



Developer Guide

Amazon Managed Streaming for Apache Kafka



Amazon Managed Streaming for Apache Kafka: Developer Guide

Copyright © 2025 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

Welcome	1
What is Amazon MSK?	1
Setting up	3
Sign up for AWS	3
Download libraries and tools	3
MSK Provisioned	5
What is MSK Provisioned?	1
Get started	5
Create a cluster	6
Create an IAM role	7
Create a client machine	10
Create a topic	12
Produce and consume data	15
View metrics	16
Delete the tutorial resources	17
How it works	17
Manage your Provisioned cluster	18
Create a cluster	19
List clusters	24
Connect to an MSK Provisioned cluster	25
Get the bootstrap brokers	46
Monitor a cluster	48
Update cluster security	88
Expand a cluster	91
Remove a broker	94
Update cluster broker size	99
Use Cruise Control	102
Update cluster configuration	107
Reboot a broker for an Amazon MSK cluster	110
Tag a cluster	112
Migrate to Amazon MSK Cluster	114
Delete a cluster	118
Key features and concepts	119
Broker types	120

Broker sizes	123
Storage management	124
Security	144
Broker configuration	209
Patching	276
Broker offline and client failover	277
Amazon MSK logging	280
Metadata management	287
Resources	291
Apache Kafka versions	291
Troubleshoot Amazon MSK cluster	304
Best practices	314
Best practices for Standard brokers	314
Best practices for Express brokers	322
Best practices for Apache Kafka clients	327
MSK Serverless	334
Use MSK Serverless clusters	335
Create a cluster	335
Create an IAM role	337
Create a client machine	339
Create a topic	341
Produce and consume data	341
Delete resources	342
Configuration	343
Monitoring	345
MSK Connect	348
Amazon MSK Connect benefits	348
Getting started	350
Set up resources required for MSK Connect	350
Create custom plugin	355
Create client machine and Apache Kafka topic	356
Create connector	358
Send data to the MSK cluster	359
Understand connectors	360
Understand connector capacity	361
Create a connector	362

Update a connector	364
Connecting from connectors	364
Create custom plugins	365
Understand MSK Connect workers	366
Default worker configuration	366
Supported worker configuration properties	366
Create a custom configuration	368
Manage connector offsets	369
Configuration providers	373
Considerations	373
Create custom plugin and upload to S3	373
Configure parameters and permissions for different providers	375
Create custom worker config	380
Create the connector	381
IAM roles and policies	381
Understand service execution role	382
Example policy	384
Prevent cross-service confused deputy problem	387
AWS managed policies	388
Use service-linked roles	392
Enable internet access	393
Set up a NAT gateway	394
Understand private DNS hostnames	396
Configure a VPC DHCP option	397
Configure DNS attributes	397
Handle connector creation failures	397
Security	398
Logging	398
Preventing secrets from appearing in connector logs	400
Monitoring	400
Examples	403
Set up Amazon S3 sink connector	403
Set up EventBridge Kafka sink connector	405
Use Debezium source connector	411
Migrate to Amazon MSK Connect	422
Understand internal topics used by Kafka Connect	422

State management	423
Migrate source connectors	423
Migrate sink connectors	425
Troubleshooting	426
MSK Replicator	427
How Amazon MSK Replicator works	428
Data replication	428
Metadata replication	429
Topic name configuration	430
Set up source and target clusters	432
Prepare the Amazon MSK source cluster	432
Prepare the Amazon MSK target cluster	435
Tutorial: Create an Amazon MSK Replicator	435
Considerations for creating an Amazon MSK Replicator	436
Create replicator with AWS console	440
Edit MSK Replicator settings	447
Delete an MSK Replicator	448
Monitor replication	449
MSK Replicator metrics	449
Use replication to increase resiliency	459
Considerations for building multi-Region Apache Kafka applications	459
Using active-active versus active-passive cluster topology	459
Create an active-passive Kafka cluster	460
Failover to the secondary Region	460
Perform a planned failover	460
Perform an unplanned failover	462
Perform failback	463
Create an active-active setup	465
Migrate from one Amazon MSK cluster to another	466
Migrate from self-managed MirrorMaker2 to MSK Replicator	467
Troubleshoot MSK Replicator	467
MSK Replicator state goes from CREATING to FAILED	467
MSK Replicator appears stuck in the CREATING state	468
MSK Replicator is not replicating data or replicating only partial data	468
Message offsets in the target cluster are different than the source cluster	469

MSK Replicator is not syncing consumer groups offsets or consumer group does not exist on target cluster	469
Replication latency is high or keeps increasing	470
Using ReplicatorFailure metric	472
Best practices for using MSK Replicator	478
Managing MSK Replicator throughput using Kafka quotas	478
Setting cluster retention period	479
MSK integrations	480
Athena connector for Amazon MSK	480
Redshift integration for Amazon MSK	480
Firehose integration for Amazon MSK	480
Access EventBridge pipes	481
Kafka Streams with Express brokers and MSK Serverless	483
Creating a Kafka Streams application	483
Real-time vector embedding blueprints	487
Logging and observability	488
Notes before enabling real-time vector embedding blueprints	489
Deploy streaming data vectorization blueprint	489
Quota	493
Requesting a quota increase in Amazon MSK	493
Standard broker quota	494
Express broker quota	496
Express broker throughput throttle limits by broker size	498
MSK Replicator quotas	499
Quota for serverless clusters	499
MSK Connect quota	501
Document history	502

Welcome to the Amazon MSK Developer Guide

Welcome to the *Amazon MSK Developer Guide*. The following topics can help you get started using this guide, based on what you're trying to do.

- Create an MSK Provisioned cluster by following the [Get started using Amazon MSK](#) tutorial.
- Dive deeper into the functionality of MSK Provisioned in [MSK Provisioned](#).
- Run Apache Kafka without having to manage and scale cluster capacity with [What is MSK Serverless?](#).
- Use [Understand MSK Connect](#) to stream data to and from your Apache Kafka cluster.
- Use [What is Amazon MSK Replicator?](#) to reliably replicate data across MSK Provisioned clusters in different or the same AWS region(s).

For highlights, product details, and pricing, see the service page for [Amazon MSK](#).

What is Amazon MSK?

Amazon Managed Streaming for Apache Kafka (Amazon MSK) is a fully managed service that enables you to build and run applications that use Apache Kafka to process streaming data. Amazon MSK provides the control-plane operations, such as those for creating, updating, and deleting clusters. It lets you use Apache Kafka data-plane operations, such as those for producing and consuming data. It runs open-source versions of Apache Kafka. This means existing applications, tooling, and plugins from partners and the Apache Kafka community are supported without requiring changes to application code. You can use Amazon MSK to create clusters that use any of the Apache Kafka versions listed under [the section called “Supported Apache Kafka versions”](#).

These components describe the architecture of Amazon MSK:

- **Broker nodes** — When creating an Amazon MSK cluster, you specify how many broker nodes you want Amazon MSK to create in each Availability Zone. The minimum is one broker per Availability Zone. Each Availability Zone has its own virtual private cloud (VPC) subnet. Amazon MSK Provisioned offers two broker types -[Amazon MSK Standard brokers](#) and [Amazon MSK Express brokers](#). In [MSK Serverless](#), MSK manages the broker nodes used to handle your traffic and you only provision your Kafka server resources at a cluster level.

- **ZooKeeper nodes** — Amazon MSK also creates the Apache ZooKeeper nodes for you. Apache ZooKeeper is an open-source server that enables highly reliable distributed coordination.
- **KRaft controllers** — The Apache Kafka community developed KRaft to replace Apache ZooKeeper for metadata management in Apache Kafka clusters. In KRaft mode, cluster metadata is propagated within a group of Kafka controllers, which are part of the Kafka cluster, instead of across ZooKeeper nodes. KRaft controllers are included at no additional cost to you, and require no additional setup or management from you.

 **Note**

From Apache Kafka version 3.7.x on MSK, you can create clusters that use KRaft mode instead of ZooKeeper mode.

- **Producers, consumers, and topic creators** — Amazon MSK lets you use Apache Kafka data-plane operations to create topics and to produce and consume data.
- **Cluster Operations** You can use the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the APIs in the SDK to perform control-plane operations. For example, you can create or delete an Amazon MSK cluster, list all the clusters in an account, view the properties of a cluster, and update the number and type of brokers in a cluster.

Amazon MSK detects and automatically recovers from the most common failure scenarios for clusters so that your producer and consumer applications can continue their write and read operations with minimal impact. When Amazon MSK detects a broker failure, it mitigates the failure or replaces the unhealthy or unreachable broker with a new one. In addition, where possible, it reuses the storage from the older broker to reduce the data that Apache Kafka needs to replicate. Your availability impact is limited to the time required for Amazon MSK to complete the detection and recovery. After a recovery, your producer and consumer apps can continue to communicate with the same broker IP addresses that they used before the failure.

Setting up Amazon MSK

Before you use Amazon MSK for the first time, complete the following tasks.

Tasks

- [Sign up for AWS](#)
- [Download libraries and tools](#)

Sign up for AWS

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

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

To sign up for an Amazon Web Services account

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

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

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

Download libraries and tools

The following libraries and tools can help you work with Amazon MSK:

- The [AWS Command Line Interface \(AWS CLI\)](#) supports Amazon MSK. The AWS CLI enables you to control multiple Amazon Web Services from the command line and automate them through scripts. Upgrade your AWS CLI to the latest version to ensure that it has support for the Amazon MSK features that are documented in this user guide. For detailed instructions on

how to upgrade the AWS CLI, see [Installing the AWS Command Line Interface](#). After you install the AWS CLI, you must configure it. For information on how to configure the AWS CLI, see [aws configure](#).

- The [Amazon Managed Streaming for Kafka API Reference](#) documents the API operations that Amazon MSK supports.
- The Amazon Web Services SDKs for [Go](#), [Java](#), [JavaScript](#), [.NET](#), [Node.js](#), [PHP](#), [Python](#), and [Ruby](#) include Amazon MSK support and samples.

MSK Provisioned

What is MSK Provisioned?

Amazon MSK Provisioned clusters offer a wide range of features and capabilities to help you optimize your cluster's performance and meet your streaming needs. The topics below describe the functionality in detail.

MSK Provisioned is an MSK cluster deployment option that allows you to manually configure and scale your Apache Kafka clusters. This provides you with varying levels of control over the infrastructure powering your Apache Kafka environment. With MSK Provisioned, you can choose the instance types, storage volumes (Standard brokers), and number of broker nodes that make up your Kafka clusters. You can also scale your cluster by adding or removing brokers as your data processing needs evolve. This flexibility enables you to optimize the clusters for your specific workload requirements, whether that's maximizing throughput, retention capacity, or other performance characteristics. In addition to the infrastructure configuration options, MSK Provisioned provides enterprise-grade security, monitoring, and operational benefits. This includes features such as Apache Kafka version upgrades, built-in security through encryption and access control, and integration with other AWS services such as Amazon CloudWatch for monitoring. MSK Provisioned offers two main broker types – Standard and Express.

For information about the MSK Provisioned API, see the [Amazon MSK API Reference](#).

Get started using Amazon MSK

This tutorial shows you an example of how you can create an MSK cluster, produce and consume data, and monitor the health of your cluster using metrics. This example doesn't represent all the options you can choose when you create an MSK cluster. In different parts of this tutorial, we choose default options for simplicity. This doesn't mean that they're the only options that work for setting up an MSK cluster or client instances.

Topics

- [Step 1: Create an MSK Provisioned cluster](#)
- [Step 2: Create an IAM role granting access to create topics on the Amazon MSK cluster](#)
- [Step 3: Create a client machine](#)
- [Step 4: Create a topic in the Amazon MSK cluster](#)

- [Step 5: Produce and consume data](#)
- [Step 6: Use Amazon CloudWatch to view Amazon MSK metrics](#)
- [Step 7: Delete the AWS resources created for this tutorial](#)

Step 1: Create an MSK Provisioned cluster

In this step of [Getting Started Using Amazon MSK](#), you create an Amazon MSK cluster.

To create an Amazon MSK cluster using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose **Create cluster**.
3. For **Creation method**, leave the **Quick create** option selected. The **Quick create** option lets you create a cluster with default settings.
4. For **Cluster name**, enter a descriptive name for your cluster. For example, **MSKTutorialCluster**.
5. For **General cluster properties**, do the following:
 - a. For **Cluster type**, choose **Provisioned**.
 - b. Choose an **Apache Kafka version** to run on the brokers. Choose **View version compatibility** to see a comparison table.
 - c. For **Broker type**, choose either Standard or Express brokers.
 - d. Choose a **Broker size**.
6. From the table under **All cluster settings**, copy the values of the following settings and save them because you need them later in this tutorial:
 - VPC
 - Subnets
 - Security groups associated with VPC
7. Choose **Create cluster**.
8. Check the cluster **Status** on the **Cluster summary** page. The status changes from **Creating** to **Active** as Amazon MSK provisions the cluster. When the status is **Active**, you can connect to the cluster. For more information about cluster status, see [Understand MSK Provisioned cluster states](#).

Next Step

[Step 2: Create an IAM role granting access to create topics on the Amazon MSK cluster](#)

Step 2: Create an IAM role granting access to create topics on the Amazon MSK cluster

In this step, you perform two tasks. The first task is to create an IAM policy that grants access to create topics on the cluster and to send data to those topics. The second task is to create an IAM role and associate this policy with it. In a later step, you create a client machine that assumes this role and uses it to create a topic on the cluster and to send data to that topic.

To create an IAM policy that makes it possible to create topics and write to them

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. On the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. In **Policy editor**, choose **JSON**, and then replace the JSON in the editor window with the following JSON.

In the following example, replace the following:

- *region* with the code of the AWS Region where you created your cluster.
- *Account-ID* with your AWS account ID.
- *MSKTutorialCluster* and *MSKTutorialCluster/7d7131e1-25c5-4e9a-9ac5-ea85bee4da11-14*, with the name of your cluster and its ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:Connect",
        "kafka-cluster:AlterCluster",
        "kafka-cluster:DescribeCluster"
      ],
      "Resource": [
```

```

        "arn:aws:kafka:region:Account-ID:cluster/MSKTutorialCluster/7d7131e1-25c5-4e9a-9ac5-ea85bee4da11-14"
    ],
    {
        "Effect": "Allow",
        "Action": [
            "kafka-cluster:*Topic*",
            "kafka-cluster:WriteData",
            "kafka-cluster:ReadData"
        ],
        "Resource": [
            "arn:aws:kafka:region:Account-ID:topic/MSKTutorialCluster/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "kafka-cluster:AlterGroup",
            "kafka-cluster:DescribeGroup"
        ],
        "Resource": [
            "arn:aws:kafka:region:Account-ID:group/MSKTutorialCluster/*"
        ]
    }
]
}

```

For instructions about how to write secure policies, see [the section called “IAM access control”](#).

5. Choose **Next**.
6. On the **Review and create** page, do the following:
 - a. For **Policy name**, enter a descriptive name, such as **msk-tutorial-policy**.
 - b. In **Permissions defined in this policy**, review and/or edit the permissions defined in your policy.
 - c. (Optional) To help identify, organize, or search for the policy, choose **Add new tag** to add tags as key-value pairs. For example, add a tag to your policy with the key-value pair of **Environment** and **Test**.

For more information about using tags, see [Tags for AWS Identity and Access Management resources](#) in the *IAM User Guide*.

7. Choose **Create policy**.

To create an IAM role and attach the policy to it

1. On the navigation pane, choose **Roles**, and then choose **Create role**.
2. On the **Select trusted entity** page, do the following:
 - a. For **Trusted entity type**, choose **AWS service**.
 - b. For **Service or use case**, choose **EC2**.
 - c. Under **Use case**, choose **EC2**.
3. Choose **Next**.
4. On the **Add permissions** page, do the following:
 - a. In the search box under **Permissions policies**, enter the name of the policy that you previously created for this tutorial. Then, choose the box to the left of the policy name.
 - b. (Optional) Set a [permissions boundary](#). This is an advanced feature that is available for service roles, but not service-linked roles. For information about setting a permissions boundary, see [Creating roles and attaching policies \(console\)](#) in the *IAM User Guide*.
5. Choose **Next**.
6. On the **Name, review, and create** page, do the following:
 - a. For **Role name**, enter a descriptive name, such as **msk-tutorial-role**.

Important

When you name a role, note the following:

- Role names must be unique within your AWS account, and can't be made unique by case.

For example, don't create roles named both **PRODRROLE** and **prodrole**. When a role name is used in a policy or as part of an ARN, the role name is case sensitive, however when a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive.

- You can't edit the name of the role after it's created because other entities might reference the role.

- b. (Optional) For **Description**, enter a description for the role.
- c. (Optional) To edit the use cases and permissions for the role, in **Step 1: Select trusted entities** or **Step 2: Add permissions** sections, choose **Edit**.
- d. (Optional) To help identify, organize, or search for the role, choose **Add new tag** to add tags as key-value pairs. For example, add a tag to your role with the key-value pair of **ProductManager** and **John**.

For more information about using tags, see [Tags for AWS Identity and Access Management resources](#) in the *IAM User Guide*.

7. Review the role, and then choose **Create role**.

Next Step

[Step 3: Create a client machine](#)

Step 3: Create a client machine

In this step of [Get Started Using Amazon MSK](#), you create a client machine. You use this client machine to create a topic that produces and consumes data. For simplicity, you'll create this client machine in the VPC that is associated with the MSK cluster so that the client can easily connect to the cluster.

To create a client machine

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the Amazon EC2 console dashboard, choose **Launch instance**.
3. Under **Name and tags**, for **Name**, enter a descriptive name for your client machine so that you can easily keep track of it. For example, **MSKTutorialClient**.
4. Under **Application and OS Images (Amazon Machine Image)**, for **Amazon Machine Image (AMI)**, choose **Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type**.
5. For **Instance type**, keep the default selection of **t2.micro**.
6. Under **Key pair (login)**, choose an existing key pair or create a new one. If you don't require a key pair to connect to your instance, you can choose **Proceed without a key pair (not recommended)**.

To create a new key pair, do the following:

- a. Choose **Create new key pair**.
- b. For **Key pair name**, enter **MSKKeyPair**.
- c. For **Key pair type** and **Private key file format**, keep the default selections.
- d. Choose **Create key pair**.

Alternatively, you can use an existing key pair.

7. Scroll down the page and expand the **Advanced details** section, then do the following:
 - For **IAM instance profile**, choose an IAM role that you want the client machine to assume.

If you don't have an IAM role, do the following:
 - i. Choose **Create new IAM profile**.
 - ii. Perform the steps mentioned in [Step 2: Create an IAM role](#).
8. Choose **Launch instance**.
9. Choose **View Instances**. Then, in the **Security Groups** column, choose the security group that is associated with your new instance. Copy the ID of the security group, and save it for later.
10. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
11. In the navigation pane, choose **Security Groups**. Find the security group whose ID you saved in [the section called "Create a cluster"](#).
12. In the **Inbound Rules** tab, choose **Edit inbound rules**.
13. Choose **Add rule**.
14. In the new rule, choose **All traffic** in the **Type** column. In the second field in the **Source** column, select the security group of your client machine. This is the group whose name you saved after you launched the client machine instance.
15. Choose **Save rules**. Now the cluster's security group can accept traffic that comes from the client machine's security group.

Next Step

[Step 4: Create a topic in the Amazon MSK cluster](#)

Step 4: Create a topic in the Amazon MSK cluster

In this step of [Getting Started Using Amazon MSK](#), you install Apache Kafka client libraries and tools on the client machine, and then you create a topic.

Warning

Apache Kafka version numbers used in this tutorial are examples only. We recommend that you use the same version of the client as your MSK cluster version. An older client version might be missing certain features and critical bug fixes.

To find the version of your MSK cluster

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. In the navigation bar, choose the Region where you created the MSK cluster.
3. Choose the MSK cluster.
4. Note the version of Apache Kafka used on the cluster.
5. Replace instances of Amazon MSK version numbers in this tutorial with the version obtained in Step 3.

To create a topic on the client machine

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, choose **Instances**. Then select the check box beside the name of the client machine that you created in [Step 3: Create a client machine](#).
3. Choose **Actions**, and then choose **Connect**. Follow the instructions in the console to connect to your client machine.
4. Install Java on the client machine by running the following command:

```
sudo yum -y install java-11
```

5. Run the following command to download Apache Kafka.

```
wget https://archive.apache.org/dist/kafka/{YOUR MSK VERSION}/kafka_2.13-{YOUR MSK VERSION}.tgz
```

For example, if you want to use Amazon MSK with Apache Kafka version 3.5.1, run the following command.

```
wget https://archive.apache.org/dist/kafka/3.5.1/kafka_2.13-3.5.1.tgz
```

 **Note**

If you want to use a mirror site other than the one used in this command, you can choose a different one on the [Apache](#) website.

6. Run the following command in the directory where you downloaded the TAR file in the previous step.

