**	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 considering monitoring the change rate with `RATE(lastCheckpointSize)` and `RATE(lastCheckpointDuration)` . If the `lastCheckpointDuration` 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
`lastCheckpointSize` > **threshold**	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 considering monitoring the change rate with `RATE(lastCheckpointSize)` and `RATE(lastCheckpointDuration)` . If the `lastCheckpointSize` continuously increases, rising above this threshold can indicate that

Metric Expression	Statistic	Threshold	Description
			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.
`heapMemoryUtilization` > **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 `heapMemoryUtilization` size during normal conditions, with a recommended value of 90 percent.	You can use this metric to monitor the maximum memory utilization of task managers across the application. If the application reaches this threshold, you need to provision more resources . You do this by enabling automatic scaling or increasing the application parallelism. For more information about increasing resources , see Implement application scaling.

Metric Expression	Statistic	Threshold	Description
`cpuUtilization` > **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 `cpuUtilization` 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 Implement application scaling.
`threadsCount` > **threshold**	Maximum	The maximum expected `threadsCount` 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
`(oldGarbageCollectionTime * 100)/60_000 over 1 min period')` > **threshold**	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.
`RATE(oldGarbageCollectionCount)` > **threshold**	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.

Write 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"
}
```

Log 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.

Understand 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": "kinesisanlaytics.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": "kinesisanlaytics.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"
        }
    ]
}
```

Tune 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

- Troubleshoot performance issues
- Use performance best practices
- Monitor performance

Troubleshoot performance issues

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.

Understand 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 Control 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 Implement application 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 KPUs 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 “Analyze logs with CloudWatch Logs Insights”.

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 returing `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 `CheckpointDuration`metric spiking or gradually increasing. This condition may also lead to task manager or job manager failures.

Use 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 KPUs 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 KPUs. 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 Implement application 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 Use 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 Review Managed Service for Apache Flink 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.

Monitor performance

This section describes tools for monitoring an application's performance.

Monitor performance 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.

Monitor performance 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 Use CloudWatch Alarms with Amazon Managed Service for Apache Flink.

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 now plan to end support for these versions in Amazon Managed Service for Apache Flink. From **November 5, 2024**, you will not be able to create new applications for these Flink versions. You can continue running existing applications at this time.
> For all Regions with exception of the China Regions and the AWS GovCloud (US) Regions, from **February 5, 2025**, you will no longer be able to create, start, or run applications using these versions of Apache Flink in Amazon Managed Service for Apache Flink.
> For the China Regions and the AWS GovCloud (US) Regions, from **March 19, 2025**, you will no longer be able to create, start, or run applications using these versions of Apache Flink in Amazon Managed Service for Apache Flink.
> You can upgrade your applications statefully using the in-place version upgrades feature in Managed Service for Apache Flink. For more information, see Use 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 100 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. For more information, see Manage application backups using snapshots.

- 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

Manage maintenance tasks for Managed Service for Apache Flink

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. A maintenance window for a Managed Service for Apache Flink application is a time window of 8 hours during which Managed Service for Apache Flink performs application maintenance activities on an application. The maintenance might begin on different days for different AWS Regions as scheduled by the service team. Consult the table in the following section for maintenance time windows.

As part of the maintenance procedure, your Managed Service for Apache Flink application will be restarted. This causes a downtime of 10 to 30 seconds during the application's maintenance window. The actual downtime duration depends on the application state, size, and snapshot/checkpoint recency. For information on how to minimize the impact of this downtime, see the section called “Fault tolerance: checkpoints and savepoints”. You can find out if Managed Service for Apache Flink has performed a maintenance action on your application using the `ListApplicationOperations` API. For more information, see Identify when maintenance has ocurred on your application.

Maintenance time windows in AWS Regions

AWS 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

AWS Region	Maintenance time window
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
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

AWS Region	Maintenance time window
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
Israel (Tel Aviv)	20:00–04:00 UTC

Choose a maintenance window

Managed Service for Apache Flink notifies you about upcoming planned maintenance events through email and AWS Health notifications. In Managed Service for Apache Flink, you can change the time of the day during which maintenance begins by using the `UpdateApplicationMaintenanceConfiguration` API and updating your maintenance window configuration. For more information, see UpdateApplicationMaintenanceConfiguration. Managed Service for Apache Flink uses the updated maintenance configuration the next time it schedules maintenance for the application. If you invoke this operation after the service has already scheduled maintenance, the service applies the configuration update the next time it schedules maintenance for the application.

> **Note**
>
> To provide the highest possible security posture, Managed Service for Apache Flink does not support any exception to opt out of maintenance, pause maintenance, or perform maintenance on specific days.

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"
}
```

Achieve production readiness for your Managed Service for Apache Flink applications

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-test your 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.

Define 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 <= 128
- `MIN(nextPowerOfTwo(parallelism + (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.

Maintain best practices for Managed Service for Apache Flink applications

This section contains information and recommendations for developing a stable, performant Managed Service for Apache Flink applications.

Topics

- Minimize the size of the uber JAR
- 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

Minimize the size of the uber JAR

Java/Scala application must be packaged in an uber (super/fat) JAR and include all the additional required dependencies that are not already provided by the runtime. However, the size of the uber JAR affects the application start and restart times and may cause the JAR to exceed the limit of 512 MB.

To optimize the deployment time, your uber JAR should **not** include the following:

- **Any dependencies provided by the runtime** as illustrated in the following example. They should have `provided` scope in the POM file or `compileOnly` in your Gradle configuration.

- **Any dependencies used for testing only**, for example JUnit or Mockito. They should have `test` scope in the POM file or `testImplementation` in your Gradle configuration.
- **Any dependencies not actually used by your application**.
- **Any static data or metadata required by your application.** Static data should be loaded by the application at runtime, for example from a datastore or from Amazon S3.
- See this POM example file for details on the preceding configuration settings.

Provided dependencies

The Managed Service for Apache Flink runtime provides a number of dependencies. These dependencies should not be included in the fat JAR and must have `provided` scope in the POM file or be explicitly excluded in the `maven-shade-plugin` configuration. Any of these dependencies included in the fat JAR is ignored at runtime, but increases the size of the JAR adding overhead during the deployment.

Dependencies provided by the runtime, in runtime versions 1.18, 1.19, and 1.20:

- `org.apache.flink:flink-core`
- `org.apache.flink:flink-java`
- `org.apache.flink:flink-streaming-java`
- `org.apache.flink:flink-scala_2.12`
- `org.apache.flink:flink-table-runtime`
- `org.apache.flink:flink-table-planner-loader`
- `org.apache.flink:flink-json`
- `org.apache.flink:flink-connector-base`
- `org.apache.flink:flink-connector-files`
- `org.apache.flink:flink-clients`
- `org.apache.flink:flink-runtime-web`
- `org.apache.flink:flink-metrics-code`
- `org.apache.flink:flink-table-api-java`
- `org.apache.flink:flink-table-api-bridge-base`
- `org.apache.flink:flink-table-api-java-bridge`
- `org.apache.logging.log4j:log4j-slf4j-impl`

- `org.apache.logging.log4j:log4j-api`
- `org.apache.logging.log4j:log4j-core`
- `org.apache.logging.log4j:log4j-1.2-api`

Additionally, the runtime provides the library that is used to fetch application runtime properties in Managed Service for Apache Flink, `com.amazonaws:aws-kinesisanalytics-runtime:1.2.0`.

All dependencies provided by the runtime must use the following recommendations to **not** include them in the uber JAR:

- In Maven (`pom.xml`) and SBT (`build.sbt`), use `provided` scope.
- In Gradle (`build.gradle`), use `compileOnly` configuration.

Any provided dependency accidentally included in the uber JAR will be ignored at runtime because of Apache Flink's parent-first class loading. For more information, see parent-first-patterns in the Apache Flink documentation.

Connectors

Most of the connectors, except the FileSystem connector, that are not included in the runtime must be included in the POM file with the default scope (`compile`).

Other recommendations

As a rule, your Apache Flink uber JAR provided to Managed Service for Apache Flink should contain the minimum code required to run the application. Including dependencies that include the source classes, test datasets, or bootstrapping state should not be included in this jar. If static resources need to be pulled in at runtime, separate this concern into a resource such as Amazon S3. Examples of this include state bootstraps or an inference model.

Take some time to consider your deep dependency tree and remove non-runtime dependencies.

Although Managed Service for Apache Flink supports 512MB jar sizes, this should be seen as the exception to the rule. Apache Flink currently supports ~104MB jar sizes through its default configuration, and that should be the maximum target size of a jar needed.

Fault tolerance: checkpoints and savepoints

Use checkpoints and savepoints to implement fault tolerance in your Managed Service for Apache Flink application. Keep the following in mind when developing and maintaining your application:

- We recommend that you keep checkpointing enabled for your application. Checkpointing provides fault tolerance for your application during scheduled maintenance, and also for unexpected failures due to service issues, application dependency failures, and other issues. For information about scheduled maintenance, see *Manage maintenance tasks for Managed Service for Apache Flink*.
- 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 you set your application to 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.

  For information about monitoring application downtime, see Metrics and dimensions in Managed Service for Apache Flink.

For more information about implementing fault tolerance, see Implement 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 Implement application 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.setParalleism(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, consider passing the operator parallelism as application configuration parameter to change it without changing the code, 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 the processing of data.
- Create CloudWatch alarms to notify you when your application is not running properly. For more information, see Use CloudWatch Alarms with Amazon Managed Service for Apache Flink

For more information about implementing logging, see ???.

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 permissions to that role. The running Flink application can then select temporary credentials with the respective permissions from the environment. In case authentication is needed for a service that is not natively integrated with IAM, for example, 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 (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 optimal 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 built-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.

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. The idle subtask therefore no longer blocks the advancement of event time in downstream operators.

Depending on the shard assigner you use, some workers might not be assigned any Kinesis shards. In that case, these workers will manifest the idle source behavior even if all Kinesis shards continuously deliver event data. You can mitigate this risk by using `uniformShardAssigner` with the source operator. This makes sure that all source subtasks have shards to process as long as the number of workers is less or equal to the number of active shards.

However, the idleness option with the build-in watermark strategies will not advance the event time if no subtask is reading any event, that is 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 not close.

Summary

- The `withIdleness` setting will not generate new watermarks in case a shard is idle. It will 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 not 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 (for example 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 interrupt 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(l -> 0l)
    .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("XXXXXXXXXXXXXX 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 kinesis put-record --stream-name hp-16 --partition-key 1 --data MQ==
$ aws kinesis put-record --stream-name hp-16 --partition-key 2 --data Mg==
$ aws kinesis 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 kinesis put-record --stream-name hp-16 --partition-key 4 --data NA==
$ aws kinesis 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 kinesis 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:
 850705917302346158658436518579420528 64,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:
 638029437976759618993827388934565396 48,EndingHashKey:
 850705917302346158658436518579420528 63},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:
 850705917302346158658436518579420528 64,EndingHashKey:
 106338239662793269832304564822427566079},SequenceNumberRange: {StartingSequenceNumber:
 496278943384362495989125660432414778027473288653837435 54,}}'} 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:
 638029437976759618993827388934565396 48,EndingHashKey:
 850705917302346158658436518579420528 63},SequenceNumberRange: {StartingSequenceNumber:
 496278943384139488537140354200999420844746805038777631 22,}}'} 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:
 212676479325586539664609129644855132 16,EndingHashKey:
 425352958651173079329218259289710264 31},SequenceNumberRange: {StartingSequenceNumber:
 496278943383693473633169741738168706479293837808658022 58,}}'}, 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:
 191408831393027885698148216680369618 944,EndingHashKey:
 212676479325586539664609129644855132 159},SequenceNumberRange: {StartingSequenceNumber:
 496278943385477533249052191589491563941105706729136457 14,}}'}, 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:
 212676479325586539664609129644855132 15},SequenceNumberRange: {StartingSequenceNumber:
 496278943383470466181184435506753349296567354193598218 26,}}'}, 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:
 255211775190703847597530955573826158 592,EndingHashKey:
 276479423123262501563991868538311671 807},SequenceNumberRange: {StartingSequenceNumber:
 496278943386146555605008110283737635489285157574315870 10,}}'}, 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:
 496278943385254525797066885358076206758379223114076652 82,}}'}, 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:
 212676479325586539664609129644855132 16,EndingHashKey:
 425352958651173079329218259289710264 31},SequenceNumberRange: {StartingSequenceNumber:
 496278943383693473633169741738168706479293837808658022 58,}}'} 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:
 496278943385923548153022804052322278306558673959256065 78,}}'}, 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:
 212676479325586539664609129644855132 15},SequenceNumberRange: {StartingSequenceNumber:
 496278943383470466181184435506753349296567354193598218 26,}}'} 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:
 425352958651173079329218259289710264 32,EndingHashKey:
 638029437976759618993827388934565396 47},SequenceNumberRange: {StartingSequenceNumber:
 496278943383916481085155047969584063662020321423717826 90,}}'}, 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:
 496278943386369563056993416515152992672011641189375674 42,}}'}, 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:
 496278943385254525797066885358076206758379223114076652 82,}}'} 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:
 496278943386369563056993416515152992672011641189375674 42,}}'} 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:
 496278943386146555605008110283737635489285157574315870 10,}}'} 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:
 496278943385923548153022804052322278306558673959256065 78,}}'} 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
XXXXXXXXXXXXXX Window with 3 events; Watermark: 1648030809281, 2022-03-23T10:20:09.281Z
3
1
2
XXXXXXXXXXXXXX Window with 2 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
4
5
XXXXXXXXXXXXXX Window with 1 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
```

```
6
XXXXXXXXXXXXXX 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 the pre-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 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 "global watermark" of the window operator cannot advance.

When the Flink application starts (or if there is data skew), some shards might be consumed faster than others. This can cause some watermarks to be emitted too early from a subtask (the subtask might 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 different watermarking strategies that add a safety buffer (`forBoundedOutOfOrderness(Duration.ofSeconds(30))` or explicitly allow 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 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 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 the 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 behaviors. We recommend that you 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.

Use the links in this section to learn more about Apache flink settings and which ones are modifiable.

This topic contains the following sections:

- Apache Flink configuration
- State backend
- Checkpointing
- Savepointing
- Heap sizes
- Buffer debloating
- Modifiable Flink configuration properties
- Programmatic Flink configuration properties
- View 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 CloudFormation	True
CheckpointInterval	Modifiable	Create Application Update Application CloudFormation	60000
MinPauseBetweenCheckpoints	Modifiable	Create Application	5000

Setting	Can be modified?	How	Default Value
		Update Application CloudFormation	
Unaligned checkpoints	Modifiable	Support case	False
Number of Concurrent Checkpoints	Not Modifiable	N/A	1
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 "Implement application 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-muliplier:

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.type:`

`state.backend.fs.memory-threshold:`

`state.backend.incremental:`

Checkpointing

`execution.checkpointing.unaligned:`

`execution.checkpointing.interval-during-backlog:`

`execution.checkpointing.unaligned.during-recovery.enabled:`

`execution.checkpointing.unaligned.recover-output-on-downstream.enabled:`

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.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:
```