```
tar -xzf kafka_2.13-{YOUR MSK VERSION}.tgz
```

7. Go to the `kafka_2.13-{YOUR MSK VERSION}/libs` directory, then run the following command to download the Amazon MSK IAM JAR file. The Amazon MSK IAM JAR makes it possible for the client machine to access the cluster.

```
wget https://github.com/aws/aws-msk-iam-auth/releases/download/v2.3.0/aws-msk-iam-auth-2.3.0-all.jar
```

Using this command, you can also [download the latest version](#) of `aws-msk-iam-auth-*-all.jar`.

8. Go to the `kafka_2.13-{YOUR MSK VERSION}/config` directory. Copy the following property settings and paste them into a new file. Name the file **client.properties** and save it.

```
security.protocol=SASL_SSL
sasl.mechanism=AWS_MSK_IAM
sasl.jaas.config=software.amazon.msk.auth.iam.IAMLoginModule required;
sasl.client.callback.handler.class=software.amazon.msk.auth.iam.IAMClientCallbackHandler
```

9. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
10. Wait for the status of your cluster to become **Active**. This might take several minutes. After the status becomes **Active**, choose the cluster name. This takes you to a page containing the cluster summary.

11. Choose **View client information**.
12. Copy the connection string for the private endpoint.

You will get three endpoints for each of the brokers. You only need one broker endpoint for the following step.

13. Run the following command, replacing *BootstrapServerString* with one of the broker endpoints that you obtained in the previous step.

```
<path-to-your-kafka-installation>/bin/kafka-topics.sh --create --bootstrap-server  
BootstrapServerString --command-config client.properties --replication-factor 3 --  
partitions 1 --topic MSKTutorialTopic
```

Note

Before running this command, you must add the AWS MSK IAM JAR file to your classpath to avoid the `Class not found` error. You can do this by setting the `CLASSPATH` environment variable, as shown in the following command.

```
export CLASSPATH=<path-to-your-kafka-installation>/libs/aws-msk-iam-  
auth-2.3.0-all.jar
```

Alternatively, you can also modify your command to explicitly add the JAR to the classpath, as shown in the following example.

```
classpath=<path-to-your-kafka-installation>/libs/aws-msk-iam-auth-2.3.0-  
all.jar \  
<path-to-your-kafka-installation>/bin/kafka-topics.sh --create \  
--bootstrap-server BootstrapServerString \  
--command-config client.properties \  
--replication-factor 3 \  
--partitions 1 \  
--topic MSKTutorialTopic
```

For example, if Apache Kafka is installed in `/home/ec2-user/kafka_2.13-2.8.1`, and your broker endpoint is `myBrokerEndpoint-1.myCluster.abc123.kafka.us-east-1.amazonaws.com:9098`, you'd run the following command.

```
export CLASSPATH=/home/ec2-user/kafka_2.13-2.8.1/libs/aws-msk-iam-auth-2.3.0-  
all.jar  
/home/ec2-user/kafka_2.13-2.8.1/bin/kafka-topics.sh --create --bootstrap-server  
myBrokerEndpoint-1.myCluster.abc123.kafka.us-east-1.amazonaws.com:9098 --  
command-config client.properties --replication-factor 3 --partitions 1 --topic  
MSKTutorialTopic
```

If the command succeeds, you see the following message: Created topic MSKTutorialTopic.

If the command is unsuccessful or you run into an error, see [Troubleshoot your Amazon MSK cluster](#) for troubleshooting information.

Next Step

[Step 5: Produce and consume data](#)

Step 5: Produce and consume data

In this step of [Get Started Using Amazon MSK](#), you produce and consume data.

To produce and consume messages

1. Run the following command to start a console producer. Replace *BootstrapServerString* with the plaintext connection string that you obtained in [Create a topic](#). For instructions on how to retrieve this connection string, see [Getting the bootstrap brokers for an Amazon MSK cluster](#).

```
<path-to-your-kafka-installation>/bin/kafka-console-producer.sh --broker-  
list BootstrapServerString --producer.config <path-to-config>/client.properties --  
topic MSKTutorialTopic
```

2. Enter any message that you want, and press **Enter**. Repeat this step two or three times. Every time you enter a line and press **Enter**, that line is sent to your Apache Kafka cluster as a separate message.
3. Keep the connection to the client machine open, and then open a second, separate connection to that machine in a new window.

4. In the following command, replace *BootstrapServerString* with the plaintext connection string that you saved earlier. Then, to create a console consumer, run the following command with your second connection to the client machine.

```
<path-to-your-kafka-installation>/bin/kafka-console-consumer.sh --bootstrap-server BootstrapServerString --consumer.config <path-to-config>/client.properties --topic MSKTutorialTopic --from-beginning
```

You start seeing the messages you entered earlier when you used the console producer command.

5. Enter more messages in the producer window, and watch them appear in the consumer window.

Next Step

[Step 6: Use Amazon CloudWatch to view Amazon MSK metrics](#)

Step 6: Use Amazon CloudWatch to view Amazon MSK metrics

In this step of [Getting Started Using Amazon MSK](#), you look at the Amazon MSK metrics in Amazon CloudWatch.

To view Amazon MSK metrics in CloudWatch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **All metrics** tab, and then choose **AWS/Kafka**.
4. To view broker-level metrics, choose **Broker ID, Cluster Name**. For cluster-level metrics, choose **Cluster Name**.
5. (Optional) In the graph pane, select a statistic and a time period, and then create a CloudWatch alarm using these settings.

Next Step

[Step 7: Delete the AWS resources created for this tutorial](#)

Step 7: Delete the AWS resources created for this tutorial

In the final step of [Getting Started Using Amazon MSK](#), you delete the MSK cluster and the client machine that you created for this tutorial.

To delete the resources using the AWS Management Console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. Choose the name of your cluster. For example, **MSKTutorialCluster**.
3. Choose **Actions**, then choose **Delete**.
4. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
5. Choose the instance that you created for your client machine, for example, **MSKTutorialClient**.
6. Choose **Instance state**, then choose **Terminate instance**.

To delete the IAM policy and role

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. On the navigation pane, choose **Roles**.
3. In the search box, enter the name of the IAM role that you created for this tutorial.
4. Choose the role. Then choose **Delete role**, and confirm the deletion.
5. On the navigation pane, choose **Policies**.
6. In the search box, enter the name of the policy that you created for this tutorial.
7. Choose the policy to open its summary page. On the policy's **Summary** page, choose **Delete policy**.
8. Choose **Delete**.

Amazon MSK: How it works

Amazon MSK is a fully managed Apache Kafka service that makes it easy to build and run applications that use Apache Kafka to process streaming data. This guide provides information to help developers understand how Amazon MSK works and how to use it effectively in their applications.

At a high level, Amazon MSK provides a fully managed Apache Kafka cluster that is provisioned and operated by AWS. This means that you don't have to worry about provisioning EC2 instances, configuring network settings, managing Kafka brokers, or performing ongoing maintenance tasks. Instead, you can focus on building your application and let Amazon MSK handle the infrastructure. Amazon MSK automatically provisions the necessary compute, storage, and network resources, and provides features like automatic scaling, high availability, and failover to ensure that your Kafka cluster is reliable and highly available. This guide covers the key components of Amazon MSK and how you can use it to build streaming data applications.

Manage your Provisioned cluster

An Amazon MSK cluster is the primary Amazon MSK resource that you can create in your account. The topics in this section describe how to perform common Amazon MSK operations. For a list of all the operations that you can perform on an MSK cluster, see the following:

- The [AWS Management Console](#)
- The [Amazon MSK API Reference](#)
- The [Amazon MSK CLI Command Reference](#)

Topics

- [Create an MSK Provisioned cluster](#)
- [List Amazon MSK clusters](#)
- [Connect to an Amazon MSK Provisioned cluster](#)
- [Get the bootstrap brokers for an Amazon MSK cluster](#)
- [Monitor an Amazon MSK Provisioned cluster](#)
- [Update security settings of a Amazon MSK cluster](#)
- [Expand the number of brokers in an Amazon MSK cluster](#)
- [Remove a broker from an Amazon MSK cluster](#)
- [Provision storage throughput for Standard brokers in a Amazon MSK cluster](#)
- [Update the Amazon MSK cluster broker size](#)
- [Use LinkedIn's Cruise Control for Apache Kafka with Amazon MSK](#)
- [Update the configuration of an Amazon MSK cluster](#)
- [Reboot a broker for an Amazon MSK cluster](#)

- [Tag an Amazon MSK cluster](#)
- [Migrate to an Amazon MSK Cluster](#)
- [Delete an Amazon MSK Provisioned cluster](#)

Create an MSK Provisioned cluster

Important

You can't change the VPC for an MSK Provisioned cluster after you create the cluster.

Before you can create an MSK Provisioned cluster, you need to have an Amazon Virtual Private Cloud (VPC) and set up subnets within that VPC.

For Standard brokers in the US West (N. California) Region, you need two subnets in two different Availability Zones. In all other Regions where Amazon MSK is available, you can specify either two or three subnets. Your subnets must all be in different Availability Zones. For Express brokers, you need three subnets in three different Availability Zones. When you create an MSK Provisioned cluster, Amazon MSK distributes the broker nodes evenly over the subnets that you specify.

Topics

- [Create an MSK Provisioned cluster using the AWS Management Console](#)
- [Create a provisioned Amazon MSK cluster using the AWS CLI](#)
- [Create an MSK Provisioned cluster with a custom Amazon MSK configuration using the AWS CLI](#)
- [Create an MSK Provisioned cluster using the Amazon MSK API](#)

Create an MSK Provisioned cluster using the AWS Management Console

This process describes the common task of creating an MSK Provisioned cluster using custom create options in the AWS Management Console. You can select other options in the AWS Management Console to create a serverless cluster.

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose **Create cluster**.

3. For **Cluster creation method**, choose **Custom create**.
4. Specify a **Cluster name** that is unique and no more than 64 characters.
5. For **General cluster properties**, choose **Provisioned** as the **Cluster type**.
6. Select the **Apache Kafka version** to run on the brokers. To see a comparison of Amazon MSK features that are supported by each Apache Kafka version, click **View version compatibility**.
7. Choose either **Express brokers** or **Standard brokers** broker type.
8. Select a **broker size** to use for the cluster based on the cluster's compute, memory, and storage needs. See [Amazon MSK broker types](#),
9. Select the **Number of zones** across which brokers are distributed. Express brokers require 3 Availability Zones for higher availability.
10. Specify the number of brokers you want MSK to create in each Availability Zone. The minimum is one broker per Availability Zone and the maximum is 30 brokers per cluster for ZooKeeper-based clusters and 60 brokers per cluster for [KRaft-based clusters](#).
11. (Standard brokers only) Select the initial amount of **Storage** you want your cluster to have. You can't decrease storage capacity after you create the cluster. You don't need to manage storage for Express brokers.
12. (Standard brokers only) Depending on the broker size (instance size) you selected, you can specify **Provisioned storage throughput per broker**. To enable this option, choose broker size (instance size) kafka.m5.4xlarge or larger for x86, and kafka.m7g.2xlarge or larger for Graviton-based instances. See [???](#).
13. (Standard brokers only) Select a **Cluster storage mode** option, either EBS storage only or tiered storage and EBS storage. Express brokers don't require you to manage storage.
14. If you want to create and use a custom **Cluster configuration** (or if you already have a cluster configuration saved), choose a configuration. Otherwise, you can create the cluster using the **Amazon MSK default cluster configuration**. For information about Amazon MSK configurations, see [the section called "Broker configuration"](#).
15. Select **Next**.
16. For Networking settings, choose the VPC you want to use for the cluster.
17. Based on the **Number of zones** you previously selected, specify the Availability Zones and subnets where brokers will deploy. For Standard brokers in the US West (N. California) Region, you need two subnets in two different Availability Zones. In all other Regions where Amazon MSK is available, you can specify either two or three subnets. Your subnets must all be in different Availability Zones. For Express brokers, you need three subnets in three different

- Availability Zones. When you create an MSK Provisioned cluster, MSK distributes the broker nodes evenly over the subnets that you specify.
18. You can select one or more security groups that you want to give access to your cluster (for example, the security groups of client machines). If you specify security groups that are shared with you, you must ensure that you have permissions to use them. Specifically, you need the `ec2:DescribeSecurityGroups` permission. [Connecting to an MSK cluster](#).
 19. Select **Next**.
 20. Select the cluster's **Access control methods** and **Encryption settings** for encrypting data as it transits between clients and brokers. For more information, see [the section called "Amazon MSK encryption in transit"](#).
 21. Choose the kind of KMS key that you want to use for encrypting data at rest. For more information, see [the section called "Amazon MSK encryption at rest"](#).
 22. Select **Next**.
 23. Choose the **Monitoring and tags** you want. This determines the set of metrics you get. For more information, see [the section called "Monitor a cluster"](#). [Amazon CloudWatch](#), [Prometheus](#), [Broker log delivery](#), or [Cluster tags](#), then select **Next**.
 24. Review the settings for your cluster. You can go back and change settings by selecting **Previous** to go back to the previous console screen, or **Edit** to change specific cluster settings. If the settings are correct, select **Create cluster**.
 25. Check the cluster **Status** on the **Cluster summary** page. The status changes from **Creating** to **Active** as Amazon MSK provisions the cluster. When the status is **Active**, you can connect to the cluster. For more information about cluster status, see [Understand MSK Provisioned cluster states](#).

Create a provisioned Amazon MSK cluster using the AWS CLI

1. Copy the following JSON and save it to a file. Name the file `brokernodegroupinfo.json`. Replace the subnet IDs in the JSON with the values that correspond to your subnets. These subnets must be in different Availability Zones. Replace *"Security-Group-ID"* with the ID of one or more security groups of the client VPC. Clients associated with these security groups get access to the cluster. If you specify security groups that were shared with you, you must ensure that you have permissions to them. Specifically, you need the `ec2:DescribeSecurityGroups` permission. For an example, see [Amazon EC2: Allows Managing Amazon EC2 Security Groups Associated With a Specific VPC, Programmatically and](#)

[in the Console](#). Finally, save the updated JSON file on the computer where you have the AWS CLI installed.

```
{
  "InstanceType": "kafka.m5.large",
  "ClientSubnets": [
    "Subnet-1-ID",
    "Subnet-2-ID"
  ],
  "SecurityGroups": [
    "Security-Group-ID"
  ]
}
```

Important

For Express brokers, you need three subnets in three different Availability Zones. You also do not need to define any storage-related properties.

For Standard brokers in the US West (N. California) Region, you need two subnets in two different Availability Zones. In all other Regions where Amazon MSK is available, you can specify either two or three subnets. Your subnets must all be in different Availability Zones. When you create a cluster, Amazon MSK distributes the broker nodes evenly over the subnets that you specify.

2. Run the following AWS CLI command in the directory where you saved the `brokernodegroupinfo.json` file, replacing *"Your-Cluster-Name"* with a name of your choice. For *"Monitoring-Level"*, you can specify one of the following three values: `DEFAULT`, `PER_BROKER`, or `PER_TOPIC_PER_BROKER`. For information about these three different levels of monitoring, see [???](#). The enhanced-monitoring parameter is optional. If you don't specify it in the `create-cluster` command, you get the `DEFAULT` level of monitoring.

```
aws kafka create-cluster --cluster-name "Your-Cluster-Name" --broker-node-group-
info file://brokernodegroupinfo.json --kafka-version "2.8.1" --number-of-broker-
nodes 3 --enhanced-monitoring "Monitoring-Level"
```

The output of the command looks like the following JSON:

```
{
```

```
"ClusterArn": "...",  
"ClusterName": "AWSKafkaTutorialCluster",  
"State": "CREATING"  
}
```

Note

The `create-cluster` command might return an error stating that one or more subnets belong to unsupported Availability Zones. When this happens, the error indicates which Availability Zones are unsupported. Create subnets that don't use the unsupported Availability Zones and try the `create-cluster` command again.

3. Save the value of the `ClusterArn` key because you need it to perform other actions on your cluster.
4. Run the following command to check your cluster `STATE`. The `STATE` value changes from `CREATING` to `ACTIVE` as Amazon MSK provisions the cluster. When the state is `ACTIVE`, you can connect to the cluster. For more information about cluster status, see [Understand MSK Provisioned cluster states](#).

```
aws kafka describe-cluster --cluster-arn <your-cluster-ARN>
```

Create an MSK Provisioned cluster with a custom Amazon MSK configuration using the AWS CLI

For information about custom Amazon MSK configurations and how to create them, see [the section called "Broker configuration"](#).

1. Save the following JSON to a file, replacing *configuration-arn* with the ARN of the configuration that you want to use to create the cluster.

```
{  
  "Arn": configuration-arn,  
  "Revision": 1  
}
```

2. Run the `create-cluster` command and use the `configuration-info` option to point to the JSON file you saved in the previous step. The following is an example.

```
aws kafka create-cluster --cluster-name ExampleClusterName --broker-node-group-info file://brokernodegroupinfo.json --kafka-version "2.8.1" --number-of-broker-nodes 3 --enhanced-monitoring PER_TOPIC_PER_BROKER --configuration-info file://configuration.json
```

The following is an example of a successful response after running this command.

```
{
  "ClusterArn": "arn:aws:kafka:us-east-1:123456789012:cluster/CustomConfigExampleCluster/abcd1234-abcd-dcba-4321-a1b2abcd9f9f-2",
  "ClusterName": "CustomConfigExampleCluster",
  "State": "CREATING"
}
```

Create an MSK Provisioned cluster using the Amazon MSK API

The Amazon MSK API allows you to programmatically create and manage your MSK Provisioned cluster as part of automated infrastructure provisioning or deployment scripts.

To create an MSK Provisioned cluster using the API, see [CreateCluster](#).

List Amazon MSK clusters

To get a bootstrap broker for an Amazon MSK cluster, you need the cluster Amazon Resource Name (ARN). If you don't have the ARN for your cluster, you can find it by listing all clusters. See [the section called "Get the bootstrap brokers"](#).

Topics

- [List clusters using the AWS Management Console](#)
- [List clusters using the AWS CLI](#)
- [List clusters using the API](#)

List clusters using the AWS Management Console

To get a bootstrap broker for an Amazon MSK cluster, you need the cluster Amazon Resource Name (ARN). If you don't have the ARN for your cluster, you can find it by listing all clusters. See [the section called "Get the bootstrap brokers"](#).

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. The table shows all the clusters for the current region under this account. Choose the name of a cluster to view its details.

List clusters using the AWS CLI

To get a bootstrap broker for an Amazon MSK cluster, you need the cluster Amazon Resource Name (ARN). If you don't have the ARN for your cluster, you can find it by listing all clusters. See [the section called "Get the bootstrap brokers"](#).

```
aws kafka list-clusters
```

List clusters using the API

To get a bootstrap broker for an Amazon MSK cluster, you need the cluster Amazon Resource Name (ARN). If you don't have the ARN for your cluster, you can find it by listing all clusters. See [the section called "Get the bootstrap brokers"](#).

To list clusters using the API, see [ListClusters](#).

Connect to an Amazon MSK Provisioned cluster

By default, clients can access an MSK Provisioned cluster only if they're in the same VPC as the cluster. All communication between your Kafka clients and your MSK Provisioned cluster are private by default and your streaming data never traverses the internet. To connect to your MSK Provisioned cluster from a client that's in the same VPC as the cluster, make sure the cluster's security group has an inbound rule that accepts traffic from the client's security group. For information about setting up these rules, see [Security Group Rules](#). For an example of how to access a cluster from an Amazon EC2 instance that's in the same VPC as the cluster, see [the section called "Get started"](#).

Note

KRaft metadata mode and MSK Express brokers can't have open monitoring and public access both enabled.

To connect to your MSK Provisioned cluster from a client that's outside the cluster's VPC, see [Access from within AWS but outside cluster's VPC](#).

Topics

- [Turn on public access to an MSK Provisioned cluster](#)
- [Access from within AWS but outside cluster's VPC](#)

Turn on public access to an MSK Provisioned cluster

Amazon MSK gives you the option to turn on public access to the brokers of MSK Provisioned clusters running Apache Kafka 2.6.0 or later versions. For security reasons, you can't turn on public access while creating an MSK cluster. However, you can update an existing cluster to make it publicly accessible. You can also create a new cluster and then update it to make it publicly accessible.

You can turn on public access to an MSK cluster at no additional cost, but standard AWS data transfer costs apply for data transfer in and out of the cluster. For information about pricing, see [Amazon EC2 On-Demand Pricing](#).

Note

If you're using the SASL/SCRAM, or mTLS access-control methods, you must first set Apache Kafka ACLs for your cluster. Then, update the cluster's configuration to set the `allow.everyone.if.no.acl.found` property to `false`. For information about how to update the configuration of a cluster, see [the section called "Broker configuration operations"](#).

To turn on public access to an MSK Provisioned cluster, make sure that the cluster meets all of the following conditions:

- The subnets that are associated with the cluster must be public. Each public subnet has a public IPv4 address associated with it and public IPv4 addresses are priced as shown in [Amazon VPC pricing page](#). This means that the subnets must have an associated route table with an internet gateway attached. For information about how to create and attach an internet gateway, see [Enable VPC internet access using internet gateways](#) in the *Amazon VPC User Guide*.

- Unauthenticated access control must be off and at least one of the following access-control methods must be on: SASL/IAM, SASL/SCRAM, mTLS. For information about how to update the access-control method of a cluster, see [the section called “Update cluster security”](#).
- Encryption within the cluster must be turned on. The on setting is the default when creating a cluster. It's not possible to turn on encryption within the cluster for a cluster that was created with it turned off. It is therefore not possible to turn on public access for a cluster that was created with encryption within the cluster turned off.
- Plaintext traffic between brokers and clients must be off. For information about how to turn it off if it's on, see [the section called “Update cluster security”](#).
- If you're using IAM access control and want to apply authorization policies or update your authorization policies, see [the section called “IAM access control”](#). For information about Apache Kafka ACLs, see [the section called “Apache Kafka ACLs”](#).

After you ensure that an MSK cluster meets the conditions listed above, you can use the AWS Management Console, the AWS CLI, or the Amazon MSK API to turn on public access. After you turn on public access to a cluster, you can get a public bootstrap-brokers string for it. For information about getting the bootstrap brokers for a cluster, see [the section called “Get the bootstrap brokers”](#).

Important

In addition to turning on public access, ensure that the cluster's security groups have inbound TCP rules that allow public access from your IP address. We recommend that you make these rules as restrictive as possible. For information about security groups and inbound rules, see [Security groups for your VPC](#) in the Amazon VPC User Guide. For port numbers, see [the section called “Port information”](#). For instructions on how to change a cluster's security group, see [the section called “Changing security groups”](#).

Note

If you use the following instructions to turn on public access and then still cannot access the cluster, see [the section called “Unable to access cluster that has public access turned on”](#).

Turning on public access using the console

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. In the list of clusters, choose the cluster to which you want to turn on public access.
3. Choose the **Properties** tab, then find the **Network settings** section.
4. Choose **Edit public access**.

Turning on public access using the AWS CLI

1. Run the following AWS CLI command, replacing *ClusterArn* and *Current-Cluster-Version* with the ARN and current version of the cluster. To find the current version of the cluster, use the [DescribeCluster](#) operation or the [describe-cluster](#) AWS CLI command. An example version is KTVDPKIKX0DER.

```
aws kafka update-connectivity --cluster-arn ClusterArn --current-  
version Current-Cluster-Version --connectivity-info '{"PublicAccess": {"Type":  
"SERVICE_PROVIDED_EIPS"}}'
```

The output of this update-connectivity command looks like the following JSON example.

```
{  
  "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/  
abcdefab-1234-abcd-5678-cdef0123ab01-2",  
  "ClusterOperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-  
operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-  
abcd-4f7f-1234-9876543210ef"  
}
```

Note

To turn off public access, use a similar AWS CLI command, but with the following connectivity info instead:

```
'{"PublicAccess": {"Type": "DISABLED"}}'
```

2. To get the result of the update-connectivity operation, run the following command, replacing *ClusterOperationArn* with the ARN that you obtained in the output of the update-connectivity command.

```
aws kafka describe-cluster-operation --cluster-operation-arn ClusterOperationArn
```

The output of this describe-cluster-operation command looks like the following JSON example.

```
{
  "ClusterOperationInfo": {
    "ClientRequestId": "982168a3-939f-11e9-8a62-538df00285db",
    "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/
exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2",
    "CreationTime": "2019-06-20T21:08:57.735Z",
    "OperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
abcd-4f7f-1234-9876543210ef",
    "OperationState": "UPDATE_COMPLETE",
    "OperationType": "UPDATE_CONNECTIVITY",
    "SourceClusterInfo": {
      "ConnectivityInfo": {
        "PublicAccess": {
          "Type": "DISABLED"
        }
      }
    },
    "TargetClusterInfo": {
      "ConnectivityInfo": {
        "PublicAccess": {
          "Type": "SERVICE_PROVIDED_EIPS"
        }
      }
    }
  }
}
```

If *OperationState* has the value *UPDATE_IN_PROGRESS*, wait a while, then run the describe-cluster-operation command again.

Turning on public access using the Amazon MSK API

- To use the API to turn public access to a cluster on or off, see [UpdateConnectivity](#).

Note

For security reasons, Amazon MSK doesn't allow public access to Apache ZooKeeper or KRaft controller nodes.

Access from within AWS but outside cluster's VPC

To connect to an MSK cluster from inside AWS but outside the cluster's Amazon VPC, the following options exist.

Amazon VPC peering

To connect to your MSK cluster from a VPC that's different from the cluster's VPC, you can create a peering connection between the two VPCs. For information about VPC peering, see the [Amazon VPC Peering Guide](#).

AWS Direct Connect

AWS Direct Connect links your on-premise network to AWS over a standard 1 gigabit or 10 gigabit Ethernet fiber-optic cable. One end of the cable is connected to your router, the other to an AWS Direct Connect router. With this connection in place, you can create virtual interfaces directly to the AWS cloud and Amazon VPC, bypassing Internet service providers in your network path. For more information, see [AWS Direct Connect](#).

AWS Transit Gateway

AWS Transit Gateway is a service that enables you to connect your VPCs and your on-premises networks to a single gateway. For information about how to use AWS Transit Gateway, see [AWS Transit Gateway](#).

VPN connections

You can connect your MSK cluster's VPC to remote networks and users using the VPN connectivity options described in the following topic: [VPN Connections](#).

REST proxies

You can install a REST proxy on an instance running within your cluster's Amazon VPC. REST proxies enable your producers and consumers to communicate with the cluster through HTTP API requests.

Multiple Region multi-VPC connectivity

The following document describes connectivity options for multiple VPCs that reside in different Regions: [Multiple Region Multi-VPC Connectivity](#).

Single Region multi-VPC private connectivity

Multi-VPC private connectivity (powered by [AWS PrivateLink](#)) for Amazon Managed Streaming for Apache Kafka (Amazon MSK) clusters is a feature that enables you to more quickly connect Kafka clients hosted in different Virtual Private Clouds (VPCs) and AWS accounts to an Amazon MSK cluster.

See [Single Region multi-VPC connectivity for cross-account clients](#).

EC2-Classic networking is retired

Amazon MSK no longer supports Amazon EC2 instances running with Amazon EC2-Classic networking.

See [EC2-Classic Networking is Retiring – Here's How to Prepare](#).

Amazon MSK multi-VPC private connectivity in a single Region

Multi-VPC private connectivity (powered by [AWS PrivateLink](#)) for Amazon Managed Streaming for Apache Kafka (Amazon MSK) clusters is a feature that enables you to more quickly connect Kafka clients hosted in different Virtual Private Clouds (VPCs) and AWS accounts to an Amazon MSK cluster.

Multi-VPC private connectivity is a managed solution that simplifies the networking infrastructure for multi-VPC and cross-account connectivity. Clients can connect to the Amazon MSK cluster over PrivateLink while keeping all traffic within the AWS network. Multi-VPC private connectivity for Amazon MSK clusters is available in all AWS Regions where Amazon MSK is available.

Topics

- [What is multi-VPC private connectivity?](#)

- [Benefits of multi-VPC private connectivity](#)
- [Requirements and limitations for multi-VPC private connectivity](#)
- [Get started using multi-VPC private connectivity](#)
- [Update the authorization schemes on a cluster](#)
- [Reject a managed VPC connection to an Amazon MSK cluster](#)
- [Delete a managed VPC connection to an Amazon MSK cluster](#)
- [Permissions for multi-VPC private connectivity](#)

What is multi-VPC private connectivity?

Multi-VPC private connectivity for Amazon MSK is a connectivity option that enables you to connect Apache Kafka clients that are hosted in different Virtual Private Clouds (VPCs) and AWS accounts to a MSK cluster.

Amazon MSK simplifies cross-account access with [cluster policies](#). These policies allow the cluster owner to grant permissions for other AWS accounts to establish private connectivity to the MSK cluster.

Benefits of multi-VPC private connectivity

Multi-VPC private connectivity has several advantages over [other connectivity solutions](#):

- It automates operational management of the AWS PrivateLink connectivity solution.
- It allows overlapping IPs across connecting VPCs, eliminating the need to maintain non-overlapping IPs, complex peering, and routing tables associated with other VPC connectivity solutions.

You use a cluster policy for your MSK cluster to define which AWS accounts have permissions to set up cross-account private connectivity to your MSK cluster. The cross-account admin can delegate permissions to appropriate roles or users. When used with IAM client authentication, you can also use the cluster policy to define Kafka data plane permissions on a granular basis for the connecting clients.

Requirements and limitations for multi-VPC private connectivity

Note these MSK cluster requirements for running multi-VPC private connectivity:

- Multi-VPC private connectivity is supported only on Apache Kafka 2.7.1 or higher. Make sure that any clients that you use with the MSK cluster are running Apache Kafka versions that are compatible with the cluster.
- Multi-VPC private connectivity supports auth types IAM, TLS and SASL/SCRAM. Unauthenticated clusters can't use multi-VPC private connectivity.
- If you are using the SASL/SCRAM or mTLS access-control methods, you must set Apache Kafka ACLs for your cluster. First, set the Apache Kafka ACLs for your cluster. Then, update the cluster's configuration to have the property `allow.everyone.if.no.acl.found` set to `false` for the cluster. For information about how to update the configuration of a cluster, see [the section called "Broker configuration operations"](#). If you are using IAM access control and want to apply authorization policies or update your authorization policies, see [the section called "IAM access control"](#). For information about Apache Kafka ACLs, see [the section called "Apache Kafka ACLs"](#).
- Multi-VPC private connectivity doesn't support the `t3.small` instance type.
- Multi-VPC private connectivity isn't supported across AWS Regions, only on AWS accounts within the same Region.
- To set up multi-VPC private connectivity, you must have the same number of client subnets as cluster subnets. You must also make sure that [Availability Zone IDs](#) are same for the client subnet and cluster subnet.
- Amazon MSK doesn't support multi-VPC private connectivity to Zookeeper nodes.

Get started using multi-VPC private connectivity

Topics

- [Step 1: On the MSK cluster in Account A, turn on multi-VPC connectivity for IAM auth scheme on the cluster](#)
- [Step 2: Attach a cluster policy to the MSK cluster](#)
- [Step 3: Cross-account user actions to configure client-managed VPC connections](#)

This tutorial uses a common use case as an example of how you can use multi-VPC connectivity to privately connect an Apache Kafka client to an MSK cluster from inside AWS, but outside VPC of the cluster. This process requires the cross-account user to create a MSK managed VPC connection and configuration for each client, including required client permissions. The process also requires the MSK cluster owner to enable PrivateLink connectivity on the MSK cluster and select authentication schemes to control access to the cluster.

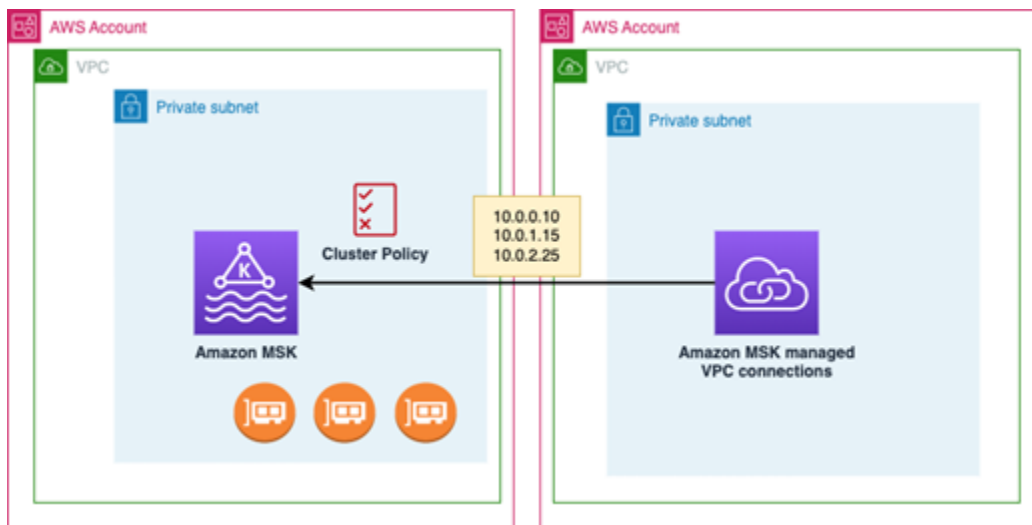
In different parts of this tutorial, we choose options that apply to this example. This doesn't mean that they're the only options that work for setting up an MSK cluster or client instances.

The network configuration for this use case is as follows:

- A cross-account user (Kafka client) and an MSK cluster are in the same AWS network/Region, but in different accounts:
- MSK cluster in Account A
- Kafka client in Account B
- The cross-account user will connect privately to the MSK cluster using IAM auth scheme.

This tutorial assumes that there is a provisioned MSK cluster created with Apache Kafka version 2.7.1 or higher. The MSK cluster must be in an ACTIVE state before beginning the configuration process. To avoid potential data loss or downtime, clients that will use multi-VPC private connection to connect to the cluster should use Apache Kafka versions that are compatible with the cluster.

The following diagram illustrates the architecture of Amazon MSK multi-VPC connectivity connected to a client in a different AWS account.



Step 1: On the MSK cluster in Account A, turn on multi-VPC connectivity for IAM auth scheme on the cluster

The MSK cluster owner needs to make configuration settings on the MSK cluster after the cluster is created and in an ACTIVE state.

The cluster owner turns on multi-VPC private connectivity on the ACTIVE cluster for any auth schemes that will be active on the cluster. This can be done using the [UpdateSecurity API](#) or MSK console. The IAM, SASL/SCRAM, and TLS auth schemes support multi-VPC private connectivity. Multi-VPC private connectivity can't be enabled for unauthenticated clusters.

For this use case, you'll configure the cluster to use the IAM auth scheme.

 **Note**

If you are configuring your MSK cluster to use SASL/SCRAM auth scheme, the Apache Kafka ACLs property `"allow.everyone.if.no.acl.found=false"` is mandatory. See [Apache Kafka ACLs](#).

When you update multi-VPC private connectivity settings, Amazon MSK starts a rolling reboot of broker nodes that updates the broker configurations. This can take up to 30 minutes or more to complete. You can't make other updates to the cluster while connectivity is being updated.

Turn on multi-VPC for selected auth schemes on the cluster in Account A using the console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/> for the account where the cluster is located.
2. In the navigation pane, under **MSK Clusters**, choose **Clusters** to display the list of clusters in the account.
3. Select the cluster to configure for multi-VPC private connectivity. The cluster must be in an ACTIVE state.
4. Select the cluster **Properties** tab, and then go to **Network** settings.
5. Select the **Edit** drop down menu and select **Turn on multi-VPC connectivity**.
6. Select one or more authentication types you want turned on for this cluster. For this use case, select **IAM role-based authentication**.
7. Select **Save changes**.

Example - UpdateConnectivity API that turns on Multi-VPC private connectivity auth schemes on a cluster

As an alternative to the MSK console, you can use the [UpdateConnectivity API](#) to turn on multi-VPC private connectivity and configure auth schemes on an ACTIVE cluster. The following example shows the IAM auth scheme turned on for the cluster.

```
{
  "currentVersion": "K3T4TT2Z381HKD",
  "connectivityInfo": {
    "vpcConnectivity": {
      "clientAuthentication": {
        "sasl": {
          "iam": {
            "enabled": TRUE
          }
        }
      }
    }
  }
}
```

Amazon MSK creates the networking infrastructure required for private connectivity. Amazon MSK also creates a new set of bootstrap broker endpoints for each auth type that requires private connectivity. Note that the plaintext auth scheme does not support multi-VPC private connectivity.

Step 2: Attach a cluster policy to the MSK cluster

The cluster owner can attach a cluster policy (also known as a [resource-based policy](#)) to the MSK cluster where you will turn on multi-VPC private connectivity. The cluster policy gives the clients permission to access the cluster from another account. Before you can edit the cluster policy, you need the account ID(s) for the accounts that should have permission to access the MSK cluster. See [How Amazon MSK works with IAM](#).

The cluster owner must attach a cluster policy to the MSK cluster that authorizes the cross-account user in Account B to get bootstrap brokers for the cluster and to authorize the following actions on the MSK cluster in Account A:

- CreateVpcConnection
- GetBootstrapBrokers
- DescribeCluster

- DescribeClusterV2

Example

For reference, the following is an example of the JSON for a basic cluster policy, similar to the default policy shown in the MSK console IAM policy editor. The following policy grants permissions for cluster, topic, and group-level access.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": [
        "kafka:CreateVpcConnection",
        "kafka:GetBootstrapBrokers",
        "kafka:DescribeCluster",
        "kafka:DescribeClusterV2",
        "kafka-cluster:*"
      ],
      "Resource": "arn:aws:kafka:us-east-1:111122223333:cluster/testing/de8982fa-8222-4e87-8b20-9bf3cdfa1521-2"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": "kafka-cluster:*",
      "Resource": "arn:aws:kafka:us-east-1:111122223333:topic/testing/*"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": "kafka-cluster:*",
      "Resource": "arn:aws:kafka:us-east-1:111122223333:group/testing/*"
    }
  ]
}
```

```
}
```

Attach a cluster policy to the MSK cluster

1. In the Amazon MSK console, under **MSK Clusters**, choose **Clusters**.
2. Scroll down to **Security settings** and select **Edit cluster policy**.
3. In the console, on the **Edit Cluster Policy** screen, select **Basic policy for multi-VPC connectivity**.
4. In the **Account ID** field, enter the account ID for each account that should have permission to access this cluster. As you type the ID, it is automatically copied over into the displayed policy JSON syntax. In our example cluster policy, the Account ID is 123456789012.
5. Select **Save changes**.

For information about cluster policy APIs, see [Amazon MSK resource-based policies](#).

Step 3: Cross-account user actions to configure client-managed VPC connections

To set up multi-VPC private connectivity between a client in a different account from the MSK cluster, the cross-account user creates a managed VPC connection for the client. Multiple clients can be connected to the MSK cluster by repeating this procedure. For the purposes of this use case, you'll configure just one client.

Clients can use the supported auth schemes IAM, SASL/SCRAM, or TLS. Each managed VPC connection can have only one auth scheme associated with it. The client auth scheme must be configured on the MSK cluster where the client will connect.

For this use case, configure the client auth scheme so that the client in Account B uses the IAM auth scheme.

Prerequisites

This process requires the following items:

- The previously created cluster policy that grants the client in Account B permission to perform actions on the MSK cluster in Account A.
- An identity policy attached to the client in Account B that grants permissions for `kafka:CreateVpcConnection`, `ec2:CreateTags`, `ec2:CreateVPCEndpoint` and `ec2:DescribeVpcAttribute` action.