RocksDB options

```
state.backend.rocksdb.compaction.style:

state.backend.rocksdb.memory.partitioned-index-filters:

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.memory.network.fraction:

taskmanager.memory.network.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:`

Programmatic Flink configuration properties

Following are Flink configuration properties that you can modify directly in your application code. Starting with MSF 2.2, an exception will be thrown if you try to modify a property not listed on this page in your application code.

> **Note**
>
> These properties are distinct from settings modified via AWS Support case. For infrastructure-level settings such as TaskManager memory, state backend, RocksDB tuning, and restart strategies, see Modifiable Flink configuration properties.

Pipeline configuration

`execution.runtime-mode`

pipeline.auto-generate-uids

pipeline.auto-watermark-interval

pipeline.cached-files

pipeline.classpaths

pipeline.closure-cleaner-level

pipeline.force-avro

pipeline.force-kryo

pipeline.generic-types

pipeline.global-job-parameters

pipeline.jars

pipeline.jobvertex-parallelism-overrides

pipeline.max-parallelism

pipeline.name

pipeline.object-reuse

pipeline.operator-chaining.chain-operators-with-different-max-parallelism

pipeline.operator-chaining.enabled

pipeline.serialization-config

pipeline.vertex-description-mode

pipeline.vertex-name-include-index-prefix

pipeline.watermark-alignment.allow-unaligned-source-splits

Network

taskmanager.network.adaptive-partitioner.enabled

taskmanager.network.adaptive-partitioner.max-traverse-size

Python API

python.execution-mode

python.operator-chaining.enabled

python.job-options

python.internal.archives-key-map

python.internal.files-key-map

python.internal.requirements-file-key

Table API / SQL

table.exec.async-lookup.buffer-capacity

table.exec.async-lookup.key-ordered-enabled

table.exec.async-lookup.output-mode

table.exec.async-lookup.timeout

table.exec.async-ml-predict.max-concurrent-operations

table.exec.async-ml-predict.output-mode

table.exec.async-ml-predict.timeout

table.exec.async-scalar.max-concurrent-operations

table.exec.async-scalar.max-attempts

table.exec.async-scalar.retry-delay

table.exec.async-scalar.retry-strategy

table.exec.async-scalar.timeout

table.exec.async-state.enabled

```
table.exec.async-table.max-concurrent-operations

table.exec.async-table.max-retries

table.exec.async-table.retry-delay

table.exec.async-table.retry-strategy

table.exec.async-table.timeout

table.exec.async-vector-search.max-concurrent-operations

table.exec.async-vector-search.output-mode

table.exec.async-vector-search.timeout

table.exec.deduplicate.insert-update-after-sensitive-enabled

table.exec.deduplicate.mini-batch.compact-changes-enabled

table.exec.delta-join.cache-enabled

table.exec.disabled-operators

table.exec.iceberg.use-v2-sink

table.exec.interval-join.min-cleanup-interval

table.exec.legacy-cast-behaviour

table.exec.local-hash-agg.adaptive.distinct-value-rate-threshold

table.exec.local-hash-agg.adaptive.enabled

table.exec.local-hash-agg.adaptive.sampling-threshold

table.exec.mini-batch.allow-latency

table.exec.mini-batch.enabled

table.exec.mini-batch.size

table.exec.operator-fusion-codegen.enabled
```

```
table.exec.simplify-operator-name-enabled

table.exec.sink.keyed-shuffle

table.exec.sink.nested-constraint-enforcer

table.exec.sink.not-null-enforcer

table.exec.sink.rowtime-inserter

table.exec.sink.type-length-enforcer

table.exec.sink.upsert-materialize

table.exec.sink.upsert-materialize-strategy.adaptive.threshold.high

table.exec.sink.upsert-materialize-strategy.adaptive.threshold.low

table.exec.sink.upsert-materialize-strategy.type

table.exec.sort.async-merge-enabled

table.exec.sort.default-limit

table.exec.sort.max-num-file-handles

table.exec.source.cdc-events-duplicate

table.exec.source.idle-timeout

table.exec.spill-compression.block-size

table.exec.spill-compression.enabled

table.exec.state.ttl

table.exec.uid.format

table.exec.uid.generation

table.exec.unbounded-over.version

table.exec.window-agg.buffer-size-limit
```

```
table.optimizer.agg-phase-strategy

table.optimizer.adaptive-broadcast-join.strategy

table.optimizer.bushy-join-reorder-threshold

table.optimizer.delta-join.strategy

table.optimizer.distinct-agg.split.bucket-num

table.optimizer.distinct-agg.split.enabled

table.optimizer.dynamic-filtering.enabled

table.optimizer.incremental-agg-enabled

table.optimizer.join-reorder-enabled

table.optimizer.multi-join.enabled

table.optimizer.multiple-input-enabled

table.optimizer.non-deterministic-update.strategy

table.optimizer.reuse-optimize-block-with-digest-enabled

table.optimizer.reuse-sink-enabled

table.optimizer.reuse-source-enabled

table.optimizer.reuse-sub-plan-enabled

table.optimizer.runtime-filter.enabled

table.optimizer.skewed-join-optimization.skewed-factor

table.optimizer.skewed-join-optimization.skewed-threshold

table.optimizer.skewed-join-optimization.strategy

table.optimizer.source.report-statistics-enabled

table.optimizer.union-all-as-breakpoint-enabled
```

```
table.builtin-catalog-name

table.builtin-database-name

table.catalog-modification.listeners

table.column-expansion-strategy

table.display.max-column-width

table.dml-sync

table.dynamic-table-options.enabled

table.generated-code.max-length

table.legacy-nested-row-nullability

table.local-time-zone

table.plan.compile.catalog-objects

table.plan.force-recompile

table.plan.restore.catalog-objects

table.rtas-ctas.atomicity-enabled

table.sql-dialect
```

View 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.

Configure 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
- Use the Managed Service for Apache Flink VPC API
- Example: Use 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 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 Implement application 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.

Add a permissions policy for accessing an Amazon VPC

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "VPCReadOnlyPermissions",
      "Effect": "Allow",
```

```
      "Action": [
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeDhcpOptions"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ENIReadWritePermissions",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DeleteNetworkInterface"
      ],
      "Resource": "*"
    }

    ]
}
```

Note

When you specify application resources using the console (such as CloudWatch Logs or an Amazon VPC), the console modifies your application execution role to grant permission to access those resources. You only need to manually modify your application's execution role if you create your application without using the console.