Example

For reference, the following is an example of the JSON for a basic client identity policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kafka:CreateVpcConnection",
        "ec2:CreateTags",
        "ec2:CreateVPCEndpoint",
        "ec2:DescribeVpcAttribute"
      ],
      "Resource": "*"
    }
  ]
}
```

To create a managed VPC connection for a client in Account B

1. From the cluster administrator, get the **Cluster ARN** of the MSK cluster in Account A that you want the client in Account B to connect to. Make note of the cluster ARN to use later.
2. In the MSK console for the client Account B, choose **Managed VPC connections**, and then choose **Create connection**.
3. In the **Connection settings** pane, paste the cluster ARN into the cluster ARN text field, and then choose **Verify**.
4. Select the **Authentication type** for the client in Account B. For this use case, choose IAM when creating the client VPC connection.
5. Choose the **VPC** for the client.
6. Choose at least two availability **Zones** and associated **Subnets**. You can get the availability zone IDs from the AWS Management Console cluster details or by using the [DescribeCluster](#) API or the [describe-cluster](#) AWS CLI command. The zone IDs that you specify for the client subnet must match those of the cluster subnet. If the values for a subnet are missing, first create a subnet with the same zone ID as your MSK cluster.
7. Choose a **Security group** for this VPC connection. You can use the default security group. For more information on configuring a security group, see [Control traffic to resources using security groups](#).

8. Select **Create connection**.
9. To get the list of new bootstrap broker strings from the cross-account user's MSK console (**Cluster details** > **Managed VPC connection**), see the bootstrap broker strings shown under "**Cluster connection string**." From the client Account B, the list of bootstrap brokers can be viewed by calling the [GetBootstrapBrokers](#) API or by viewing the list of bootstrap brokers in the console cluster details.
10. Update the security groups associated with the VPC connections as follows:
 - a. Set **inbound rules** for the PrivateLink VPC to allow all traffic for the IP range from the Account B network.
 - b. [Optional] Set **Outbound rules** connectivity to the MSK cluster. Choose the **Security Group** in the VPC console, **Edit Outbound Rules**, and add a rule for **Custom TCP Traffic** for port ranges 14001-14100. The multi-VPC network load balancer is listening on the 14001-14100 port ranges. See [Network Load Balancers](#).
11. Configure the client in Account B to use the new bootstrap brokers for multi-VPC private connectivity to connect to the MSK cluster in Account A. See [Produce and consume data](#).

After authorization is complete, Amazon MSK creates a managed VPC connection for each specified VPC and auth scheme. The chosen security group is associated with each connection. This managed VPC connection is configured by Amazon MSK to connect privately to the brokers. You can use the new set of bootstrap brokers to connect privately to the Amazon MSK cluster.

Update the authorization schemes on a cluster

Multi-VPC private connectivity supports several authorization schemes: SASL/SCRAM, IAM, and TLS. The cluster owner can turn on/off private connectivity for one or more auth schemes. The cluster has to be in ACTIVE state to perform this action.

To turn on an auth scheme using the Amazon MSK console

1. Open the Amazon MSK console at [AWS Management Console](#) for the cluster that you want to edit.
2. In the navigation pane, under **MSK Clusters**, choose **Clusters** to display the list of clusters in the account.
3. Select the cluster that you want to edit. The cluster must be in an ACTIVE state.
4. Select the cluster **Properties** tab, and then go to **Network settings**.

5. Select the **Edit** dropdown menu and select **Turn on multi-VPC connectivity** to turn on a new auth scheme.
6. Select one or more authentication types that you want turned on for this cluster.
7. Select **Turn on selection**.

When you turn on a new auth scheme, you should also create new managed VPC connections for the new auth scheme and update your clients to use the bootstrap brokers specific to the new auth scheme.

To turn off an auth scheme using the Amazon MSK console

Note

When you turn off multi-VPC private connectivity for auth schemes, all connectivity related infrastructure, including the managed VPC connections, are deleted.

When you turn off multi-VPC private connectivity for auth schemes, existing VPC connections on client side change to INACTIVE, and Privatelink infrastructure on the cluster side, including the managed VPC connections, on the cluster side is removed. The cross-account user can only delete the inactive VPC connection. If private connectivity is turned on again on the cluster, the cross-account user needs to create a new connection to the cluster.

1. Open the Amazon MSK console at [AWS Management Console](#).
2. In the navigation pane, under **MSK Clusters**, choose **Clusters** to display the list of clusters in the account.
3. Select the cluster you want to edit. The cluster must be in an ACTIVE state.
4. Select the cluster **Properties** tab, then go to **Network settings**.
5. Select the **Edit** drop down menu and select **Turn off multi-VPC connectivity** (to turn off an auth scheme).
6. Select one or more authentication types you want turned off for this cluster.
7. Select **Turn off selection**.

Example To turn on/off an auth scheme with the API

As an alternative to the MSK console, you can use the [UpdateConnectivity API](#) to turn on multi-VPC private connectivity and configure auth schemes on an ACTIVE cluster. The following example shows SASL/SCRAM and IAM auth schemes turned on for the cluster.

When you turn on a new auth scheme, you should also create new managed VPC connections for the new auth scheme and update your clients to use the bootstrap brokers specific to the new auth scheme.

When you turn off multi-VPC private connectivity for auth schemes, existing VPC connections on client side change to INACTIVE, and Privatelink infrastructure on the cluster side, including the managed VPC connections, is removed. The cross-account user can only delete the inactive VPC connection. If private connectivity is turned on again on the cluster, the cross-account user needs to create a new connection to the cluster.

Request:

```
{
  "currentVersion": "string",
  "connectivityInfo": {
    "publicAccess": {
      "type": "string"
    },
    "vpcConnectivity": {
      "clientAuthentication": {
        "sasl": {
          "scram": {
            "enabled": TRUE
          },
          "iam": {
            "enabled": TRUE
          }
        },
        "tls": {
          "enabled": FALSE
        }
      }
    }
  }
}
```

Response:

```
{
  "clusterArn": "string",
  "clusterOperationArn": "string"
}
```

Reject a managed VPC connection to an Amazon MSK cluster

From the Amazon MSK console on the cluster admin account, you can reject a client VPC connection. The client VPC connection must be in the AVAILABLE state to be rejected. You might want to reject a managed VPC connection from a client that is no longer authorized to connect to your cluster. To prevent new managed VPC connections from a connecting to a client, deny access to the client in the cluster policy. A rejected connection still incurs cost until its deleted by the connection owner. See [Delete a managed VPC connection to an Amazon MSK cluster](#).

To reject a client VPC connection using the MSK console

1. Open the Amazon MSK console at [AWS Management Console](#).
2. In the navigation pane, select **Clusters** and scroll to the **Network settings > Client VPC connections** list.
3. Select the connection that you want to reject and select **Reject client VPC connection**.
4. Confirm that you want to reject the selected client VPC connection.

To reject a managed VPC connection using the API, use the `RejectClientVpcConnection` API.

Delete a managed VPC connection to an Amazon MSK cluster

The cross-account user can delete a managed VPC connection for an MSK cluster from the client account console. Because the cluster owner user doesn't own the managed VPC connection, the connection can't be deleted from the cluster admin account. Once a VPC connection is deleted, it no longer incurs cost.

To delete a managed VPC connection using the MSK console

1. From the client account, open the Amazon MSK console at [AWS Management Console](#).
2. In the navigation pane, select **Managed VPC connections**.
3. From the connection list, select the connection that you want to delete.
4. Confirm that you want to delete the VPC connection.

To delete a managed VPC connection using the API, use the `DeleteVpcConnection` API.

Permissions for multi-VPC private connectivity

This section summarizes the permissions needed for clients and clusters using the multi-VPC private connectivity feature. Multi-VPC private connectivity requires the client admin to create permissions on each client that will have a managed VPC connection to the MSK cluster. It also requires the MSK cluster admin to enable PrivateLink connectivity on the MSK cluster and select authentication schemes to control access to the cluster.

Cluster auth type and topic access permissions

Turn on the multi-VPC private connectivity feature for auth schemes that are enabled for your MSK cluster. See [Requirements and limitations for multi-VPC private connectivity](#). If you are configuring your MSK cluster to use SASL/SCRAM auth scheme, the Apache Kafka ACLs property `allow.everyone.if.no.acl.found=false` is mandatory. After you set the [Apache Kafka ACLs](#) for your cluster, update the cluster's configuration to have the property `allow.everyone.if.no.acl.found` set to `false` for the cluster. For information about how to update the configuration of a cluster, see [Broker configuration operations](#).

Cross-account cluster policy permissions

If a Kafka client is in an AWS account that is different than the MSK cluster, attach a cluster-based policy to the MSK cluster that authorizes the client root user for cross-account connectivity. You can edit the multi-VPC cluster policy using the IAM policy editor in the MSK console (cluster **Security settings** > **Edit cluster policy**), or use the following APIs to manage the cluster policy:

PutClusterPolicy

Attaches the cluster policy to the cluster. You can use this API to create or update the specified MSK cluster policy. If you're updating the policy, the `currentVersion` field is required in the request payload.

GetClusterPolicy

Retrieves the JSON text of the cluster policy document attached to the cluster.

DeleteClusterPolicy

Deletes the cluster policy.

The following is an example of the JSON for a basic cluster policy, similar to the one shown in the MSK console IAM policy editor. The following policy grants permissions for cluster, topic, and group-level access.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "AWS": [
        "123456789012"
      ]
    },
    "Action": [
      "kafka-cluster:*",
      "kafka:CreateVpcConnection",
      "kafka:GetBootstrapBrokers",
      "kafka:DescribeCluster",
      "kafka:DescribeClusterV2"
    ],
    "Resource": [
      "arn:aws:kafka:us-east-1:123456789012:cluster/testing/de8982fa-8222-4e87-8b20-9bf3cdfa1521-2",
      "arn:aws:kafka:us-east-1:123456789012:topic/testing/*",
      "arn:aws:kafka:us-east-1:123456789012:group/testing/*"
    ]
  }]
}
```

Client permissions for multi-VPC private connectivity to an MSK cluster

To set up multi-VPC private connectivity between a Kafka client and an MSK cluster, the client requires an attached identity policy that grants permissions for `kafka:CreateVpcConnection`, `ec2:CreateTags` and `ec2:CreateVPCEndpoint` actions on the client. For reference, the following is an example of the JSON for a basic client identity policy.

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

```
        "kafka:CreateVpcConnection",
        "ec2:CreateTags",
        "ec2:CreateVPCEndpoint"
    ],
    "Resource": "*"
}
]
```

Port information

Use the following port numbers so that Amazon MSK can communicate with client machines:

- To communicate with brokers in plaintext, use port 9092.
- To communicate with brokers with TLS encryption, use port 9094 for access from within AWS and port 9194 for public access.
- To communicate with brokers with SASL/SCRAM, use port 9096 for access from within AWS and port 9196 for public access.
- To communicate with brokers in a cluster that is set up to use [the section called “IAM access control”](#), use port 9098 for access from within AWS and port 9198 for public access.
- To communicate with Apache ZooKeeper by using TLS encryption, use port 2182. Apache ZooKeeper nodes use port 2181 by default.

Get the bootstrap brokers for an Amazon MSK cluster

The *bootstrap brokers* refer to the list of brokers that an Apache Kafka client can use to connect to an Amazon MSK cluster. This list may not include all the brokers in the cluster. You can get bootstrap brokers using the AWS Management Console, AWS CLI, or Amazon MSK API.

Topics

- [Get the bootstrap brokers using the AWS Management Console](#)
- [Get the bootstrap brokers using the AWS CLI](#)
- [Get the bootstrap brokers using the API](#)

Get the bootstrap brokers using the AWS Management Console

This process describes how to get bootstrap brokers for a cluster using the AWS Management Console. The term *bootstrap brokers* refers to a list of brokers that an Apache Kafka client can use as a starting point to connect to the cluster. This list doesn't necessarily include all of the brokers in a cluster.

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. The table shows all the clusters for the current region under this account. Choose the name of a cluster to view its description.
3. On the **Cluster summary** page, choose **View client information**. This shows you the bootstrap brokers, as well as the Apache ZooKeeper connection string.

Get the bootstrap brokers using the AWS CLI

Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see [the section called "List clusters"](#).

```
aws kafka get-bootstrap-brokers --cluster-arn ClusterArn
```

For an MSK cluster that uses [the section called "IAM access control"](#), the output of this command looks like the following JSON example.

```
{
  "BootstrapBrokerStringSaslIam": "b-1.myTestCluster.123z8u.c2.kafka.us-west-1.amazonaws.com:9098,b-2.myTestCluster.123z8u.c2.kafka.us-west-1.amazonaws.com:9098"
}
```

The following example shows the bootstrap brokers for a cluster that has public access turned on. Use the `BootstrapBrokerStringPublicSaslIam` for public access, and the `BootstrapBrokerStringSaslIam` string for access from within AWS.

```
{
  "BootstrapBrokerStringPublicSaslIam": "b-2-public.myTestCluster.v4ni96.c2.kafka-beta.us-east-1.amazonaws.com:9198,b-1-public.myTestCluster.v4ni96.c2.kafka-
```

```
beta.us-east-1.amazonaws.com:9198,b-3-public.myTestCluster.v4ni96.c2.kafka-beta.us-east-1.amazonaws.com:9198",
  "BootstrapBrokerStringSaslIam": "b-2.myTestCluster.v4ni96.c2.kafka-beta.us-east-1.amazonaws.com:9098,b-1.myTestCluster.v4ni96.c2.kafka-beta.us-east-1.amazonaws.com:9098,b-3.myTestCluster.v4ni96.c2.kafka-beta.us-east-1.amazonaws.com:9098"
}
```

The bootstrap brokers string should contain three brokers from across the Availability Zones in which your MSK cluster is deployed (unless only two brokers are available).

Get the bootstrap brokers using the API

To get the bootstrap brokers using the API, see [GetBootstrapBrokers](#).

Monitor an Amazon MSK Provisioned cluster

There are several ways that Amazon MSK helps you monitor the status of your Amazon MSK Provisioned cluster.

- Amazon MSK gathers Apache Kafka metrics and sends them to Amazon CloudWatch where you can view them. For more information about Apache Kafka metrics, including the ones that Amazon MSK surfaces, see [Monitoring](#) in the Apache Kafka documentation.
- You can also monitor your MSK cluster with Prometheus, an open-source monitoring application. For information about Prometheus, see [Overview](#) in the Prometheus documentation. To learn how to monitor your MSK Provisioned cluster with Prometheus, see [the section called “Monitor with Prometheus”](#).
- (Standard brokers only) Amazon MSK helps you monitor your disk storage capacity by automatically sending you storage capacity alerts when a Provisioned cluster is about to reach its storage capacity limit. The alerts also provide recommendations on the best steps to take to address detected issues. This helps you to identify and quickly resolve disk capacity issues before they become critical. Amazon MSK automatically sends these alerts to the [Amazon MSK console](#), AWS Health Dashboard, Amazon EventBridge, and email contacts for your AWS account. For information about storage capacity alerts, see [Use Amazon MSK storage capacity alerts](#).

Topics

- [View Amazon MSK metrics using CloudWatch](#)
- [Amazon MSK metrics for monitoring Standard brokers with CloudWatch](#)

- [Amazon MSK metrics for monitoring Express brokers with CloudWatch](#)
- [Monitor an MSK Provisioned cluster with Prometheus](#)
- [Monitor consumer lags](#)
- [Use Amazon MSK storage capacity alerts](#)

View Amazon MSK metrics using CloudWatch

You can monitor metrics for Amazon MSK using the CloudWatch console, the command line, or the CloudWatch API. The following procedures show you how to access metrics using these different methods.

To access metrics using the CloudWatch console

Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

1. In the navigation pane, choose **Metrics**.
2. Choose the **All metrics** tab, and then choose **AWS/Kafka**.
3. To view topic-level metrics, choose **Topic, Broker ID, Cluster Name**; for broker-level metrics, choose **Broker ID, Cluster Name**; and for cluster-level metrics, choose **Cluster Name**.
4. (Optional) In the graph pane, select a statistic and a time period, and then create a CloudWatch alarm using these settings.

To access metrics using the AWS CLI

Use the [list-metrics](#) and [get-metric-statistics](#) commands.

To access metrics using the CloudWatch CLI

Use the [mon-list-metrics](#) and [mon-get-stats](#) commands.

To access metrics using the CloudWatch API

Use the [ListMetrics](#) and [GetMetricStatistics](#) operations.

Amazon MSK metrics for monitoring Standard brokers with CloudWatch

Amazon MSK integrates with Amazon CloudWatch so that you can collect, view, and analyze CloudWatch metrics for your MSK Standard brokers. The metrics that you configure for your MSK

Provisioned clusters are automatically collected and pushed to CloudWatch at 1 minute intervals. You can set the monitoring level for an MSK Provisioned cluster to one of the following: DEFAULT, PER_BROKER, PER_TOPIC_PER_BROKER, or PER_TOPIC_PER_PARTITION. The tables in the following sections show all the metrics that are available starting at each monitoring level.

Note

The names of some Amazon MSK metrics for CloudWatch monitoring have changed in version 3.6.0 and higher. Use the new names for monitoring these metrics. For metrics with changed names, the table below shows the name used in version 3.6.0 and higher, followed by the name in version 2.8.2.tiered.

DEFAULT-level metrics are free. Pricing for other metrics is described in the [Amazon CloudWatch pricing](#) page.

DEFAULT Level monitoring

The metrics described in the following table are available at the DEFAULT monitoring level. They are free.

Name	When visible	Dimensions	Description
ActiveControllerCount	After the cluster gets to the ACTIVE state.	Cluster Name	Only one controller per cluster should be active at any given time.
BurstBalance	After the cluster gets to the ACTIVE state.	Cluster Name , Broker ID	<p>The remaining balance of input-output burst credits for EBS volumes in the cluster. Use it to investigate latency or decreased throughput.</p> <p>BurstBalance is not reported for EBS volumes when the baseline performance of a volume is higher than the maximum burst performance. For more information, see I/O Credits and burst performance.</p>

Name	When visible	Dimensions	Description
BytesInPerSec	After you create a topic.	Cluster Name, Broker ID, Topic	The number of bytes per second received from clients. This metric is available per broker and also per topic.
BytesOutPerSec	After you create a topic.	Cluster Name, Broker ID, Topic	The number of bytes per second sent to clients. This metric is available per broker and also per topic.
ClientConnectionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID, Client Authentication	The number of active authenticated client connections.
ConnectionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of active authenticated, unauthenticated, and inter-broker connections.

Name	When visible	Dimensions	Description
CPUCreditBalance	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of earned CPU credits that a broker has accrued since it was launched. Credits are accrued in the credit balance after they are earned, and removed from the credit balance when they are spent. If you run out of the CPU credit balance, it can have a negative impact on your cluster's performance. You can take steps to reduce CPU load. For example, you can reduce the number of client requests or update the broker type to an M5 broker type.
CpuIdle	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU idle time.
CpuIoWait	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU idle time during a pending disk operation.
CpuSystem	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU in kernel space.
CpuUser	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU in user space.

Name	When visible	Dimensions	Description
GlobalPartitionCount	After the cluster gets to the ACTIVE state.	Cluster Name	The number of partitions across all topics in the cluster, excluding replicas. Because GlobalPartitionCount doesn't include replicas, the sum of the PartitionCount values can be higher than GlobalPartitionCount if the replication factor for a topic is greater than 1.
GlobalTopicCount	After the cluster gets to the ACTIVE state.	Cluster Name	Total number of topics across all brokers in the cluster.
EstimatedMaxTimeLag	After consumer group consumes from a topic.	Cluster Name, Consumer Group, Topic	Time estimate (in seconds) to drain MaxOffsetLag .
KafkaAppLogsDiskUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of disk space used for application logs.
KafkaDataLogsDiskUsed (Cluster Name, Broker ID dimension)	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of disk space used for data logs.
LeaderCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The total number of leaders of partitions per broker, not including replicas.

Name	When visible	Dimensions	Description
MaxOffsetLag	After consumer group consumes from a topic.	Cluster Name, Consumer Group, Topic	The maximum offset lag across all partitions in a topic.
MemoryBuffered	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of buffered memory for the broker.
MemoryCached	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of cached memory for the broker.
MemoryFree	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of memory that is free and available for the broker.
HeapMemoryAfterGC	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of total heap memory in use after garbage collection.
MemoryUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of memory that is in use for the broker.
MessagesInPerSec	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of incoming messages per second for the broker.

Name	When visible	Dimensions	Description
NetworkRx Dropped	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of dropped receive packages.
NetworkRx Errors	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of network receive errors for the broker.
NetworkRx Packets	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of packets received by the broker.
NetworkTx Dropped	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of dropped transmit packages.
NetworkTx Errors	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of network transmit errors for the broker.
NetworkTx Packets	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of packets transmitted by the broker.
OfflinePartitionsCount	After the cluster gets to the ACTIVE state.	Cluster Name	Total number of partitions that are offline in the cluster.

Name	When visible	Dimensions	Description
PartitionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The total number of topic partitions per broker, including replicas.
ProduceTo taTimeMsMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean produce time in milliseconds.
RequestBy tesMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean number of request bytes for the broker.
RequestTime	After request throttling is applied.	Cluster Name, Broker ID	The average time in milliseconds spent in broker network and I/O threads to process requests.
RootDiskUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of the root disk used by the broker.
SumOffsetLag	After consumer group consumes from a topic.	Cluster Name, Consumer Group, Topic	The aggregated offset lag for all the partitions in a topic.
SwapFree	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of swap memory that is available for the broker.

Name	When visible	Dimensions	Description
SwapUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of swap memory that is in use for the broker.
TrafficShaping	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	High-level metrics indicating the number of packets shaped (dropped or queued) due to exceeding network allocations. Finer detail is available with PER_BROKER metrics.
UnderMinIsrPartitionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of under minIsr partitions for the broker.
UnderReplicatedPartitions	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of under-replicated partitions for the broker.
UserPartitionExists	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	A Boolean metric that indicates the presence of a user-owned partition on a broker. A value of 1 indicates the presence of partitions on the broker.
ZooKeeperRequestLatencyMsMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	For ZooKeeper-based cluster. The mean latency in milliseconds for Apache ZooKeeper requests from broker.

Name	When visible	Dimensions	Description
ZooKeeper SessionState	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	For ZooKeeper-based cluster. Connection status of broker's ZooKeeper session which may be one of the following: NOT_CONNECTED: '0.0', ASSOCIATING: '0.1', CONNECTING: '0.5', CONNECTED_READONLY: '0.8', CONNECTED: '1.0', CLOSED: '5.0', AUTH_FAILED: '10.0'.

PER_BROKER Level monitoring

When you set the monitoring level to PER_BROKER, you get the metrics described in the following table in addition to all the DEFAULT level metrics. You pay for the metrics in the following table, whereas the DEFAULT level metrics continue to be free. The metrics in this table have the following dimensions: Cluster Name, Broker ID.

Name	When visible	Description
BwInAllowanceExceeded	After the cluster gets to the ACTIVE state.	The number of packets shaped because the inbound aggregate bandwidth exceeded the maximum for the broker.
BwOutAllowanceExceeded	After the cluster gets to the ACTIVE state.	The number of packets shaped because the outbound aggregate bandwidth exceeded the maximum for the broker.
ConntrackAllowanceExceeded	After the cluster gets to the ACTIVE state.	The number of packets shaped because the connection tracking exceeded the maximum for the broker. Connection tracking is related to security groups that track each connection established to ensure

Name	When visible	Description
		that return packets are delivered as expected.
ConnectionCloseRate	After the cluster gets to the ACTIVE state.	The number of connections closed per second per listener. This number is aggregated per listener and filtered for the client listeners.
ConnectionCreationRate	After the cluster gets to the ACTIVE state.	The number of new connections established per second per listener. This number is aggregated per listener and filtered for the client listeners.
CpuCreditUsage	After the cluster gets to the ACTIVE state.	The number of CPU credits spent by the broker. If you run out of the CPU credit balance, it can have a negative impact on your cluster's performance. You can take steps to reduce CPU load. For example, you can reduce the number of client requests or update the broker type to an M5 broker type.
FetchConsumerLocalTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the consumer request is processed at the leader.
FetchConsumerRequestQueueTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the consumer request waits in the request queue.
FetchConsumerResponseQueueTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the consumer request waits in the response queue.
FetchConsumerResponseSendTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds for the consumer to send a response.

Name	When visible	Description
FetchConsumerTotalTimeMsMean	After there's a producer/consumer.	The mean total time in milliseconds that consumers spend on fetching data from the broker.
FetchFollowerLocalTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the follower request is processed at the leader.
FetchFollowerRequestQueueTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the follower request waits in the request queue.
FetchFollowerResponseQueueTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the follower request waits in the response queue.
FetchFollowerResponseSendTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds for the follower to send a response.
FetchFollowerTotalTimeMsMean	After there's a producer/consumer.	The mean total time in milliseconds that followers spend on fetching data from the broker.
FetchMessageConversionsPerSec	After you create a topic.	The number of fetch message conversions per second for the broker.
FetchThrottleByteRate	After bandwidth throttling is applied.	The number of throttled bytes per second.
FetchThrottleQueueSize	After bandwidth throttling is applied.	The number of messages in the throttle queue.
FetchThrottleTime	After bandwidth throttling is applied.	The average fetch throttle time in milliseconds.
IAMNumberOfConnectionRequests	After the cluster gets to the ACTIVE state.	The number of IAM authentication requests per second.

Name	When visible	Description
IAMTooManyConnections	After the cluster gets to the ACTIVE state.	The number of connections attempted beyond 100. 0 means the number of connections is within the limit. If >0, the throttle limit is being exceeded and you need to reduce number of connections.
NetworkProcessorAvgIdlePercent	After the cluster gets to the ACTIVE state.	The average percentage of the time the network processors are idle.
PpsAllowanceExceeded	After the cluster gets to the ACTIVE state.	The number of packets shaped because the bidirectional PPS exceeded the maximum for the broker.
ProduceLocalTimeMsMean	After the cluster gets to the ACTIVE state.	The mean time in milliseconds that the request is processed at the leader.
ProduceMessageConversionsPerSec	After you create a topic.	The number of produce message conversions per second for the broker.
ProduceMessageConversionsTimeMsMean	After the cluster gets to the ACTIVE state.	The mean time in milliseconds spent on message format conversions.
ProduceRequestQueueTimeMsMean	After the cluster gets to the ACTIVE state.	The mean time in milliseconds that request messages spend in the queue.
ProduceResponseQueueTimeMsMean	After the cluster gets to the ACTIVE state.	The mean time in milliseconds that response messages spend in the queue.
ProduceResponseSendTimeMsMean	After the cluster gets to the ACTIVE state.	The mean time in milliseconds spent on sending response messages.
ProduceThrottleByteRate	After bandwidth throttling is applied.	The number of throttled bytes per second.

Name	When visible	Description
ProduceThrottleQueueSize	After bandwidth throttling is applied.	The number of messages in the throttle queue.
ProduceThrottleTime	After bandwidth throttling is applied.	The average produce throttle time in milliseconds.
ProduceTotalTimeMs Mean	After the cluster gets to the ACTIVE state.	The mean produce time in milliseconds.
RemoteFetchBytesPerSec (RemoteBytesInPerSec in v2.8.2.tiered)	After there's a producer/consumer.	The total number of bytes transferred from tiered storage in response to consumer fetches. This metric includes all topic-partitions that contribute to downstream data transfer traffic. Category: Traffic and error rates. This is a KIP-405 metric.
RemoteCopyBytesPerSec (RemoteBytesOutPerSec in v2.8.2.tiered)	After there's a producer/consumer.	The total number of bytes transferred to tiered storage, including data from log segments, indexes, and other auxiliary files. This metric includes all topic-partitions that contribute to upstream data transfer traffic. Category: Traffic and error rates. This is a KIP-405 metric.
RemoteLogManagerTasksAvgIdlePercent	After the cluster gets to the ACTIVE state.	The average percentage of time the remote log manager spent idle. The remote log manager transfers data from the broker to tiered storage. Category: Internal activity. This is a KIP-405 metric.

Name	When visible	Description
RemoteLogReaderAvgIdlePercent	After the cluster gets to the ACTIVE state.	The average percentage of time the remote log reader spent idle. The remote log reader transfers data from the remote storage to the broker in response to consumer fetches. Category: Internal activity. This is a KIP-405 metric.
RemoteLogReaderTaskQueueSize	After the cluster gets to the ACTIVE state.	The number of tasks responsible for reads from tiered storage that are waiting to be scheduled. Category: Internal activity. This is a KIP-405 metric.
RemoteFetchErrorsPerSec (RemoteReadErrorPerSec in v2.8.2.tiered)	After the cluster gets to the ACTIVE state.	The total rate of errors in response to read requests that the specified broker sent to tiered storage to retrieve data in response to consumer fetches. This metric includes all topic partitions that contribute to downstream data transfer traffic. Category: traffic and error rates. This is a KIP-405 metric.
RemoteFetchRequestPerSec (RemoteReadRequestsPerSec in v2.8.2.tiered)	After the cluster gets to the ACTIVE state.	The total number of read requests that the specifies broker sent to tiered storage to retrieve data in response to consumer fetches. This metric includes all topic partitions which contribute to downstream data transfer traffic. Category: traffic and error rates. This is a KIP-405 metric.

Name	When visible	Description
RemoteCopyErrorsPerSec (RemoteWriteErrorPerSec in v2.8.2.tiered)	After the cluster gets to the ACTIVE state.	The total rate of errors in response to write requests that the specified broker sent to tiered storage to transfer data upstream. This metric includes all topic partitions that contribute to upstream data transfer traffic. Category: traffic and error rates. This is a KIP-405 metric.
RemoteLogSizeBytes	After the cluster gets to the ACTIVE state.	The number of bytes stored on the remote tier. This metric is available for tiered storage clusters from Apache Kafka version 3.7.x on Amazon MSK.
ReplicationBytesInPerSec	After you create a topic.	The number of bytes per second received from other brokers.
ReplicationBytesOutPerSec	After you create a topic.	The number of bytes per second sent to other brokers.
RequestExemptFromThrottleTime	After request throttling is applied.	The average time in milliseconds spent in broker network and I/O threads to process requests that are exempt from throttling.
RequestHandlerAvgIdlePercent	After the cluster gets to the ACTIVE state.	The average percentage of the time the request handler threads are idle.
RequestThrottleQueueSize	After request throttling is applied.	The number of messages in the throttle queue.
RequestThrottleTime	After request throttling is applied.	The average request throttle time in milliseconds.

Name	When visible	Description
TcpConnections	After the cluster gets to the ACTIVE state.	Shows number of incoming and outgoing TCP segments with the SYN flag set.
RemoteCopyLagBytes (TotalTierBytesLag in v2.8.2.tiered)	After you create a topic.	The total number of bytes of the data that is eligible for tiering on the broker but has not been transferred to tiered storage yet. This metrics show the efficiency of upstream data transfer. As the lag increases, the amount of data that doesn't persist in tiered storage increases. Category: Archive lag. This is a not a KIP-405 metric.
TrafficBytes	After the cluster gets to the ACTIVE state.	Shows network traffic in overall bytes between clients (producers and consumers) and brokers. Traffic between brokers isn't reported.
VolumeQueueLength	After the cluster gets to the ACTIVE state.	The number of read and write operation requests waiting to be completed in a specified time period.
VolumeReadBytes	After the cluster gets to the ACTIVE state.	The number of bytes read in a specified time period.
VolumeReadOps	After the cluster gets to the ACTIVE state.	The number of read operations in a specified time period.
VolumeTotalReadTime	After the cluster gets to the ACTIVE state.	The total number of seconds spent by all read operations that completed in a specified time period.

Name	When visible	Description
VolumeTotalWriteTime	After the cluster gets to the ACTIVE state.	The total number of seconds spent by all write operations that completed in a specified time period.
VolumeWriteBytes	After the cluster gets to the ACTIVE state.	The number of bytes written in a specified time period.
VolumeWriteOps	After the cluster gets to the ACTIVE state.	The number of write operations in a specified time period.

PER_TOPIC_PER_BROKER Level monitoring

When you set the monitoring level to PER_TOPIC_PER_BROKER, you get the metrics described in the following table, in addition to all the metrics from the PER_BROKER and DEFAULT levels. Only the DEFAULT level metrics are free. The metrics in this table have the following dimensions: Cluster Name, Broker ID, Topic.

Important

For an Amazon MSK cluster that uses Apache Kafka 2.4.1 or a newer version, the metrics in the following table appear only after their values become nonzero for the first time. For example, to see BytesInPerSec, one or more producers must first send data to the cluster.

Name	When visible	Description
FetchMessageConversionsPerSec	After you create a topic.	The number of fetched messages converted per second.
MessagesInPerSec	After you create a topic.	The number of messages received per second.
ProduceMessageConversionsPerSec	After you create a topic.	The number of conversions per second for produced messages.

Name	When visible	Description
RemoteFetchBytesPerSec (RemoteBytesInPerSec in v2.8.2.tiered)	After you create a topic and the topic is producing/consuming.	The number of bytes transferred from tiered storage in response to consumer fetches for the specified topic and broker. This metric includes all partitions from the topic that contribute to downstream data transfer traffic on the specified broker. Category: traffic and error rates. This is a KIP-405 metric.
RemoteCopyBytesPerSec (RemoteBytesOutPerSec in v2.8.2.tiered)	After you create a topic and the topic is producing/consuming.	The number of bytes transferred to tiered storage, for the specified topic and broker. This metric includes all partitions from the topic that contribute to upstream data transfer traffic on the specified broker. Category: traffic and error rates. This is a KIP-405 metric.
RemoteFetchErrorsPerSec (RemoteReadErrorPerSec in v2.8.2.tiered)	After you create a topic and the topic is producing/consuming.	The rate of errors in response to read requests that the specified broker sends to tiered storage to retrieve data in response to consumer fetches on the specified topic. This metric includes all partitions from the topic that contribute to downstream data transfer traffic on the specified broker. Category: traffic and error rates. This is a KIP-405 metric.
RemoteFetchRequestPerSec (RemoteReadRequestsPerSec in v2.8.2.tiered)	After you create a topic and the topic is producing/consuming.	The number of read requests that the specified broker sends to tiered storage to retrieve data in response to consumer fetches on the specified topic. This metric includes all partitions from the topic that contribute to downstream data transfer traffic on the specified broker. Category: traffic and error rates. This is a KIP-405 metric.

Name	When visible	Description
RemoteCopyErrorsPerSec (RemoteWriteErrorPerSec in v2.8.2.tiered)	After you create a topic and the topic is producing/consuming.	The rate of errors in response to write requests that the specified broker sends to tiered storage to transfer data upstream. This metric includes all partitions from the topic that contribute to upstream data transfer traffic on the specified broker. Category: traffic and error rates. This is a KIP-405 metric.
RemoteLogSizeBytes	After you create a topic.	The number of bytes stored on the remote tier. This metric is available for tiered storage clusters from Apache Kafka version 3.7.x on Amazon MSK.

PER_TOPIC_PER_PARTITION Level monitoring

When you set the monitoring level to PER_TOPIC_PER_PARTITION, you get the metrics described in the following table, in addition to all the metrics from the PER_TOPIC_PER_BROKER, PER_BROKER, and DEFAULT levels. Only the DEFAULT level metrics are free. The metrics in this table have the following dimensions: Consumer Group, Topic, Partition.

Name	When visible	Description
EstimatedTimeLag	After consumer group consumes from a topic.	Time estimate (in seconds) to drain the partition offset lag.
OffsetLag	After consumer group consumes from a topic.	Partition-level consumer lag in number of offsets.

Understand MSK Provisioned cluster states

The following table shows the possible states of a MSK Provisioned cluster and describes what they mean. Unless otherwise specified, MSK Provisioned cluster states apply to both Standard and Express broker types. This table also describes what actions you can and cannot perform when an MSK Provisioned cluster is in one of these states. To find out the state of a cluster, you can visit the AWS Management Console. You can also use the [describe-cluster-v2](#) command or the [DescribeClusterV2](#) operation to describe the Provisioned cluster. The description of a cluster includes its state.

MSK Provisioned cluster state	Meaning and possible actions
ACTIVE	You can produce and consume data. You can also perform Amazon MSK API and AWS CLI operations on the cluster.
CREATING	Amazon MSK is setting up the Provisioned cluster. You must wait for the cluster to reach the ACTIVE state before you can use it to produce or consume data or to perform Amazon MSK API or AWS CLI operations on it.
DELETING	The Provisioned cluster is being deleted. You cannot use it to produce or consume data. You also cannot perform Amazon MSK API or AWS CLI operations on it.
FAILED	The Provisioned cluster creation or deletion process failed. You cannot use the cluster to produce or consume data. You can delete the cluster but cannot perform Amazon MSK API or AWS CLI update operations on it.
HEALING	Amazon MSK is running an internal operation , like replacing an unhealthy broker. For example, the broker might be unresponsive. You can still use the Provisioned cluster to produce and consume data. However, you

MSK Provisioned cluster state	Meaning and possible actions
	cannot perform Amazon MSK API or AWS CLI update operations on the cluster until it returns to the ACTIVE state.
MAINTENANCE	(Standard brokers only) Amazon MSK is performing routine maintenance operations on the cluster. Such maintenance operations include security patching. You can still use the cluster to produce and consume data. However, you cannot perform Amazon MSK API or AWS CLI update operations on the cluster until it returns to the ACTIVE state. The cluster State remains ACTIVE during maintenance on Express brokers. See Patching .
REBOOTING_BROKER	Amazon MSK is rebooting a broker. You can still use the Provisioned cluster to produce and consume data. However, you cannot perform Amazon MSK API or AWS CLI update operations on the cluster until it returns to the ACTIVE state.
UPDATING	A user-initiated Amazon MSK API or AWS CLI operation is updating the Provisioned cluster. You can still use the Provisioned cluster to produce and consume data. However, you cannot perform any additional Amazon MSK API or AWS CLI update operations on the cluster until it returns to the ACTIVE state.

Amazon MSK metrics for monitoring Express brokers with CloudWatch

Amazon MSK integrates with CloudWatch so that you can collect, view, and analyze CloudWatch metrics for your MSK Express brokers. The metrics that you configure for your MSK Provisioned clusters are automatically collected and pushed to CloudWatch at 1 minute intervals. You can

set the monitoring level for an MSK Provisioned cluster to one of the following: DEFAULT, PER_BROKER, PER_TOPIC_PER_BROKER, or PER_TOPIC_PER_PARTITION. The tables in the following sections show the metrics that are available starting at each monitoring level.

DEFAULT-level metrics are free. Pricing for other metrics is described in the [Amazon CloudWatch pricing](#) page.

DEFAULT Level monitoring for Express brokers

The metrics described in the following table are available at the DEFAULT monitoring level. They are free.

DEFAULT Level monitoring for Express brokers

Name	When visible	Dimensions	Description
ActiveControllerCount	After the cluster gets to the ACTIVE state.	Cluster Name	Only one controller per cluster should be active at any given time.
BytesInPerSec	After you create a topic.	Cluster Name, Broker ID, Topic	The number of bytes per second received from clients. This metric is available per broker and also per topic.
BytesOutPerSec	After you create a topic.	Cluster Name, Broker ID, Topic	The number of bytes per second sent to clients. This metric is available per broker and also per topic.
ClientConnectionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID, Client Authentication	The number of active authenticated client connections.
ConnectionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of active authenticated,

Name	When visible	Dimensions	Description
			unauthenticated, and inter-broker connections.
CpuIdle	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU idle time.
CpuSystem	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU in kernel space.
CpuUser	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU in user space.
GlobalPartitionCount	After the cluster gets to the ACTIVE state.	Cluster Name	The number of partitions across all topics in the cluster, excluding replicas. Because GlobalPartitionCount doesn't include replicas, the sum of the PartitionCount values can be higher than GlobalPartitionCount if the replication factor for a topic is greater than 1.
GlobalTopicCount	After the cluster gets to the ACTIVE state.	Cluster Name	Total number of topics across all brokers in the cluster.

Name	When visible	Dimensions	Description
EstimatedMaxTimeLag	After consumer group consumes from a topic.	Consumer Group, Topic	Time estimate (in seconds) to drain MaxOffsetLag .
LeaderCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The total number of leaders of partitions per broker, not including replicas.
MaxOffsetLag	After consumer group consumes from a topic.	Consumer Group, Topic	The maximum offset lag across all partitions in a topic.
MemoryBuffered	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of buffered memory for the broker.
MemoryCached	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of cached memory for the broker.
MemoryFree	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of memory that is free and available for the broker.
MemoryUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of memory that is in use for the broker.
MessagesInPerSec	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of incoming messages per second for the broker.

Name	When visible	Dimensions	Description
NetworkRxDropped	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of dropped receive packages.
NetworkRxErrors	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of network receive errors for the broker.
NetworkRxPackets	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of packets received by the broker.
NetworkTxDropped	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of dropped transmit packages.
NetworkTxErrors	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of network transmit errors for the broker.
NetworkTxPackets	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of packets transmitted by the broker.
PartitionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The total number of topic partitions per broker, including replicas.
ProduceTotalTimeMs Mean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean produce time in milliseconds.
RequestBytesMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean number of request bytes for the broker.

Name	When visible	Dimensions	Description
RequestTime	After request throttling is applied.	Cluster Name, Broker ID	The average time in milliseconds spent in broker network and I/O threads to process requests.
SumOffsetLag	After consumer group consumes from a topic.	Consumer Group, Topic	The aggregated offset lag for all the partitions in a topic.
UserPartitionExists	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	Boolean metric that indicates the presence of a user-owned partition on a broker. A value of 1 indicates the presence of partitions on the broker.

PER_BROKER Level monitoring for Express brokers

When you set the monitoring level to PER_BROKER, you get the metrics described in the following table in addition to all the DEFAULT level metrics. You pay for the metrics in the following table, whereas the DEFAULT level metrics continue to be free of charge. The metrics in this table have the following dimensions: Cluster Name, Broker ID.

Additional metrics available starting at the PER_BROKER monitoring level

Name	When visible	Description
ConnectionCloseRate	After the cluster gets to the ACTIVE state.	The number of connections closed per second per listener. This number is aggregated per listener and filtered for the client listeners.

Name	When visible	Description
ConnectionCreationRate	After the cluster gets to the ACTIVE state.	The number of new connections established per second per listener. This number is aggregated per listener and filtered for the client listeners.
FetchConsumerLocalTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the consumer request is processed at the leader.
FetchConsumerRequestQueueTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the consumer request waits in the request queue.
FetchConsumerResponseQueueTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the consumer request waits in the response queue.
FetchConsumerResponseSendTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds for the consumer to send a response.
FetchConsumerTotalTimeMsMean	After there's a producer/consumer.	The mean total time in milliseconds that consumers spend on fetching data from the broker.
FetchFollowerLocalTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the follower request is processed at the leader.