Internet and service access for a VPC-connected 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:

- Sign in to the AWS Management Console, and open the Amazon MSF 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

Use the 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:::amzn-s3-demo-bucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "CodeContentType":"ZIPFILE"
    },
```

```
      "FlinkApplicationConfiguration":{
      "ParallelismConfiguration":{
        "ConfigurationType":"CUSTOM",
        "Parallelism":2,
        "ParallelismPerKPU":1,
        "AutoScalingEnabled":true
      }
    },
  "VpcConfigurations": [
        {
           "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
           "SubnetIds": [ "subnet-0123456789abcdef0" ]
        }
      ]
  }
}
```

AddApplicationVpcConfiguration

Use the AddApplicationVpcConfiguration action to add a VPC configuration to your application after it has been created.

The following example request code for the AddApplicationVpcConfiguration action adds a VPC configuration to an existing application:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfiguration": {
      "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
      "SubnetIds": [ "subnet-0123456789abcdef0" ]
   }
}
```

DeleteApplicationVpcConfiguration

Use the DeleteApplicationVpcConfiguration action to remove a VPC configuration from your application.

The following example request code for the AddApplicationVpcConfiguration action removes an existing VPC configuration from an application:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfigurationId": "1.1"
}
```

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: Use 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.

Troubleshoot Managed Service for Apache Flink

The following topics can help you troubleshoot problems that you might encounter with Amazon Managed Service for Apache Flink.

Choose the appropriate topic to review solutions.

Topics

- Development troubleshooting
- Runtime troubleshooting

Development troubleshooting

This section contains information about diagnosing and fixing development issues with your Managed Service for Apache Flink application.

Topics

- 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
- Real-time vector embedding blueprints FAQ and troubleshooting

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 Enable 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 `Credential Provider` 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://amzn-s3-demo-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 `ClosedException`. 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:

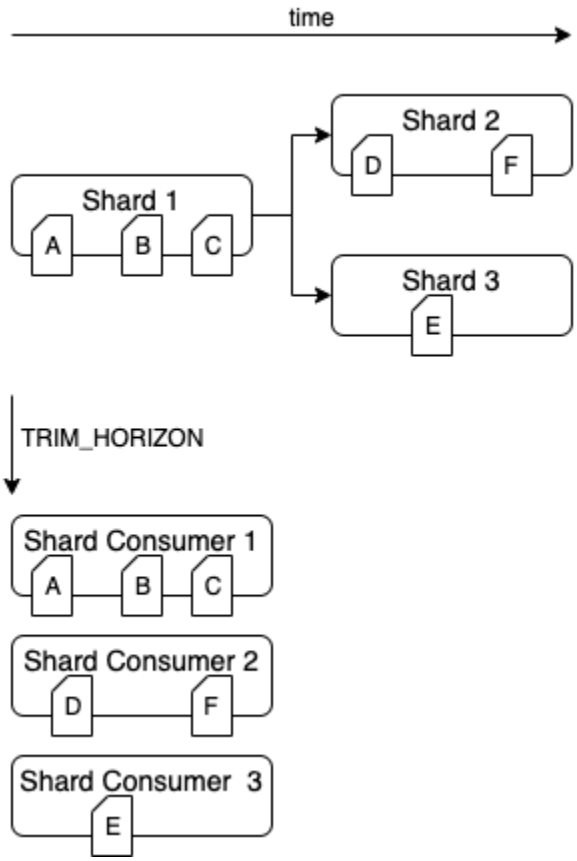
```
mvn clean install -DskipTests -Dfast -Dflink.version=1.15.4 -
Daws.sdk.version=2.20.144
```

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.

Real-time vector embedding blueprints FAQ and troubleshooting

Review the following FAQ and troubleshooting sections to troubleshoot real-time vector embedding blueprint issues. For more information about real-time vector embedding blueprints, see Real-time vector embedding blueprints.

For general Managed Service for Apache Flink application troubleshooting, see https://docs.aws.amazon.com/managed-flink/latest/java/troubleshooting-runtime.html.

Topics

- Real-time vector embedding blueprints - FAQ
- Real-time vector embedding blueprints - troubleshooting

Real-time vector embedding blueprints - FAQ

Review the following FAQ about real-time vector embedding blueprints. For more information about real-time vector embedding blueprints, see Real-time vector embedding blueprints.

FAQ

- What AWS resources does this blueprint create?
- What are my actions after the AWS CloudFormation stack deployment is complete?
- What should be the structure of the data in the source Amazon MSK topic(s)?
- Can I specify parts of a message to embed?
- Can I read data from multiple Amazon MSK topics?
- Can I use regex to configure Amazon MSK topic names?
- What is the maximum size of a message that can be read from an Amazon MSK topic?
- What type of OpenSearch is supported?
- Why do I need to use a vector search collection, vector index, and add a vector field in my OpenSearch Serverless colelction?
- What should I set as the dimension for my vector field?
- What does the output look like in the configured OpenSearch index?
- Can I specify metadata fields to add to the document stored in the OpenSearch index?
- Should I expect duplicate entries in the OpenSearch index?
- Can I send data to multiple OpenSearch indices?
- Can I deploy multiple real-time vector embedding applications in a single AWS account?
- Can multiple real-time vector embedding applications use the same data source or sink?
- Does the application support cross-account connectivity?
- Does the application support cross-Region connectivity?
- Can my Amazon MSK cluster and OpenSearch collection be in different VPCs or subnets?
- What embedding models are supported by the application?
- Can I fine-tune the performance of my application based on my workload?
- What Amazon MSK authentication types are supported?
- What is sink.os.bulkFlushIntervalMillis and how do I set it?

- When I deploy my Managed Service for Apache Flink application, from what point in the Amazon MSK topic will it begin reading messages?
- How do I use source.msk.starting.offset?
- What chunking strategies are supported?
- How do I read records in my vector datastore?
- Where can I find new updates to the source code?
- Can I make a change to the AWS CloudFormation template and update the Managed Service for Apache Flink application?
- Will AWS monitor and maintain the application on my behalf?
- Does this application move my data outside my AWS account?

What AWS resources does this blueprint create?

To find resources deployed in your account, navigate to AWS CloudFormation console and identify the stack name that starts with the name you provided for your Managed Service for Apache Flink application. Choose the **Resources** tab to check the resources that were created as part of the stack. The following are the key resources that the stack creates:

- Real-time vector embedding Managed Service for Apache Flink application
- Amazon S3 bucket for holding the source code for the real-time vector embedding application
- CloudWatch log group and log stream for storing logs
- Lambda functions for fetching and creating resources
- IAM roles and policies for Lambdas, Managed Service for Apache Flink application, and accessing Amazon Bedrock and Amazon OpenSearch Service
- Data access policy for Amazon OpenSearch Service
- VPC endpoints for accessing Amazon Bedrock and Amazon OpenSearch Service

What are my actions after the AWS CloudFormation stack deployment is complete?

After the AWS CloudFormation stack deployment is complete, access the Managed Service for Apache Flink console and find your blueprint Managed Service for Apache Flink application. Choose the **Configure** tab and confirm that all runtime properties are setup correctly. They may overflow to the next page. When you are confident of the settings, choose **Run**. The application will start ingesting messages from your topic.

To check for new releases, see https://github.com/awslabs/real-time-vectorization-of-streaming-data/releases.

What should be the structure of the data in the source Amazon MSK topic(s)?

We currently support structured and unstructured source data.

- Unstructured data is denoted by `STRING` in `source.msk.data.type`. The data is read as is from the incoming message.
- We currently support structured JSON data, denoted by `JSON` in `source.msk.data.type`. The data must always be in JSON format. If the application receives a malformed JSON, the application will fail.
- When using JSON as source data type, make sure that every message in all source topics is a valid JSON. If you subscribe to one or more topics that do not contain JSON objects with this setting, the application will fail. If one or more topics have a mix of structured and unstructured data, we recommended that you configure source data as unstructured in the Managed Service for Apache Flink application.

Can I specify parts of a message to embed?

- For unstructured input data where `source.msk.data.type` is `STRING`, the application will always embed the entire message and store the entire message in the configured OpenSearch index.
- For structured input data where `source.msk.data.type` is `JSON`, you can configure `embed.input.config.json.fieldsToEmbed` to specify which field in the JSON object should be selected for embedding. This only works for top-level JSON fields and does not work with nested JSONs and with messages containing a JSON array. Use .* to embed the entire JSON.

Can I read data from multiple Amazon MSK topics?

Yes, you can read data from multiple Amazon MSK topics with this application. Data from all topics must be of the same type (either STRING or JSON) or it might cause the application to fail. Data from all topics is always stored in a single OpenSearch index.

Can I use regex to configure Amazon MSK topic names?

`source.msk.topic.names` does not support a list of regex. We support either a comma separated list of topic names or `.*` regex to include all topics.

What is the maximum size of a message that can be read from an Amazon MSK topic?

The maximum size of a message that can be processed is limited by the Amazon Bedrock InvokeModel body limit that is currently set to 25,000,000. For more information, see InvokeModel.

What type of OpenSearch is supported?

We support both OpenSearch domains and collections. If you are using an OpenSearch collection, make sure to use a vector collection and create a vector index to use for this application. This will let you use the OpenSearch vector database capabilities for querying your data. To learn more, seeAmazon OpenSearch Service's vector database capabilities explained.

Why do I need to use a vector search collection, vector index, and add a vector field in my OpenSearch Serverless colelction?

The *vector search* collection type in OpenSearch Serverless provides a similarity search capability that is scalable and high performing. It streamlines building modern machine learning (ML) augmented search experiences and generative artificial intelligence (AI) applications. For more information, see Working with vector search collections.

What should I set as the dimension for my vector field?

Set the dimension of the vector field based on the embedding model that you want to use. Refer to the following table, and confirm these values from the respective documentation.

Vector field dimensions

Amazon Bedrock vector embedding model name	Output dimension support offered by the model
Amazon Titan Text Embeddings V1	1,536
Amazon Titan Text Embeddings V2	1,024 (default), 384, 256
Amazon Titan Multimodal Embeddings G1	1,024 (default), 384, 256
Cohere Embed English	1,024
Cohere Embed Multilingual	1,024

What does the output look like in the configured OpenSearch index?

Every document in the OpenSearch index contains following fields:

- **original_data**: The data that was used to generate embeddings. For STRING type, it is the entire message. For JSON object, it is the JSON object that was used for embeddings. It could be the entire JSON in the message or specified fields in the JSON. For example, if name was selected to be embedded from incoming messages, the output would look as follows:

```
"original_data": "{\"name\":\"John Doe\"}"
```

- **embedded_data**: A vector float array of embeddings generated by Amazon Bedrock
- **date**: UTC timestamp at which the document was stored in OpenSearch

Can I specify metadata fields to add to the document stored in the OpenSearch index?

No, currently, we do not support adding additional fields to the final document stored in the OpenSearch index.

Should I expect duplicate entries in the OpenSearch index?

Depending on how you configured your application, you might see duplicate messages in the index. One common reason is application restart. The application is configured by default to start reading from the earliest message in the source topic. When you change the configuraiton, the application restarts, and processes all messages in the topic again. To avoid re-processing, refer to the documentation on how to use source.msk.starting.offset and correctly set the starting offset for your application.

Can I send data to multiple OpenSearch indices?

No, the application supports storing data to a single OpenSearch index. To setup vectorization output to multiple indices, you must deploy separate Managed Service for Apache Flink applications.

Can I deploy multiple real-time vector embedding applications in a single AWS account?

Yes, you can deploy multiple real-time vector embedding Managed Service for Apache Flink applications in a single AWS account if every application has a unique name.

Can multiple real-time vector embedding applications use the same data source or sink?

Yes, you can create multiple real-time vector embedding Managed Service for Apache Flink applications that read data from the same topic(s) or store data in the same index.

Does the application support cross-account connectivity?

No, for the application to run successfully, the Amazon MSK cluster and the OpenSearch collection must be in the same AWS account where you are trying to setup your Managed Service for Apache Flink application.

Does the application support cross-Region connectivity?

No, the application only allows you to deploy an Managed Service for Apache Flink application with an Amazon MSK cluster and an OpenSearch collection in the same Region of the Managed Service for Apache Flink application.

Can my Amazon MSK cluster and OpenSearch collection be in different VPCs or subnets?

Yes, we support Amazon MSK cluster and OpenSearch collection in different VPCs and subnets as long as they are in the same AWS account. See (General MSF troubleshooting) to make sure your setup is correct.

What embedding models are supported by the application?

Currently, the application supports all models that are supported by Bedrock. These include:

- Amazon Titan Embeddings G1 - Text

- Amazon Titan Text Embeddings V2
- Amazon Titan Multimodal Embeddings G1
- Cohere Embed English
- Cohere Embed Multilingual

Can I fine-tune the performance of my application based on my workload?

Yes. The throughput of the application depends on a number of factors, all of which can be controlled by the customers:

1. **AWS MSF KPUs**: The application is deployed with default parallelism factor 2 and parallelism per KPU 1, with automatic scaling turned on. However, we recommend that you configure

scaling for the Managed Service for Apache Flink application according to your workloads. For more information, see Review Managed Service for Apache Flink application resources.

2. **Amazon Bedrock**: Based on the selected Amazon Bedrock on-demand model, different quotas might apply. Review service quotas in Bedrock to see the workload that the service will be able to handle. For more information, see Quotas for Amazon Bedrock.
3. **Amazon OpenSearch Service**: Additionally, in some situations, you might notice that OpenSearch is the bottleneck in your pipeline. For scaling information, see OpenSearch scaling Sizing Amazon OpenSearch Service domains.

What Amazon MSK authentication types are supported?

We only support the IAM MSK authentication type.

What is `sink.os.bulkFlushIntervalMillis` and how do I set it?

When sending data to Amazon OpenSearch Service, the bulk flush interval is the interval at which the bulk request is run, regardless of the number of actions or the size of the request. The default value is set to 1 millisecond.

While setting a flush interval can help to make sure that data is indexed timely, it can also lead to increased overhead if set too low. Consider your use case and the importance of timely indexing when choosing a flush interval.

When I deploy my Managed Service for Apache Flink application, from what point in the Amazon MSK topic will it begin reading messages?

The application will start reading messages from the Amazon MSK topic at the offset specified by the `source.msk.starting.offset` configuration set in the application's runtime configuration. If `source.msk.starting.offset` is not explicitly set, the default behavior of the application is to start reading from the earliest available message in the topic.

How do I use `source.msk.starting.offset`?

Explicitly set `source.msk.starting.offset` to one of the following values, based on desired behavior:

- EARLIEST: The default setting, which reads from oldest offset in the partition. This is a good choice especially if:

  - You have newly created Amazon MSK topics and consumer applications.
  - You need to replay data, so you can build or reconstruct state. This is relevant when implementing the event sourcing pattern or when initializing a new service that requires a complete view of the data history.
- LATEST: The Managed Service for Apache Flink application will read messages from the end of the partition. We recommend this option if you only care about new messages being produced and don't need to process historical data. In this setting, the consumer will ignore the existing messages and only read new messages published by the upstream producer.
- COMMITTED: The Managed Service for Apache Flink application will start consuming messages from the committed offset of the consuming group. If the committed offset doesn't exist, the EARLIEST reset strategy will be used.

What chunking strategies are supported?

We are using the langchain library to chunk inputs. Chunking is only applied if the length of the input is greater than the chosen `maxSegmentSizeInChars`. We support the following five chunking types:

- `SPLIT_BY_CHARACTER`: Will fit as many characters as it can into each chunk where each chunk length is no greater than maxSegmentSizeInChars. Doesn’t care about whitespace, so it can cut off words.
- `SPLIT_BY_WORD`: Will find whitespace characters to chunk by. No words are cut off.
- `SPLIT_BY_SENTENCE`: Sentence boundaries are detected using the Apache OpenNLP library with the English sentence model.
- `SPLIT_BY_LINE`: Will find new line characters to chunk by.
- `SPLIT_BY_PARAGRAPH`: Will find consecutive new line characters to chunk by.

The splitting strategies fall back according to the preceding order, where the larger chunking strategies like `SPLIT_BY_PARAGRAPH` fall back to `SPLIT_BY_CHARACTER`. For example, when using `SPLIT_BY_LINE`, if a line is too long then the line will be sub-chunked by sentence, where each chunk will fit in as many sentences as it can. If there are any sentences that are too long, then it will be chunked at the word-level. If a word is too long, then it will be split by character.

How do I read records in my vector datastore?

1. When `source.msk.data.type` is `STRING`

- **original_data**: The entire original string from the Amazon MSK message.
- **embedded_data**: Embedding vector created from `chunk_data` if it is not empty (chunking applied) or created from `original_data` if no chunking was applied.
- **chunk_data**: Only present when the original data was chunked. Contains the chunk of the original message that was used to create the embedding in `embedded_data`.

2. When `source.msk.data.type` is `JSON`

- **original_data**: The entire original JSON from the Amazon MSK message *after* JSON key filtering is applied.
- **embedded_data**: Embedding vector created from `chunk_data` if it is not empty (chunking applied) or created from `original_data` if no chunking was applied.
- **chunk_key**: Only present when the original data was chunked. Contains the JSON key that the chunk is from in `original_data`. For example, it can look like `jsonKey1.nestedJsonKeyA` for nested keys or *metadata* in the example of `original_data`.
- **chunk_data**: Only present when the original data was chunked. Contains the chunk of the original message that was used to create the embedding in `embedded_data`.

Yes, you can read data from multiple Amazon MSK topics with this application. Data from all topics must be of the same type (either STRING or JSON) or it might cause the application to fail. Data from all topics is always stored in a single OpenSearch index.

Where can I find new updates to the source code?

Go to https://github.com/awslabs/real-time-vectorization-of-streaming-data/releases to check for new releases.

Can I make a change to the AWS CloudFormation template and update the Managed Service for Apache Flink application?

No, making a change to the AWS CloudFormation template does not update the Managed Service for Apache Flink application. Any new change in AWS CloudFormation implies a new stack needs to be deployed.

Will AWS monitor and maintain the application on my behalf?

No, AWS will not monitor, scale, update or patch this application on your behalf.

Does this application move my data outside my AWS account?

All data read and stored by the Managed Service for Apache Flink application stays within your AWS account and never leaves your account.

Real-time vector embedding blueprints - troubleshooting

Review the following troubleshooting topics about real-time vector embedding blueprints. For more information about real-time vector embedding blueprints, see Real-time vector embedding blueprints.

Troubleshooting topics

- My CloudFormation stack deployment has failed or rolled back. What can I do to fix it?
- I don't want my application to start reading messages from the beginning of the Amazon MSK topics. What do I do?
- How do I know if there is an issue with my Managed Service for Apache Flink application and how can I debug it?
- What are the key metrics that I should be monitoring for my Managed Service for Apache Flink application?

My CloudFormation stack deployment has failed or rolled back. What can I do to fix it?