Name	When visible	Description
FetchFollowerRequestQueueTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the follower request waits in the request queue.
FetchFollowerResponseQueueTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds that the follower request waits in the response queue.
FetchFollowerResponseSendTimeMsMean	After there's a producer/consumer.	The mean time in milliseconds for the follower to send a response.
FetchFollowerTotalTimeMsMean	After there's a producer/consumer.	The mean total time in milliseconds that followers spend on fetching data from the broker.
FetchThrottleByteRate	After bandwidth throttling is applied.	The number of throttled bytes per second.
FetchThrottleQueueSize	After bandwidth throttling is applied.	The number of messages in the throttle queue.
FetchThrottleTime	After bandwidth throttling is applied.	The average fetch throttle time in milliseconds.
IAMNumberOfConnectionRequests	After the cluster gets to the ACTIVE state.	The number of IAM authentication requests per second.

Name	When visible	Description
IAMTooManyConnections	After the cluster gets to the ACTIVE state.	The number of connections attempted beyond 100. 0 means the number of connections is within the limit. If >0, the throttle limit is being exceeded and you need to reduce number of connections.
NetworkProcessorAvgIdlePercent	After the cluster gets to the ACTIVE state.	The average percentage of the time the network processors are idle.
ProduceLocalTimeMsMean	After the cluster gets to the ACTIVE state.	The mean time in milliseconds that the request is processed at the leader.
ProduceRequestQueueTimeMsMean	After the cluster gets to the ACTIVE state.	The mean time in milliseconds that request messages spend in the queue.
ProduceResponseQueueTimeMsMean	After the cluster gets to the ACTIVE state.	The mean time in milliseconds that response messages spend in the queue.
ProduceResponseSendTimeMsMean	After the cluster gets to the ACTIVE state.	The mean time in milliseconds spent on sending response messages.
ProduceThrottleByteRate	After bandwidth throttling is applied.	The number of throttled bytes per second.
ProduceThrottleQueueSize	After bandwidth throttling is applied.	The number of messages in the throttle queue.

Name	When visible	Description
ProduceThrottleTime	After bandwidth throttling is applied.	The average produce throttle time in milliseconds.
ProduceTotalTimeMsMean	After the cluster gets to the ACTIVE state.	The mean produce time in milliseconds.
ReplicationBytesInPerSec	After you create a topic.	The number of bytes per second received from other brokers.
ReplicationBytesOutPerSec	After you create a topic.	The number of bytes per second sent to other brokers.
RequestExemptFromThrottleTime	After request throttling is applied.	The average time in milliseconds spent in broker network and I/O threads to process requests that are exempt from throttling.
RequestHandlerAvgIdlePercent	After the cluster gets to the ACTIVE state.	The average percentage of the time the request handler threads are idle.
RequestThrottleQueueSize	After request throttling is applied.	The number of messages in the throttle queue.
RequestThrottleTime	After request throttling is applied.	The average request throttle time in milliseconds.
TcpConnections	After the cluster gets to the ACTIVE state.	Shows number of incoming and outgoing TCP segments with the SYN flag set.

Name	When visible	Description
TrafficBytes	After the cluster gets to the ACTIVE state.	Shows network traffic in overall bytes between clients (producers and consumers) and brokers. Traffic between brokers isn't reported.

PER_TOPIC_PER_PARTITION level monitoring for Express brokers

When you set the monitoring level to PER_TOPIC_PER_PARTITION, you get the metrics described in the following table, in addition to all the metrics from the PER_TOPIC_PER_BROKER, PER_BROKER, and DEFAULT levels. Only the DEFAULT level metrics are free of charge. The metrics in this table have the following dimensions: Consumer Group, Topic, Partition.

Additional metrics available starting at the PER_PARTITION monitoring level

Name	When visible	Description
EstimatedTimeLag	After consumer group consumes from a topic.	Time estimate (in seconds) to drain the partition offset lag.
OffsetLag	After consumer group consumes from a topic.	Partition-level consumer lag in number of offsets.

PER_TOPIC_PER_BROKER level monitoring for Express brokers

When you set the monitoring level to PER_TOPIC_PER_BROKER, you get the metrics described in the following table, in addition to all the metrics from the PER_BROKER and DEFAULT levels. Only the DEFAULT level metrics are free of charge. The metrics in this table have the following dimensions: Cluster Name, Broker ID, Topic.

Important

The metrics in the following table appear only after their values become nonzero for the first time. For example, to see BytesInPerSec, one or more producers must first send data to the cluster.

Additional metrics available starting at the PER_TOPIC_PER_BROKER monitoring level

Name	When visible	Description
MessagesInPerSec	After you create a topic.	The number of messages received per second.

Monitor an MSK Provisioned cluster with Prometheus

You can monitor your MSK Provisioned cluster with Prometheus, an open-source monitoring system for time-series metric data. You can publish this data to Amazon Managed Service for Prometheus using Prometheus's remote write feature. You can also use tools that are compatible with Prometheus-formatted metrics or tools that integrate with Amazon MSK Open Monitoring, like [Datadog](#), [Lenses](#), [New Relic](#), and [Sumo logic](#). Open monitoring is available for free but charges apply for the transfer of data across Availability Zones.

For information about Prometheus, see the [Prometheus documentation](#).

For information about using Prometheus, see [Enhance operational insights for Amazon MSK using Amazon Managed Service for Prometheus and Amazon Managed Grafana](#).

Note

KRaft metadata mode and MSK Express brokers can't have open monitoring and public access both enabled.

Enable open monitoring on new MSK Provisioned clusters

This procedure describes how to enable open monitoring on a new MSK cluster using the AWS Management Console, the AWS CLI, or the Amazon MSK API.

Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. In the **Monitoring** section, select the check box next to **Enable open monitoring with Prometheus**.

3. Provide the required information in all the sections of the page, and review all the available options.
4. Choose **Create cluster**.

Using the AWS CLI

- Invoke the [create-cluster](#) command and specify its open-monitoring option. Enable the JmxExporter, the NodeExporter, or both. If you specify open-monitoring, the two exporters can't be disabled at the same time.

Using the API

- Invoke the [CreateCluster](#) operation and specify OpenMonitoring. Enable the jmxExporter, the nodeExporter, or both. If you specify OpenMonitoring, the two exporters can't be disabled at the same time.

Enable open monitoring on existing MSK Provisioned cluster

To enable open monitoring, make sure that the MSK Provisioned cluster is in the ACTIVE state.

Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose the name of the cluster that you want to update. This takes you to a page that contains details for the cluster.
3. On the **Properties** tab, scroll down to find the **Monitoring** section.
4. Choose **Edit**.
5. Select the check box next to **Enable open monitoring with Prometheus**.
6. Choose **Save changes**.

Using the AWS CLI

- Invoke the [update-monitoring](#) command and specify its open-monitoring option. Enable the JmxExporter, the NodeExporter, or both. If you specify open-monitoring, the two exporters can't be disabled at the same time.

Using the API

- Invoke the [UpdateMonitoring](#) operation and specify `OpenMonitoring`. Enable the `jmxExporter`, the `nodeExporter`, or both. If you specify `OpenMonitoring`, the two exporters can't be disabled at the same time.

Set up a Prometheus host on an Amazon EC2 instance

This procedure describes how to set up a Prometheus host using a `prometheus.yml` file.

1. Download the Prometheus server from <https://prometheus.io/download/#prometheus> to your Amazon EC2 instance.
2. Extract the downloaded file to a directory and go to that directory.
3. Create a file with the following contents and name it `prometheus.yml`.

```
# file: prometheus.yml
# my global config
global:
  scrape_interval:     60s

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped
  # from this config.
  - job_name: 'prometheus'
    static_configs:
      # 9090 is the prometheus server port
      - targets: ['localhost:9090']
    - job_name: 'broker'
      file_sd_configs:
        - files:
            - 'targets.json'
```

4. Use the [ListNodes](#) operation to get a list of your cluster's brokers.
5. Create a file named `targets.json` with the following JSON. Replace *broker_dns_1*, *broker_dns_2*, and the rest of the broker DNS names with the DNS names you obtained for your brokers in the previous step. Include all of the brokers you obtained in the previous step. Amazon MSK uses port 11001 for the JMX Exporter and port 11002 for the Node Exporter.

ZooKeeper mode targets.json

```
[
  {
    "labels": {
      "job": "jmx"
    },
    "targets": [
      "broker_dns_1:11001",
      "broker_dns_2:11001",
      .
      .
      .
      "broker_dns_N:11001"
    ]
  },
  {
    "labels": {
      "job": "node"
    },
    "targets": [
      "broker_dns_1:11002",
      "broker_dns_2:11002",
      .
      .
      .
      "broker_dns_N:11002"
    ]
  }
]
```

KRaft mode targets.json

```
[
  {
    "labels": {
      "job": "jmx"
    },
    "targets": [
      "broker_dns_1:11001",
      "broker_dns_2:11001",
      .
    ]
  }
]
```

```
    .
    .
    "broker_dns_N:11001",
    "controller_dns_1:11001",
    "controller_dns_2:11001",
    "controller_dns_3:11001"
  ]
},
{
  "labels": {
    "job": "node"
  },
  "targets": [
    "broker_dns_1:11002",
    "broker_dns_2:11002",
    .
    .
    .
    "broker_dns_N:11002"
  ]
}
]
```

Note

To scrape JMX metrics from KRaft controllers, add controller DNS names as targets in the JSON file. For example: `controller_dns_1:11001`, replacing `controller_dns_1` with the actual controller DNS name.

6. To start the Prometheus server on your Amazon EC2 instance, run the following command in the directory where you extracted the Prometheus files and saved `prometheus.yml` and `targets.json`.

```
./prometheus
```

7. Find the IPv4 public IP address of the Amazon EC2 instance where you ran Prometheus in the previous step. You need this public IP address in the following step.

8. To access the Prometheus web UI, open a browser that can access your Amazon EC2 instance, and go to *Prometheus-Instance-Public-IP*:9090, where *Prometheus-Instance-Public-IP* is the public IP address you got in the previous step.

Use Prometheus metrics

All metrics emitted by Apache Kafka to JMX are accessible using open monitoring with Prometheus. For information about Apache Kafka metrics, see [Monitoring](#) in the Apache Kafka documentation. Along with Apache Kafka metrics, consumer-lag metrics are also available at port 11001 under the JMX MBean name `kafka.consumer.group:type=ConsumerLagMetrics`. You can also use the Prometheus Node Exporter to get CPU and disk metrics for your brokers at port 11002.

Store Prometheus metrics in Amazon Managed Service for Prometheus

Amazon Managed Service for Prometheus is a Prometheus-compatible monitoring and alerting service that you can use to monitor Amazon MSK clusters. It is a fully-managed service that automatically scales the ingestion, storage, querying, and alerting of your metrics. It also integrates with AWS security services to give you fast and secure access to your data. You can use the open-source PromQL query language to query your metrics and alert on them.

For more information, see [Getting started with Amazon Managed Service for Prometheus](#).

Monitor consumer lags

Monitoring consumer lag allows you to identify slow or stuck consumers that aren't keeping up with the latest data available in a topic. When necessary, you can then take remedial actions, such as scaling or rebooting those consumers. To monitor consumer lag, you can use Amazon CloudWatch or open monitoring with Prometheus.

Consumer lag metrics quantify the difference between the latest data written to your topics and the data read by your applications. Amazon MSK provides the following consumer-lag metrics, which you can get through Amazon CloudWatch or through open monitoring with Prometheus: `EstimatedMaxTimeLag`, `EstimatedTimeLag`, `MaxOffsetLag`, `OffsetLag`, and `SumOffsetLag`. For information about these metrics, see [the section called “Metrics for monitoring Standard brokers with CloudWatch”](#).

Note

- Consumer lag metrics are emitted only if a consumer group is in a STABLE or EMPTY state. A consumer group is STABLE after the successful completion of re-balancing, ensuring that partitions are evenly distributed among the consumers.
- Consumer lag metrics are absent in the following scenarios:
 - If the consumer group is unstable.
 - The name of the consumer group contains a colon (:).
 - You haven't set the consumer offset for the consumer group.

Amazon MSK supports consumer lag metrics for clusters with Apache Kafka 2.2.1 or a later version.

Use Amazon MSK storage capacity alerts

On Amazon MSK provisioned clusters, you choose the cluster's primary storage capacity. If you exhaust the storage capacity on a broker in your provisioned cluster, it can affect its ability to produce and consume data, leading to costly downtime. Amazon MSK offers CloudWatch metrics to help you monitor your cluster's storage capacity. However, to make it easier for you to detect and resolve storage capacity issues, Amazon MSK automatically sends you dynamic cluster storage capacity alerts. The storage capacity alerts include recommendations for short-term and long-term steps to manage your cluster's storage capacity. From the [Amazon MSK console](#), you can use quick links within the alerts to take recommended actions immediately.

There are two types of MSK storage capacity alerts: proactive and remedial.

- Proactive ("Action required") storage capacity alerts warn you about potential storage issues with your cluster. When a broker in an MSK cluster has used over 60% or 80% of its disk storage capacity, you'll receive proactive alerts for the affected broker.
- Remedial ("Critical action required") storage capacity alerts require you to take remedial action to fix a critical cluster issue when one of the brokers in your MSK cluster has run out of disk storage capacity.

Amazon MSK automatically sends these alerts to the [Amazon MSK console](#), [AWS Health Dashboard](#), [Amazon EventBridge](#), and email contacts for your AWS account. You can also [configure Amazon EventBridge](#) to deliver these alerts to Slack or to tools such as New Relic, and Datadog.

Storage capacity alerts are enabled by default for all MSK provisioned clusters and can't be turned off. This feature is supported in all regions where MSK is available.

Monitor storage capacity alerts

You can check for storage capacity alerts in several ways:

- Go to the [Amazon MSK console](#). Storage capacity alerts are displayed in the cluster alerts pane for 90 days. The alerts contain recommendations and single-click link actions to address disk storage capacity issues.
- Use [ListClusters](#), [ListClustersV2](#), [DescribeCluster](#), or [DescribeClusterV2](#) APIs to view `CustomerActionStatus` and all the alerts for a cluster.
- Go to the [AWS Health Dashboard](#) to view alerts from MSK and other AWS services.
- Set up [AWS Health API](#) and [Amazon EventBridge](#) to route alert notifications to 3rd party platforms such as Datadog, NewRelic, and Slack.

Update security settings of a Amazon MSK cluster

Use the [UpdateSecurity](#) Amazon MSK operation to update the authentication and client-broker encryption settings of your MSK cluster. You can also update the Private Security Authority used to sign certificates for mutual TLS authentication. You can't change the in-cluster (broker-to-broker) encryption setting.

The cluster must be in the ACTIVE state for you to update its security settings.

If you turn on authentication using IAM, SASL, or TLS, you must also turn on encryption between clients and brokers. The following table shows the possible combinations.

Authentication	Client-broker encryption options	Broker-broker encryption
Unauthenticated	TLS, PLAINTEXT, TLS_PLAINTEXT	Can be on or off.
mTLS	TLS, TLS_PLAINTEXT	Must be on.
SASL/SCRAM	TLS	Must be on.
SASL/IAM	TLS	Must be on.

When client-broker encryption is set to `TLS_PLAINTEXT` and client-authentication is set to `mTLS`, Amazon MSK creates two types of listeners for clients to connect to: one listener for clients to connect using `mTLS` authentication with TLS Encryption, and another for clients to connect without authentication or encryption (plaintext).

For more information about security settings, see [the section called “Security”](#).

Update Amazon MSK cluster security settings using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose the MSK cluster that you want to update.
3. In the **Security settings** section, choose **Edit**.
4. Choose the authentication and encryption settings that you want for the cluster, then choose **Save changes**.

Updating Amazon MSK cluster security settings using the AWS CLI

1. Create a JSON file that contains the encryption settings that you want the cluster to have. The following is an example.

Note

You can only update the client-broker encryption setting. You can't update the in-cluster (broker-to-broker) encryption setting.

```
{"EncryptionInTransit":{"ClientBroker": "TLS"}}
```

2. Create a JSON file that contains the authentication settings that you want the cluster to have. The following is an example.

```
{"Sasl":{"Scram":{"Enabled":true}}}
```

3. Run the following AWS CLI command:

```
aws kafka update-security --cluster-arn ClusterArn --current-version Current-Cluster-Version --client-authentication file://Path-to-Authentication-Settings-JSON-File --encryption-info file://Path-to-Encryption-Settings-JSON-File
```

The output of this update-security operation looks like the following JSON.

```
{
  "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/
  abcdefab-1234-abcd-5678-cdef0123ab01-2",
  "ClusterOperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
  operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
  abcd-4f7f-1234-9876543210ef"
}
```

4. To see the status of the update-security operation, run the following command, replacing *ClusterOperationArn* with the ARN that you obtained in the output of the update-security command.

```
aws kafka describe-cluster-operation --cluster-operation-arn ClusterOperationArn
```

The output of this describe-cluster-operation command looks like the following JSON example.

```
{
  "ClusterOperationInfo": {
    "ClientRequestId": "c0b7af47-8591-45b5-9c0c-909a1a2c99ea",
    "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/
    exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2",
    "CreationTime": "2021-09-17T02:35:47.753000+00:00",
    "OperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
    operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
    abcd-4f7f-1234-9876543210ef",
    "OperationState": "PENDING",
    "OperationType": "UPDATE_SECURITY",
    "SourceClusterInfo": {},
    "TargetClusterInfo": {}
  }
}
```

If `OperationState` has the value `PENDING` or `UPDATE_IN_PROGRESS`, wait a while, then run the `describe-cluster-operation` command again.

 **Note**

The AWS CLI and API operations for updating the security settings of a cluster are idempotent. This means that if you invoke the security update operation and specify an authentication or encryption setting that is the same setting that the cluster currently has, that setting won't change.

Updating a cluster's security settings using the API

To update the security settings for a Amazon MSK cluster using the API, see [UpdateSecurity](#).

 **Note**

The AWS CLI and API operations for updating the security settings of a MSK cluster are idempotent. This means that if you invoke the security update operation and specify an authentication or encryption setting that is the same setting that the cluster currently has, that setting won't change.

Expand the number of brokers in an Amazon MSK cluster

Use this Amazon MSK operation when you want to increase the number of brokers in your MSK cluster. To expand a cluster, make sure that it is in the `ACTIVE` state.

 **Important**

If you want to expand an MSK cluster, make sure to use this Amazon MSK operation. Don't try to add brokers to a cluster without using this operation.

For information about how to rebalance partitions after you add brokers to a cluster, see [the section called "Reassign partitions"](#).

Expand a Amazon MSK cluster using the AWS Management Console

This process describes how to increase the number of brokers in an Amazon MSK cluster using the AWS Management Console.

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose the MSK cluster whose number of brokers you want to increase.
3. From the **Actions** dropdown, choose **Edit number of brokers**.
4. Enter the number of brokers that you want the cluster to have per Availability Zone and then choose **Save changes**.

Expand a Amazon MSK cluster using the AWS CLI

This process describes how to increase the number of brokers in an Amazon MSK cluster using the AWS CLI.

1. Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see [the section called "List clusters"](#).

Replace *Current-Cluster-Version* with the current version of the cluster.

Important

Cluster versions aren't simple integers. To find the current version of the cluster, use the [DescribeCluster](#) operation or the [describe-cluster](#) AWS CLI command. An example version is KTVPDKIKX0DER.

The *Target-Number-of-Brokers* parameter represents the total number of broker nodes that you want the cluster to have when this operation completes successfully. The value you specify for *Target-Number-of-Brokers* must be a whole number that is greater than the current number of brokers in the cluster. It must also be a multiple of the number of Availability Zones.

```
aws kafka update-broker-count --cluster-arn ClusterArn --current-version Current-Cluster-Version --target-number-of-broker-nodes Target-Number-of-Brokers
```

The output of this update-broker-count operation looks like the following JSON.

```
{
  "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/
  abcdefab-1234-abcd-5678-cdef0123ab01-2",
  "ClusterOperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
  operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
  abcd-4f7f-1234-9876543210ef"
}
```

2. To get the result of the update-broker-count operation, run the following command, replacing *ClusterOperationArn* with the ARN that you obtained in the output of the update-broker-count command.

```
aws kafka describe-cluster-operation --cluster-operation-arn ClusterOperationArn
```

The output of this describe-cluster-operation command looks like the following JSON example.

```
{
  "ClusterOperationInfo": {
    "ClientRequestId": "c0b7af47-8591-45b5-9c0c-909a1a2c99ea",
    "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/
    exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2",
    "CreationTime": "2019-09-25T23:48:04.794Z",
    "OperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
    operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
    abcd-4f7f-1234-9876543210ef",
    "OperationState": "UPDATE_COMPLETE",
    "OperationType": "INCREASE_BROKER_COUNT",
    "SourceClusterInfo": {
      "NumberOfBrokerNodes": 9
    },
    "TargetClusterInfo": {
      "NumberOfBrokerNodes": 12
    }
  }
}
```

```
}  
}
```

In this output, `OperationType` is `INCREASE_BROKER_COUNT`. If `OperationState` has the value `UPDATE_IN_PROGRESS`, wait a while, then run the `describe-cluster-operation` command again.