- Go to your CFN stack and find the reason for the stack failure. It could be related to missing permissions, AWS resource name collisions, among other causes. Fix the root cause of the deployment failure. For more information, see the CloudWatch troubleshooting guide.
- [Optional] There can only be one VPC endpoint per service per VPC. If you deployed multiple real-time vector embedding blueprints to write to the Amazon OpenSearch Service collections in the same VPC, they might be sharing VPC endpoints. These might either already be present in your account for the VPC, or the first real-time vector embedding blueprint stack will create VPC endpoints for Amazon Bedrock and Amazon OpenSearch Service that will be used by all other stacks deployed in your account. If a stack fails, check if that stack created VPC endpoints for Amazon Bedrock and Amazon OpenSearch Service and delete them if they are not used anywhere else in your account. For steps for deleting VPC endpoints, refer to the documentation on how to safely delete your application.
- There might be other services or applications in your account using the VPC endpoint. Deleting it might create network disruption for other services. Be careful in deleting these endpoints.

I don't want my application to start reading messages from the beginning of the Amazon MSK topics. What do I do?

You must explicitly set `source.msk.starting.offset` to one of the following values, depending on the desired behavior:

- **Earliest offset**: The oldest offset in the partition.
- **Latest offset**: Consumers will read messages from the end of the partition.
- **Committed offset**: Read from the last message the consumer processed within a partition.

How do I know if there is an issue with my Managed Service for Apache Flink application and how can I debug it?

Use the Managed Service for Apache Flink troubleshooting guide to debug Managed Service for Apache Flink related issues with your application.

What are the key metrics that I should be monitoring for my Managed Service for Apache Flink application?

- All metrics available for a regular Managed Service for Apache Flink application can help you monitor your application. For more information, see Metrics and dimensions in Managed Service for Apache Flink.
- To monitor Amazon Bedrock metrics, see Amazon CloudWatch metrics for Amazon Bedrock.
- We have added two new metrics for monitoring performance of generating embeddings. Find them under the `EmbeddingGeneration` operation name in CloudWatch. The two metrics are:
  - **BedrockTitanEmbeddingTokenCount**: Number of tokens present in a single request to Amazon Bedrock.
  - **BedrockEmbeddingGenerationLatencyMs**: Reports the time taken to send and receive a response from Amazon Bedrock for generating embeddings, in milliseconds.
- For Amazon OpenSearch Service serverless collections, you can use metrics such as `IngestionDataRate`, `IngestionDocumentErrors` and others. For more information, see Monitoring OpenSearch Serverless with Amazon CloudWatch.
- For OpenSearch provisioned metrics, see Monitoring OpenSearch cluster metrics with Amazon CloudWatch.

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
- Integrate 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 Use 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 Manage application backups using 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 in Amazon Managed Service for Apache Flink.

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 Work 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 Troubleshoot performance issues.
- 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 Use CloudFormation.

- **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 Troubleshoot performance issues.
- 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 Implement application 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 Troubleshoot performance issues.

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 RocksBS). 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 `LimitExceededException` 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.

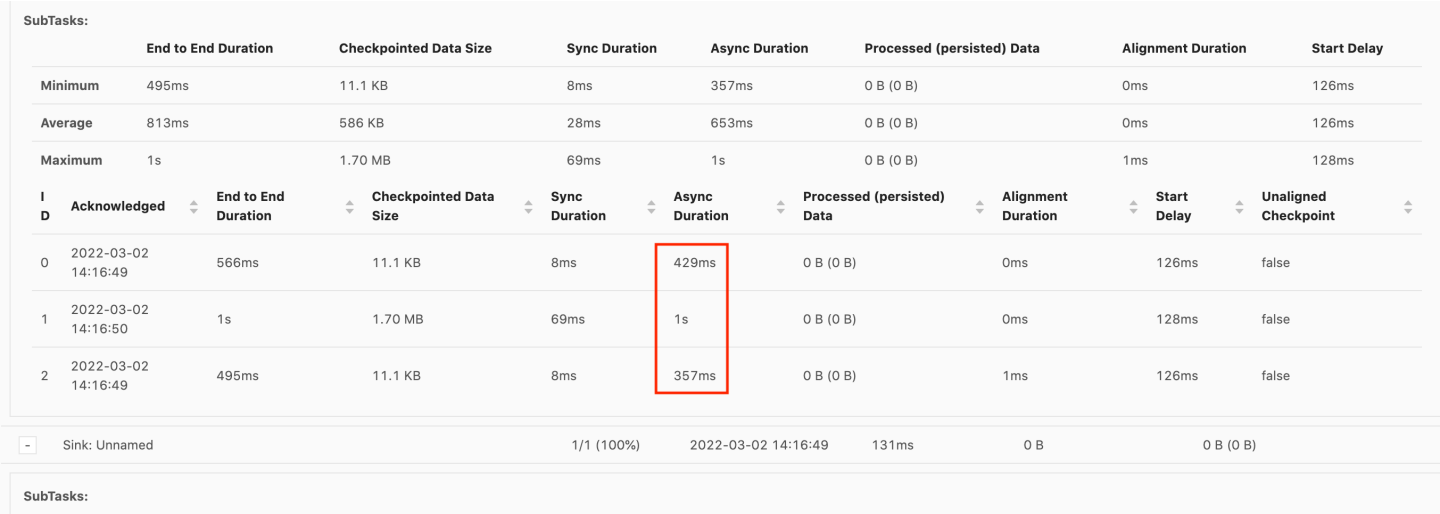
The first thing to look at is the **End to End Duration** of each operator in the Job graph to determine which operator is taking long to checkpoint and warrants further investigation. Per the Flink documentation, the definition of the duration is:

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:

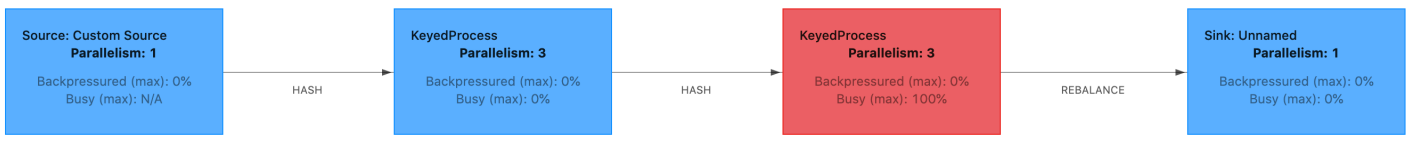
	End to End Duration	Checkpointed Data Size	Sync Duration	Async Duration	Processed (persisted) Data	Alignment Duration	Start Delay
Minimum	495ms	11.1 KB	8ms	357ms	0 B (0 B)	0ms	126ms
Average	813ms	586 KB	28ms	653ms	0 B (0 B)	0ms	126ms
Maximum	1s	1.70 MB	69ms	1s	0 B (0 B)	1ms	128ms

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(KeyedProcessOperat
    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 it's 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.

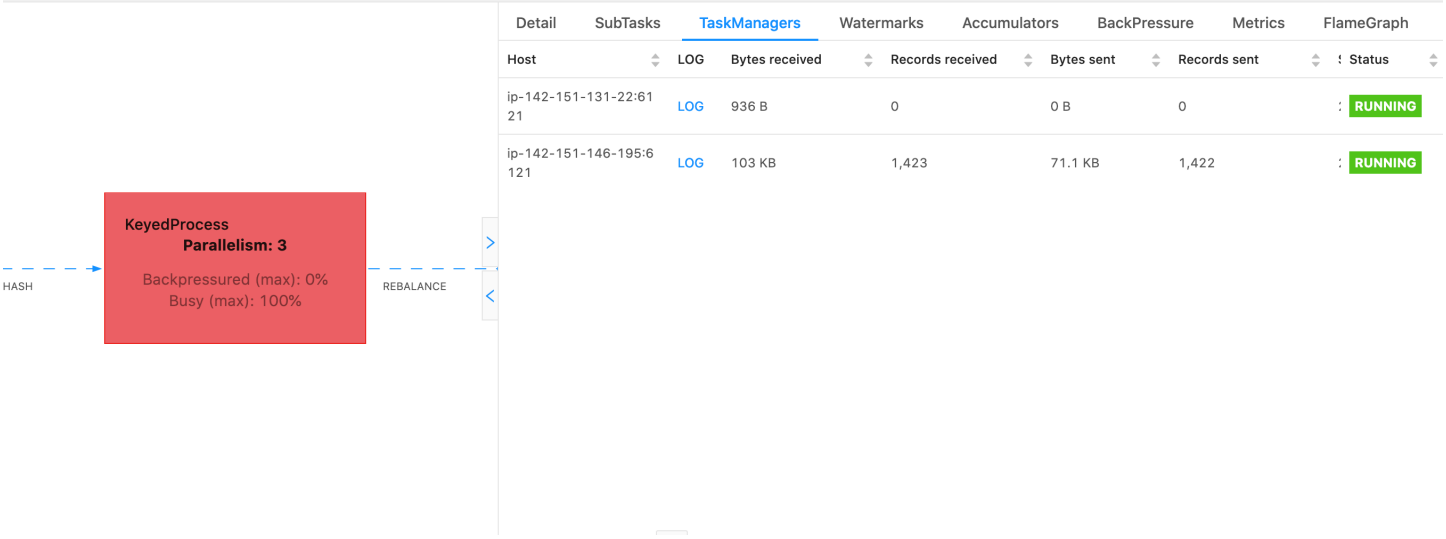
> **Note**
>
> In Flink 1.13, the maximum depth of a single stacktrace in the thread dump is limited to 8.

> **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 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(KeyedProcessOperat
    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
    ...
```

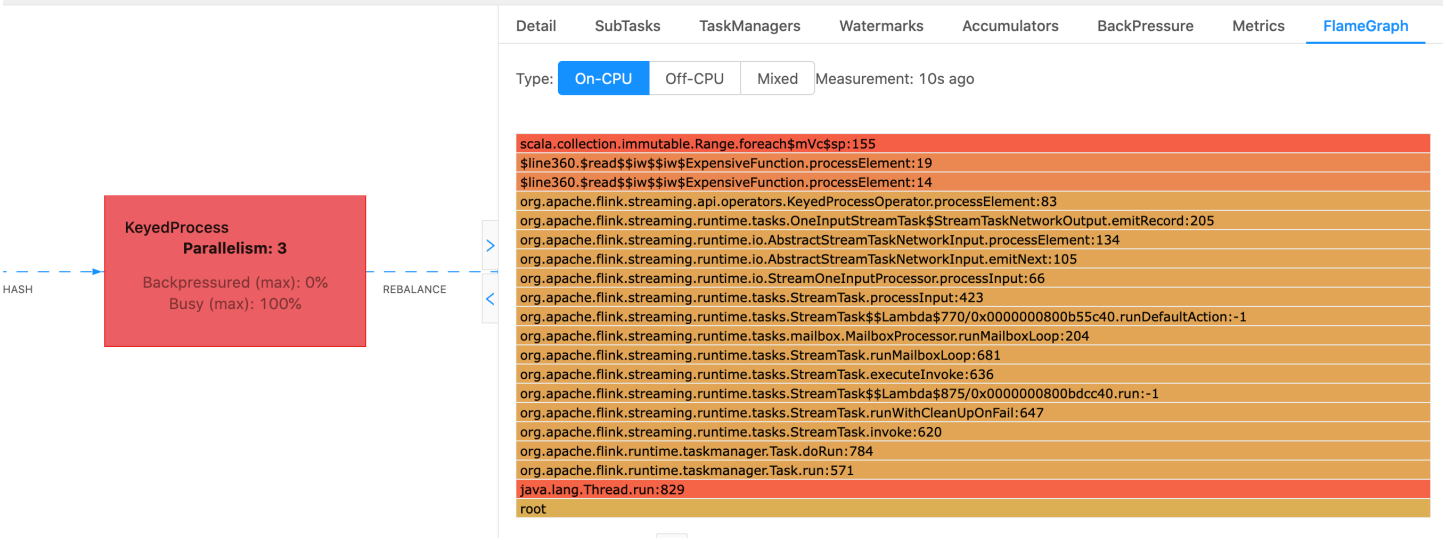
This can become confusing if there are multiple operators with the same name but we can name operators to get around this. For example:

```
....
.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 Implement application 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

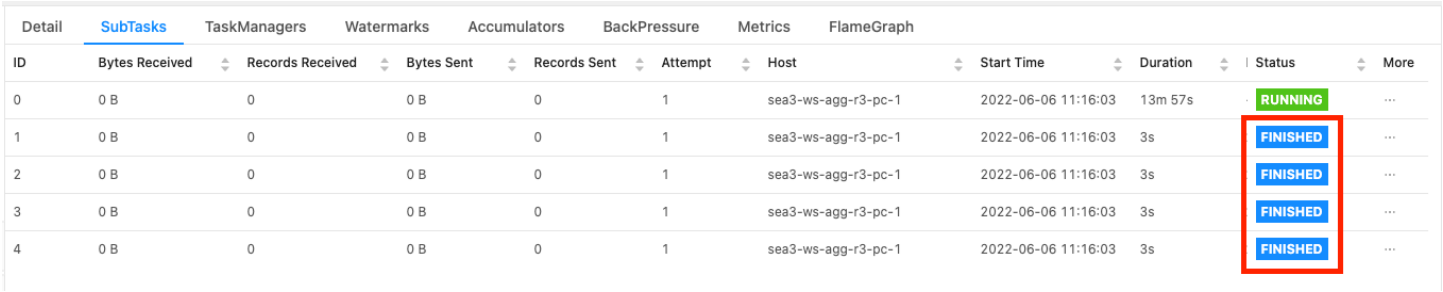
If your Beam application is configured with shutdownSourcesAfterIdleMs set to 0ms, checkpoints can fail to trigger because tasks are in "FINISHED" state. This section describes symptoms and resolution for this condition.

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(CheckpointCoordina
                "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.



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:

- 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 r esembles 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.

Integrate 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.

Change	Description	Date
Support for Apache Flink version 2.2	Amazon Managed Service for Apache Flink now supports Apache Flink version 2.2. This is a major version upgrade with breaking changes. Review breaking changes and new features in Amazon Managed Service for Apache Flink 2.2.	March 31, 2026
Support for AWS KMS customer managed keys	Amazon Managed Service for Apache Flink (Amazon MSF) now supports using AWS KMS customer managed keys (CMKs) to encrypt application data at rest. This is available for new or existing applicati ons running Apache Flink 1.20. For more informati on, see Key management in Amazon Managed Service for Apache Flink.	August 20, 2025
Support for Apache Flink version 1.15.2	Managed Service for Apache Flink now supports applicati ons that use Apache Flink version 1.15.2. Create Kinesis Data Analytics applications	November 22, 2022

Change	Description	Date
	using the Apache Flink Table API. For more information, see Create an application.	
Support for Apache Flink version 1.13.2	Managed Service for Apache Flink now supports applications that use Apache Flink version 1.13.2. Create Kinesis Data Analytics applications using the Apache Flink Table API. For more information, see Getting Started: Flink 1.13.2.	October 13, 2021
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 Use 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 Create an application.	November 19, 2020
Apache Flink Dashboard	Use the Apache Flink Dashboard to monitor application health and performance. For more information, see Use the Apache Flink Dashboard.	November 19, 2020

Change	Description	Date
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
Apache Beam	Create applications that use Apache Beam to process streaming data. For more information, see Use CloudFormation.	September 15, 2020
Performance	How to troubleshoot applicati on performance issues, and how to create a performan t application. For more information, see ???.	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 ???.	June 5, 2020
New CloudWatch Metrics	Managed Service for Apache Flink now emits 22 metrics to Amazon CloudWatch Metrics. For more information, see ???.	May 12, 2020

Change	Description	Date
Custom CloudWatch Metrics	Define application-specific metrics and emit them to Amazon CloudWatch Metrics. For more information, see ???.	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 Create an application.	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 Configure MSF to access resources in 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 ???.	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 ???.	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 Use 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 Add tags to Managed Service for Apache Flink applications.	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 ???.	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 informati on, see Firehose sink.	December 13, 2018
Public release	This is the initial release of the *Managed Service for Apache Flink Developer Guide for Java Applications*.	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 kinesisanalyticsv2 start-application --cli-input-json file://start.json
```

For more information about using JSON with the AWS CLI, see Generate CLI Skeleton and CLI Input JSON Parameters in the *AWS Command Line Interface User Guide*.

Topics

- AddApplicationCloudWatchLoggingOption
- 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:::amzn-s3-demo-bucket",
         "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:::amzn-s3-demo-bucket",
          "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:::amzn-s3-demo-bucket",
      "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:::amzn-s3-demo-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.

Release versions

This content was moved to Release versions. See Supported and deprecated Apache Flink versions.