Expand a Amazon MSK cluster using the API

To increase the number of brokers in a cluster using the API, see [UpdateBrokerCount](#).

Remove a broker from an Amazon MSK cluster

Use this Amazon MSK operation when you want to remove brokers from Amazon Managed Streaming for Apache Kafka (MSK) provisioned clusters. You can reduce your cluster's storage and compute capacity by removing sets of brokers, with no availability impact, data durability risk, or disruption to your data streaming applications.

You can add more brokers to your cluster to handle increase in traffic, and remove brokers when the traffic subsides. With broker addition and removal capability, you can best utilize your cluster capacity and optimize your MSK infrastructure costs. Broker removal gives you broker-level control over existing cluster capacity to fit your workload needs and avoid migration to another cluster.

Use the AWS Console, Command Line Interface (CLI), SDK, or AWS CloudFormation to reduce broker count of your provisioned cluster. MSK picks the brokers that do not have any partitions on them (except for canary topics) and prevents applications from producing data to those brokers, while safely removing those brokers from the cluster.

You should remove one broker per Availability Zone, if you want to reduce a cluster's storage and compute. For example, you can remove two brokers from a two Availability Zone cluster, or three brokers from a three Availability Zone cluster in a single broker removal operation.

For information about how to rebalance partitions after you remove brokers from a cluster, see [the section called "Reassign partitions"](#).

You can remove brokers from all M5 and M7g based MSK provisioned clusters, regardless of the instance size.

Broker removal is supported on Kafka versions 2.8.1 and above, including on KRaft mode clusters.

Topics

- [Prepare to remove brokers by removing all partitions](#)
- [Remove a broker with the AWS Management Console](#)
- [Remove a broker with the AWS CLI](#)
- [Remove a broker with the AWS API](#)

Prepare to remove brokers by removing all partitions

Before you start the broker removal process, first move all partitions, except ones for topics `__amazon_msk_canary` and `__amazon_msk_canary_state` from the brokers you plan to remove. These are internal topics that Amazon MSK creates for cluster health and diagnostic metrics.

You can use Kafka admin APIs or Cruise Control to move partitions to other brokers that you intend to retain in the cluster. See [Reassign partitions](#).

Example process to remove partitions

This section is an example of how to remove partitions from the broker you intend to remove. Assume you have a cluster with 6 brokers, 2 brokers in each AZ, and it has four topics:

- `__amazon_msk_canary`
- `__consumer_offsets`
- `__amazon_msk_connect_offsets_my-mskc-connector_12345678-09e7-c657f7e4ff32-2`
- `msk-brk-rmv`

1. Create a client machine as described in [Create a client machine](#).
2. After configuring the client machine, run the following command to list all the available topics in your cluster.

```
./bin/kafka-topics.sh --bootstrap-server "CLUSTER_BOOTSTRAP_STRING" --list
```

In this example, we see four topic names, `__amazon_msk_canary`, `__consumer_offsets`, `__amazon_msk_connect_offsets_my-mskc-connector_12345678-09e7-c657f7e4ff32-2`, and `msk-brk-rmv`.

3. Create a json file called `topics.json` on the client machine and add all the user topic names as in the following code example. You don't need to include the `__amazon_msk_canary` topic name as this is a service managed topic that will be automatically moved when necessary.

```
{
  "topics": [
    {"topic": "msk-brk-rmv"},
    {"topic": "__consumer_offsets"},
    {"topic": "__amazon_msk_connect_offsets_my-mskc-connector_12345678-09e7-c657f7e4ff32-2"}
  ],
  "version": 1
}
```

4. Run the following command to generate a proposal to move partitions to only 3 brokers out of 6 brokers on the cluster.

```
./bin/kafka-reassign-partitions.sh --bootstrap-server "CLUSTER_BOOTSTRAP_STRING" --
topics-to-move-json-file topics.json --broker-list 1,2,3 --generate
```

5. Create a file called `reassignment-file.json` and copy the proposed partition reassignment configuration you got from above command.
6. Run the following command to move partitions that you specified in the `reassignment-file.json`.

```
./bin/kafka-reassign-partitions.sh --bootstrap-server "CLUSTER_BOOTSTRAP_STRING" --
reassignment-json-file reassignment-file.json --execute
```

The output looks similar to the following:

```
Successfully started partition reassignments for morpheus-test-topic-1-0,test-
topic-1-0
```

7. Run the following command to verify all partitions have moved.

```
./bin/kafka-reassign-partitions.sh --bootstrap-server "CLUSTER_BOOTSTRAP_STRING" --
reassignment-json-file reassignment-file.json --verify
```

The output looks similar to the following. Monitor the status until all partitions in your requested topics have been reassigned successfully:

```
Status of partition reassignment:  
Reassignment of partition msk-brk-rmv-0 is completed.  
Reassignment of partition msk-brk-rmv-1 is completed.  
Reassignment of partition __consumer_offsets-0 is completed.  
Reassignment of partition __consumer_offsets-1 is completed.
```

8. When the status indicates that the partition reassignment for each partition is completed, monitor the `UserPartitionExists` metrics for 5 minutes to ensure it displays 0 for the brokers from which you moved the partitions. After confirming this, you can proceed to remove the broker from the cluster.

Remove a broker with the AWS Management Console

To remove brokers with the AWS Management Console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. Choose the MSK cluster that contains brokers you want to remove.
3. On the cluster details page, choose the **Actions** button and select the **Edit number of brokers** option.
4. Enter the number of brokers that you want the cluster to have per Availability Zone. The console summarizes the number of brokers across availability zones that will be removed. Make sure this what you want.
5. Choose **Save changes**.

To prevent accidental broker removal, the console asks you to confirm that you want to delete brokers.

Remove a broker with the AWS CLI

Run the following command, replacing `ClusterArn` with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, [Listing Amazon MSK clusters](#). Replace `Current-Cluster-Version` with the current version of the cluster.

⚠ Important

Cluster versions aren't simple integers. To find the current version of the cluster, use the [DescribeCluster](#) operation or the [describe-cluster](#) AWS CLI command. An example version is KTVPDKIKX0DER.

The *Target-Number-of-Brokers* parameter represents the total number of broker nodes that you want the cluster to have when this operation completes successfully. The value you specify for *Target-Number-of-Brokers* must be a whole number that is less than the current number of brokers in the cluster. It must also be a multiple of the number of Availability Zones.

```
aws kafka update-broker-count --cluster-arn ClusterArn --current-version Current-Cluster-Version --target-number-of-broker-nodes Target-Number-of-Brokers
```

The output of this update-broker-count operation looks like the following JSON.

```
{
  "ClusterOperationInfo": {
    "ClientRequestId": "c0b7af47-8591-45b5-9c0c-909a1a2c99ea",
    "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/
    abcdefab-1234-abcd-5678-cdef0123ab01-2",
    "CreationTime": "2019-09-25T23:48:04.794Z",
    "OperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
    operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
    abcd-4f7f-1234-9876543210ef",
    "OperationState": "UPDATE_COMPLETE",
    "OperationType": "DECREASE_BROKER_COUNT",
    "SourceClusterInfo": {
      "NumberOfBrokerNodes": 12
    },
    "TargetClusterInfo": {
      "NumberOfBrokerNodes": 9
    }
  }
}
```

In this output, *OperationType* is *DECREASE_BROKER_COUNT*. If *OperationState* has the value *UPDATE_IN_PROGRESS*, wait a while, then run the *describe-cluster-operation* command again.

Remove a broker with the AWS API

To remove brokers in a cluster using the API, see [UpdateBrokerCount](#) in the *Amazon Managed Streaming for Apache Kafka API Reference*.

Update the Amazon MSK cluster broker size

You can scale your MSK cluster on demand by changing the size of your brokers without reassigning Apache Kafka partitions. Changing the size of your brokers gives you the flexibility to adjust your MSK cluster's compute capacity based on changes in your workloads, without interrupting your cluster I/O. Amazon MSK uses the same broker size for all the brokers in a given cluster.

This section describes how to update the broker size for your MSK cluster. For Standard brokers, you can update your cluster broker size from M5 or T3 to M7g, or from M7g to M5. For Express brokers, you can use only M7g broker sizes.

Note

You can't migrate from a larger broker size to a smaller broker size. For example, M7g.large to T3.small.

Be aware that migrating to a smaller broker size can decrease performance and reduce maximum achievable throughput per broker. Migrating to a larger broker size can increase performance but might cost more.

The broker-size update happens in a rolling fashion while the cluster is up and running. This means that Amazon MSK takes down one broker at a time to perform the broker-size update. For information about how to make a cluster highly available during a broker-size update, see [the section called "Build highly available clusters"](#). To further reduce any potential impact on productivity, you can perform the broker-size update during a period of low traffic.

During a broker-size update, you can continue to produce and consume data. However, you must wait until the update is done before you can reboot brokers or invoke any of the update operations listed under [Amazon MSK operations](#).

If you want to update your cluster to a smaller broker size, we recommend that you try the update on a test cluster first to see how it affects your scenario.

Important

You can't update a cluster to a smaller broker size if the number of partitions per broker exceeds the maximum number specified in [the section called "Right-size your cluster: Number of partitions per Standard broker"](#).

Update the Amazon MSK cluster broker size using the AWS Management Console

This process shows how to update the Amazon MSK cluster broker size using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose the MSK cluster for which you want to update the broker size.
3. On the details page for the cluster, find the **Brokers summary** section, and choose **Edit broker size**.
4. Choose the broker size you want from the list.
5. Save changes.

Update the Amazon MSK cluster broker size using the AWS CLI

Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see [the section called "List clusters"](#).

1. Replace *Current-Cluster-Version* with the current version of the cluster and *TargetType* with the new size that you want the brokers to be. To learn more about broker sizes, see [the section called "Broker types"](#).

```
aws kafka update-broker-type --cluster-arn ClusterArn --current-version Current-Cluster-Version --target-instance-type TargetType
```

The following is an example of how to use this command:

```
aws kafka update-broker-type --cluster-arn "arn:aws:kafka:us-east-1:0123456789012:cluster/exampleName/abcd1234-0123-abcd-5678-1234abcd-1" --current-version "K1X5R6FKA87" --target-instance-type kafka.m5.large
```

The output of this command looks like the following JSON example.

```
{
  "ClusterArn": "arn:aws:kafka:us-east-1:0123456789012:cluster/exampleName/abcd1234-0123-abcd-5678-1234abcd-1",
  "ClusterOperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-abcd-4f7f-1234-9876543210ef"
}
```

2. To get the result of the `update-broker-type` operation, run the following command, replacing *ClusterOperationArn* with the ARN that you obtained in the output of the `update-broker-type` command.

```
aws kafka describe-cluster-operation --cluster-operation-arn ClusterOperationArn
```

The output of this `describe-cluster-operation` command looks like the following JSON example.

```
{
  "ClusterOperationInfo": {
    "ClientRequestId": "982168a3-939f-11e9-8a62-538df00285db",
    "ClusterArn": "arn:aws:kafka:us-east-1:0123456789012:cluster/exampleName/abcd1234-0123-abcd-5678-1234abcd-1",
    "CreationTime": "2021-01-09T02:24:22.198000+00:00",
    "OperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-abcd-4f7f-1234-9876543210ef",
    "OperationState": "UPDATE_COMPLETE",
    "OperationType": "UPDATE_BROKER_TYPE",
    "SourceClusterInfo": {
      "InstanceType": "t3.small"
    },
    "TargetClusterInfo": {
      "InstanceType": "m5.large"
    }
  }
}
```

```
}  
}
```

If `OperationState` has the value `UPDATE_IN_PROGRESS`, wait a while, then run the `describe-cluster-operation` command again.

Updating the broker size using the API

To update the broker size using the API, see [UpdateBrokerType](#).

You can use `UpdateBrokerType` to update your cluster broker size from M5 or T3 to M7g, or from M7g to M5.

Use LinkedIn's Cruise Control for Apache Kafka with Amazon MSK

You can use LinkedIn's Cruise Control to rebalance your Amazon MSK cluster, detect and fix anomalies, and monitor the state and health of the cluster.

To download and build Cruise Control

1. Create an Amazon EC2 instance in the same Amazon VPC as the Amazon MSK cluster.
2. Install Prometheus on the Amazon EC2 instance that you created in the previous step. Note the private IP and the port. The default port number is 9090. For information on how to configure Prometheus to aggregate metrics for your cluster, see [the section called "Monitor with Prometheus"](#).
3. Download [Cruise Control](#) on the Amazon EC2 instance. (Alternatively, you can use a separate Amazon EC2 instance for Cruise Control if you prefer.) For a cluster that has Apache Kafka version 2.4.*, use the latest 2.4.* Cruise Control release. If your cluster has an Apache Kafka version that is older than 2.4.*, use the latest 2.0.* Cruise Control release.
4. Decompress the Cruise Control file, then go to the decompressed folder.
5. Run the following command to install git.

```
sudo yum -y install git
```

6. Run the following command to initialize the local repo. Replace *Your-Cruise-Control-Folder* with the name of your current folder (the folder that you obtained when you decompressed the Cruise Control download).

```
git init && git add . && git commit -m "Init local repo." && git tag -a Your-Cruise-Control-Folder -m "Init local version."
```

7. Run the following command to build the source code.

```
./gradlew jar copyDependantLibs
```

To configure and run Cruise Control

1. Make the following updates to the `config/cruisecontrol.properties` file. Replace the example bootstrap servers and bootstrap-brokers string with the values for your cluster. To get these strings for your cluster, you can see the cluster details in the console. Alternatively, you can use the [GetBootstrapBrokers](#) and [DescribeCluster](#) API operations or their CLI equivalents.

```
# If using TLS encryption, use 9094; use 9092 if using plaintext
bootstrap.servers=b-1.test-cluster.2skv42.c1.kafka.us-
east-1.amazonaws.com:9094,b-2.test-cluster.2skv42.c1.kafka.us-
east-1.amazonaws.com:9094,b-3.test-cluster.2skv42.c1.kafka.us-
east-1.amazonaws.com:9094

# SSL properties, needed if cluster is using TLS encryption
security.protocol=SSL
ssl.truststore.location=/home/ec2-user/kafka.client.truststore.jks

# Use the Prometheus Metric Sampler
metric.sampler.class=com.linkedin.kafka.cruisecontrol.monitor.sampling.prometheus.Prometheus

# Prometheus Metric Sampler specific configuration
prometheus.server.endpoint=1.2.3.4:9090 # Replace with your Prometheus IP and port

# Change the capacity config file and specify its path; details below
capacity.config.file=config/capacityCores.json
```

For express brokers, we recommend that you do not use the `DiskCapacityGoal` in any of the goals configured in your [analyzer configurations](#).

2. Edit the `config/capacityCores.json` file to specify the right disk size and CPU cores and network in/out limits. For Express brokers, the `DISK` capacity entry is only needed for setting up Cruise Control. Since MSK manages all the storage for Express brokers, you should set

this value to an extremely high number, such as `Integer.MAX_VALUE` (2147483647). For Standard brokers, you can use the [DescribeCluster](#) API operation (or [describe-cluster](#) CLI) to obtain the disk size. For CPU cores and network in/out limits, see [Amazon EC2 Instance Types](#).

Standard broker config/capacityCores.json

```
{
  "brokerCapacities": [
    {
      "brokerId": "-1",
      "capacity": {
        "DISK": "10000",
        "CPU": {
          "num.cores": "2"
        },
        "NW_IN": "5000000",
        "NW_OUT": "5000000"
      },
      "doc": "This is the default capacity. Capacity unit used for disk is in MB, cpu is in number of cores, network throughput is in KB."
    }
  ]
}
```

Express broker config/capacityCores.json

```
{
  "brokerCapacities": [
    {
      "brokerId": "-1",
      "capacity": {
        "DISK": "2147483647",
        "CPU": {"num.cores": "16"},
        "NW_IN": "1073741824",
        "NW_OUT": "1073741824"
      },
      "doc": "This is the default capacity. Capacity unit used for disk is in MB, cpu is in number of cores, network throughput is in KB."
    }
  ]
}
```

3. You can optionally install the Cruise Control UI. To download it, go to [Setting Up Cruise Control Frontend](#).
4. Run the following command to start Cruise Control. Consider using a tool like screen or tmux to keep a long-running session open.

```
<path-to-your-CRUISE-CONTROL-installation>/bin/kafka-cruise-control-start.sh  
config/cruisecontrol.properties 9091
```

5. Use the Cruise Control APIs or the UI to make sure that Cruise Control has the cluster load data and that it's making rebalancing suggestions. It might take several minutes to get a valid window of metrics.

 **Important**

Only Cruise Control versions 2.5.60 and above are compatible with Express brokers as Express brokers do not expose Zookeeper endpoints.

Use automated deployment template of Cruise Control for Amazon MSK

You can also use this [CloudFormation template](#), to easily deploy Cruise Control and Prometheus to gain deeper insights into your Amazon MSK cluster's performance and optimize resource utilization.

Key features:

- Automated provisioning of an Amazon EC2 instance with Cruise Control and Prometheus pre-configured.
- Support for Amazon MSK provisioned cluster.
- Flexible authentication with [PlainText and IAM](#).
- No Zookeeper dependency for Cruise Control.
- Easily customize Prometheus targets, Cruise Control capacity settings, and other configurations by providing your own configuration files stored in an Amazon S3 bucket.

Partition rebalancing guideline

Guidelines for Kafka partition reassignment

Partition reassignment in Kafka can be resource-intensive, as it involves transferring significant data across brokers, potentially causing network congestion and affecting client operations. The following best practices help you manage partition reassignment effectively by tuning throttle rates, leveraging concurrency controls, and understanding reassignment types to minimize disruption to cluster operations.

Managing concurrency in Cruise Control

Cruise Control provides auto-adjustment parameters to control the concurrency of partition and leadership movements. The following parameters help maintain an acceptable load during reassignments:

- **Maximum concurrent partition movements:** Define the `num.concurrent.partition.movements.per.broker` to cap concurrent inter-broker partition movements, avoiding excessive network utilization.

Example Example

```
num.concurrent.partition.movements.per.broker = 5
```

This setting limits each broker to move no more than 10 partitions at any given time, balancing the load across brokers.

Use throttling to control bandwidth

- **Throttle Parameter:** When performing partition reassignment with `kafka-reassign-partitions.sh`, use the `--throttle` parameter to set a maximum transfer rate (in bytes per second) for data movement between brokers.

Example Example

```
--throttle 5000000
```

This sets a maximum bandwidth of 5 MB/s.

- **Balance Throttle Settings:** Choosing an appropriate throttle rate is crucial:

If set too low, the reassignment may take significantly longer.

If set too high, clients may experience latency increases.

- Start with a conservative throttle rate and adjust based on cluster performance monitoring. Test your chosen throttle before applying to a production environment to find the optimal balance.

Test and validate in a staging environment

Prior to implementing reassignments in production, perform load tests in a staging environment with similar configurations. This allows you to fine-tune parameters and minimize unexpected impacts in live production.

Update the configuration of an Amazon MSK cluster

To update the configuration of a cluster, make sure that the cluster is in the ACTIVE state. You must also ensure that the number of partitions per broker on your MSK cluster is under the limits described in [the section called “Right-size your cluster: Number of partitions per Standard broker”](#). You can't update the configuration of a cluster that exceeds these limits.

For information about MSK configuration, including how to create a custom configuration, which properties you can update, and what happens when you update the configuration of an existing cluster, see [the section called “Broker configuration”](#).

Updating the configuration of a cluster using the AWS CLI

1. Copy the following JSON and save it to a file. Name the file `configuration-info.json`. Replace *ConfigurationArn* with the Amazon Resource Name (ARN) of the configuration that you want to use to update the cluster. The ARN string must be in quotes in the following JSON.

Replace *Configuration-Revision* with the revision of the configuration that you want to use. Configuration revisions are integers (whole numbers) that start at 1. This integer mustn't be in quotes in the following JSON.

```
{
  "Arn": ConfigurationArn,
  "Revision": Configuration-Revision
}
```

2. Run the following command, replacing *ClusterArn* with the ARN that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see [the section called "List clusters"](#).

Replace *Path-to-Config-Info-File* with the path to your configuration info file. If you named the file that you created in the previous step `configuration-info.json` and saved it in the current directory, then *Path-to-Config-Info-File* is `configuration-info.json`.

Replace *Current-Cluster-Version* with the current version of the cluster.

Important

Cluster versions aren't simple integers. To find the current version of the cluster, use the [DescribeCluster](#) operation or the [describe-cluster](#) AWS CLI command. An example version is `KTVPDKIKX0DER`.

```
aws kafka update-cluster-configuration --cluster-arn ClusterArn --configuration-  
info file://Path-to-Config-Info-File --current-version Current-Cluster-Version
```

The following is an example of how to use this command:

```
aws kafka update-cluster-configuration --cluster-arn "arn:aws:kafka:us-  
east-1:0123456789012:cluster/exampleName/abcd1234-0123-abcd-5678-1234abcd-1" --  
configuration-info file://c:\users\tester\msk\configuration-info.json --current-  
version "K1X5R6FKA87"
```

The output of this `update-cluster-configuration` command looks like the following JSON example.

```
{  
  "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/  
abcdefab-1234-abcd-5678-cdef0123ab01-2",  
  "ClusterOperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-  
operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-  
abcd-4f7f-1234-9876543210ef"  
}
```

3. To get the result of the `update-cluster-configuration` operation, run the following command, replacing `ClusterOperationArn` with the ARN that you obtained in the output of the `update-cluster-configuration` command.

```
aws kafka describe-cluster-operation --cluster-operation-arn ClusterOperationArn
```

The output of this `describe-cluster-operation` command looks like the following JSON example.

```
{
  "ClusterOperationInfo": {
    "ClientRequestId": "982168a3-939f-11e9-8a62-538df00285db",
    "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2",
    "CreationTime": "2019-06-20T21:08:57.735Z",
    "OperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-abcd-4f7f-1234-9876543210ef",
    "OperationState": "UPDATE_COMPLETE",
    "OperationType": "UPDATE_CLUSTER_CONFIGURATION",
    "SourceClusterInfo": {},
    "TargetClusterInfo": {
      "ConfigurationInfo": {
        "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/ExampleConfigurationName/abcdabcd-abcd-1234-abcd-abcd123e8e8e-1",
        "Revision": 1
      }
    }
  }
}
```

In this output, `OperationType` is `UPDATE_CLUSTER_CONFIGURATION`. If `OperationState` has the value `UPDATE_IN_PROGRESS`, wait a while, then run the `describe-cluster-operation` command again.

Update the configuration of a Amazon MSK cluster using the API

To use the API to update the configuration of a Amazon MSK cluster, see [UpdateClusterConfiguration](#).

Reboot a broker for an Amazon MSK cluster

Use this Amazon MSK operation when you want to reboot a broker for your MSK cluster. To reboot a broker for a cluster, make sure that the cluster is in the ACTIVE state.

The Amazon MSK service may reboot the brokers for your MSK cluster during system maintenance, such as patching or version upgrades. Rebooting a broker manually lets you test resilience of your Kafka clients to determine how they respond to system maintenance.

Reboot a broker for an Amazon MSK cluster using the AWS Management Console

This process describes how to reboot a broker for an Amazon MSK cluster using the AWS Management Console.

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. Choose the MSK cluster whose broker you want to reboot.
3. Scroll down to the **Broker details** section, and choose the broker you want to reboot.
4. Choose the **Reboot broker** button.

Reboot a broker for an Amazon MSK cluster using the AWS CLI

This process describes how to reboot a broker for an Amazon MSK cluster using the AWS CLI.

1. Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster, and the *BrokerId* with the ID of the broker that you want to reboot.

Note

The `reboot-broker` operation only supports rebooting one broker at a time.

If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see [the section called "List clusters"](#).

If you don't have the broker IDs for your cluster, you can find them by listing the broker nodes. For more information, see [list-nodes](#).

```
aws kafka reboot-broker --cluster-arn ClusterArn --broker-ids BrokerId
```

The output of this reboot-broker operation looks like the following JSON.

```
{
  "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/
  abcdefab-1234-abcd-5678-cdef0123ab01-2",
  "ClusterOperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
  operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
  abcd-4f7f-1234-9876543210ef"
}
```

2. To get the result of the reboot-broker operation, run the following command, replacing *ClusterOperationArn* with the ARN that you obtained in the output of the reboot-broker command.

```
aws kafka describe-cluster-operation --cluster-operation-arn ClusterOperationArn
```

The output of this describe-cluster-operation command looks like the following JSON example.

```
{
  "ClusterOperationInfo": {
    "ClientRequestId": "c0b7af47-8591-45b5-9c0c-909a1a2c99ea",
    "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/
    exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2",
    "CreationTime": "2019-09-25T23:48:04.794Z",
    "OperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
    operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
    abcd-4f7f-1234-9876543210ef",
    "OperationState": "REBOOT_IN_PROGRESS",
    "OperationType": "REBOOT_NODE",
    "SourceClusterInfo": {},
    "TargetClusterInfo": {}
  }
}
```

When the reboot operation is complete, the `OperationState` is `REBOOT_COMPLETE`.

Reboot a broker for an Amazon MSK cluster using the using the API

To reboot a broker in a cluster using the API, see [RebootBroker](#).

Tag an Amazon MSK cluster

You can assign your own metadata in the form of *tags* to an Amazon MSK resource, such as an MSK cluster. A tag is a key-value pair that you define for the resource. Using tags is a simple yet powerful way to manage AWS resources and organize data, including billing data.

Topics

- [Tag basics for Amazon MSK clusters](#)
- [Track Amazon MSK cluster costs using tagging](#)
- [Tag restrictions](#)
- [Tag resources using the Amazon MSK API](#)

Tag basics for Amazon MSK clusters

You can use the Amazon MSK API to complete the following tasks:

- Add tags to an Amazon MSK resource.
- List the tags for an Amazon MSK resource.
- Remove tags from an Amazon MSK resource.

You can use tags to categorize your Amazon MSK resources. For example, you can categorize your Amazon MSK clusters by purpose, owner, or environment. Because you define the key and value for each tag, you can create a custom set of categories to meet your specific needs. For example, you might define a set of tags that help you track clusters by owner and associated application.

The following are several examples of tags:

- Project: *Project name*
- Owner: *Name*
- Purpose: Load testing
- Environment: Production

Track Amazon MSK cluster costs using tagging

You can use tags to categorize and track your AWS costs. When you apply tags to your AWS resources, including Amazon MSK clusters, your AWS cost allocation report includes usage and costs aggregated by tags. You can organize your costs across multiple services by applying tags that represent business categories (such as cost centers, application names, or owners). For more information, see [Use Cost Allocation Tags for Custom Billing Reports](#) in the *AWS Billing User Guide*.

Tag restrictions

The following restrictions apply to tags in Amazon MSK.

Basic restrictions

- The maximum number of tags per resource is 50.
- Tag keys and values are case-sensitive.
- You can't change or edit tags for a deleted resource.

Tag key restrictions

- Each tag key must be unique. If you add a tag with a key that's already in use, your new tag overwrites the existing key-value pair.
- You can't start a tag key with `aws :` because this prefix is reserved for use by AWS. AWS creates tags that begin with this prefix on your behalf, but you can't edit or delete them.
- Tag keys must be between 1 and 128 Unicode characters in length.
- Tag keys must consist of the following characters: Unicode letters, digits, white space, and the following special characters: `_ . / = + - @`.

Tag value restrictions

- Tag values must be between 0 and 255 Unicode characters in length.
- Tag values can be blank. Otherwise, they must consist of the following characters: Unicode letters, digits, white space, and any of the following special characters: `_ . / = + - @`.

Tag resources using the Amazon MSK API

You can use the following operations to tag or untag an Amazon MSK resource or to list the current set of tags for a resource:

- [ListTagsForResource](#)
- [TagResource](#)
- [UntagResource](#)

Migrate to an Amazon MSK Cluster

Amazon MSK Replicator can be used for MSK cluster migration. See [What is Amazon MSK Replicator?](#). Alternatively, you can use Apache MirrorMaker 2.0 to migrate from a non-MSK cluster to an Amazon MSK cluster. For an example of how to do this, see [Migrate an on-premises Apache Kafka cluster to Amazon MSK by using MirrorMaker](#). For information about how to use MirrorMaker, see [Mirroring data between clusters](#) in the Apache Kafka documentation. We recommend setting up MirrorMaker in a highly available configuration.

An outline of the steps to follow when using MirrorMaker to migrate to an MSK cluster

1. Create the destination MSK cluster
2. Start MirrorMaker from an Amazon EC2 instance within the same Amazon VPC as the destination cluster.
3. Inspect the MirrorMaker lag.
4. After MirrorMaker catches up, redirect producers and consumers to the new cluster using the MSK cluster bootstrap brokers.
5. Shut down MirrorMaker.

Migrate your Apache Kafka cluster to Amazon MSK

Suppose that you have an Apache Kafka cluster named `CLUSTER_ONPREM`. That cluster is populated with topics and data. If you want to migrate that cluster to a newly created Amazon MSK cluster named `CLUSTER_AWSMSK`, this procedure provides a high-level view of the steps that you need to follow.

To migrate your existing Apache Kafka cluster to Amazon MSK

1. In CLUSTER_AWSMSK, create all the topics that you want to migrate.

You can't use MirrorMaker for this step because it doesn't automatically re-create the topics that you want to migrate with the right replication level. You can create the topics in Amazon MSK with the same replication factors and numbers of partitions that they had in CLUSTER_ONPREM. You can also create the topics with different replication factors and numbers of partitions.

2. Start MirrorMaker from an instance that has read access to CLUSTER_ONPREM and write access to CLUSTER_AWSMSK.
3. Run the following command to mirror all topics:

```
<path-to-your-kafka-installation>/bin/kafka-mirror-maker.sh --consumer.config  
config/mirrormaker-consumer.properties --producer.config config/mirrormaker-  
producer.properties --whitelist '.*'
```

In this command, `config/mirrormaker-consumer.properties` points to a bootstrap broker in CLUSTER_ONPREM; for example, `bootstrap.servers=localhost:9092`. And `config/mirrormaker-producer.properties` points to a bootstrap broker in CLUSTER_AWSMSK; for example, `bootstrap.servers=10.0.0.237:9092,10.0.2.196:9092,10.0.1.233:9092`.

4. Keep MirrorMaker running in the background, and continue to use CLUSTER_ONPREM. MirrorMaker mirrors all new data.
5. Check the progress of mirroring by inspecting the lag between the last offset for each topic and the current offset from which MirrorMaker is consuming.

Remember that MirrorMaker is simply using a consumer and a producer. So, you can check the lag using the `kafka-consumer-groups.sh` tool. To find the consumer group name, look inside the `mirrormaker-consumer.properties` file for the `group.id`, and use its value. If there is no such key in the file, you can create it. For example, set `group.id=mirrormaker-consumer-group`.

6. After MirrorMaker finishes mirroring all topics, stop all producers and consumers, and then stop MirrorMaker. Then redirect the producers and consumers to the CLUSTER_AWSMSK cluster by changing their producer and consumer bootstrap brokers values. Restart all producers and consumers on CLUSTER_AWSMSK.

Migrate from one Amazon MSK cluster to another

You can use Apache MirrorMaker 2.0 to migrate from a non-MSK cluster to a MSK cluster. For example, you can migrate from one version of Apache Kafka to another. For an example of how to do this, see [Migrate an on-premises Apache Kafka cluster to Amazon MSK by using MirrorMaker](#). Alternatively, Amazon MSK Replicator can be used for MSK cluster migration. For more information about Amazon MSK Replicator, see [What is Amazon MSK Replicator?](#).

MirrorMaker 1.0 best practices

This list of best practices applies to MirrorMaker 1.0.

- Run MirrorMaker on the destination cluster. This way, if a network problem happens, the messages are still available in the source cluster. If you run MirrorMaker on the source cluster and events are buffered in the producer and there is a network issue, events might be lost.
- If encryption is required in transit, run it in the source cluster.
- For consumers, set `auto.commit.enabled=false`
- For producers, set
 - `max.in.flight.requests.per.connection=1`
 - `retries=Int.MaxValue`
 - `acks=all`
 - `max.block.ms = Long.MaxValue`
- For a high producer throughput:
 - Buffer messages and fill message batches — tune `buffer.memory`, `batch.size`, `linger.ms`
 - Tune socket buffers — `receive.buffer.bytes`, `send.buffer.bytes`
- To avoid data loss, turn off auto commit at the source, so that MirrorMaker can control the commits, which it typically does after it receives the ack from the destination cluster. If the producer has `acks=all` and the destination cluster has `min.insync.replicas` set to more than 1, the messages are persisted on more than one broker at the destination before the MirrorMaker consumer commits the offset at the source.
- If order is important, you can set `retries` to 0. Alternatively, for a production environment, set max inflight connections to 1 to ensure that the batches sent out are not committed out of order if a batch fails in the middle. This way, each batch sent is retried until the next batch is sent out. If `max.block.ms` is not set to the maximum value, and if the producer buffer is full, there can

be data loss (depending on some of the other settings). This can block and back-pressure the consumer.

- For high throughput
 - Increase `buffer.memory`.
 - Increase batch size.
 - Tune `linger.ms` to allow the batches to fill. This also allows for better compression, less network bandwidth usage, and less storage on the cluster. This results in increased retention.
 - Monitor CPU and memory usage.
- For high consumer throughput
 - Increase the number of threads/consumers per MirrorMaker process — `num.streams`.
 - Increase the number of MirrorMaker processes across machines first before increasing threads to allow for high availability.
 - Increase the number of MirrorMaker processes first on the same machine and then on different machines (with the same group ID).
 - Isolate topics that have very high throughput and use separate MirrorMaker instances.
- For management and configuration
 - Use AWS CloudFormation and configuration management tools like Chef and Ansible.
 - Use Amazon EFS mounts to keep all configuration files accessible from all Amazon EC2 instances.
 - Use containers for easy scaling and management of MirrorMaker instances.
- Typically, it takes more than one consumer to saturate a producer in MirrorMaker. So, set up multiple consumers. First, set them up on different machines to provide high availability. Then, scale individual machines up to having a consumer for each partition, with consumers equally distributed across machines.
- For high throughput ingestion and delivery, tune the receive and send buffers because their defaults might be too low. For maximum performance, ensure that the total number of streams (`num.streams`) matches all of the topic partitions that MirrorMaker is trying to copy to the destination cluster.

Advantages of MirrorMaker 2.*

- Makes use of the Apache Kafka Connect framework and ecosystem.
- Detects new topics and partitions.

- Automatically syncs topic configuration between clusters.
- Supports "active/active" cluster pairs, as well as any number of active clusters.
- Provides new metrics including end-to-end replication latency across multiple data centers and clusters.
- Emits offsets required to migrate consumers between clusters and provides tooling for offset translation.
- Supports a high-level configuration file for specifying multiple clusters and replication flows in one place, compared to low-level producer/consumer properties for each MirrorMaker 1.* process.

Delete an Amazon MSK Provisioned cluster

Note

If your provisioned Amazon MSK cluster has an auto-scaling policy, we recommend that you remove the policy before you delete the cluster. For more information, see [Automatic scaling for Amazon MSK clusters](#).

Topics

- [Delete an Amazon MSK Provisioned cluster using the AWS Management Console](#)
- [Delete an Amazon MSK Provisioned cluster using the AWS CLI](#)
- [Delete an Amazon MSK Provisioned cluster using the API](#)

Delete an Amazon MSK Provisioned cluster using the AWS Management Console

This process describes how to delete an Amazon MSK Provisioned cluster using the AWS Management Console. Before you delete a MSK cluster, ensure that you have a backup of any important data stored in the cluster and that there aren't any scheduled tasks dependant on the cluster. You can't undo a MSK cluster deletion.

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose the MSK cluster that you want to delete by selecting the check box next to it.

3. Choose **Delete**, and then confirm deletion.

Delete an Amazon MSK Provisioned cluster using the AWS CLI

This process describes how to delete an MSK Provisioned cluster using the AWS CLI. Before you delete a MSK cluster, ensure that you have a backup of any important data stored in the cluster and that there aren't any scheduled tasks dependant on the cluster. You can't undo a MSK cluster deletion.

Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see [the section called "List clusters"](#).

```
aws kafka delete-cluster --cluster-arn ClusterArn
```

Delete an Amazon MSK Provisioned cluster using the API

The Amazon MSK API allows you to programmatically create and manage your MSK Provisioned cluster as part of automated infrastructure provisioning or deployment scripts. This process describes how to delete an Amazon MSK Provisioned cluster using the Amazon MSK API. Before you delete a Amazon MSK cluster, ensure that you have a backup of any important data stored in the cluster and that there aren't any scheduled tasks dependant on the cluster. You can't undo a MSK cluster deletion.

To delete a cluster using the API, see [DeleteCluster](#).

Amazon MSK key features and concepts

Amazon MSK Provisioned clusters offer a wide range of features and capabilities to help you optimize your cluster's performance and meet your streaming needs. The topics below describe these functionalities in detail.

- The [AWS Management Console](#)
- The [Amazon MSK API Reference](#)
- The [Amazon MSK CLI Command Reference](#)

Topics

- [Amazon MSK broker types](#)
- [Amazon MSK broker sizes](#)
- [Storage management for Standard brokers](#)
- [Security in Amazon MSK](#)
- [Amazon MSK Provisioned configuration](#)
- [Patching](#)
- [Broker offline and client failover](#)
- [Amazon MSK logging](#)
- [Metadata management](#)
- [Amazon MSK resources](#)
- [Apache Kafka versions](#)
- [Troubleshoot your Amazon MSK cluster](#)

Amazon MSK broker types

MSK Provisioned offers two broker types - Standard and Express. Standard brokers give you the most flexibility to configure your clusters, while Express brokers offer more elasticity, throughput, resilience, and ease-of-use for running high performance streaming applications. See the sub-sections below for more details on each offering. The table below also highlights the key feature comparison between Standard and Express brokers.

MSK Provisioned Broker Types Comparison

Feature	Standard broker	Express broker
Storage Management	Customer managed (Features include EBS storage, Tiered storage, Provisioned storage throughput, Auto-scaling, Storage capacity alerts)	Fully MSK managed
Supported instances	T3, M5, M7g	M7g
Sizing and scaling considerations	Throughput, connections, partitions, storage	Throughput, connections, partitions

Feature	Standard broker	Express broker
Broker scaling	Vertical and horizontal scaling	Vertical and horizontal scaling
Kafka versions	See Apache Kafka versions	Starts at version 3.6
Apache Kafka Configuration	More configurable	Mostly MSK managed for higher resilience
Security	Encryption, Private/Public access, Authentication & Authorization - IAM, SASL/SCRAM, mTLS, plaintext, Kafka ACLs	Encryption, Private/Public access, Authentication & Authorization - IAM, SASL/SCRAM, mTLS, plaintext, Kafka ACLs
Monitoring	CloudWatch, Open Monitoring	CloudWatch, Open Monitoring

Note

You can't change an MSK Provisioned cluster from a Standard broker type to an Express broker type by switching the broker type using the MSK API. You have to create a new cluster with the desired broker type (Standard or Express).

Topics

- [Amazon MSK Standard brokers](#)
- [Amazon MSK Express brokers](#)

Amazon MSK Standard brokers

Standard brokers for MSK Provisioned offer the most flexibility to configure your cluster's performance. You can choose from a wide range cluster configurations to achieve the availability, durability, throughput, and latency characteristics required for your applications. You can also provision storage capacity and increase it as needed. Amazon MSK handles the hardware maintenance of Standard brokers and attached storage resources, automatically repairing hardware issues that may arise. You can find more details in this document about various topics

related to Standard brokers, including topics on [storage management](#), [configurations](#), and [maintenance](#).

Amazon MSK Express brokers

Express brokers for MSK Provisioned make Apache Kafka simpler to manage, more cost-effective to run at scale, and more elastic with the low latency you expect. Brokers include pay-as-you-go storage that scales automatically and requires no sizing, provisioning, or proactive monitoring. Depending on the instance size selected, each broker node can provide up to 3x more throughput per broker, scale up to 20x faster, and recover 90% quicker compared to standard Apache Kafka brokers. Express brokers come pre-configured with Amazon MSK's best practice defaults and enforce client throughput quotas to minimize resource contention between clients and Kafka's background operations.

Here are some key factors and capabilities to consider when using Express brokers.

- **No storage management:** Express brokers eliminate the need to [provision or manage any storage resources](#). You get elastic, virtually unlimited, pay-as-you-go, and fully managed storage. For high throughput use cases, you do not need to reason about the interactions between compute instances and storage volumes, and the associated throughput bottlenecks. These capabilities simplify cluster management and eliminate storage management operational overhead.
- **Faster scaling:** Express brokers allow you to scale your cluster and move partitions up to 20x faster than on Standard brokers. This capability is crucial when you need to scale out your cluster to handle upcoming load spikes or scale in your cluster to reduce cost. See the sections on [expanding your cluster](#), [removing brokers](#), [reassigning partitions](#), and [setting up LinkedIn's Cruise Control for rebalancing](#) for more details on scaling your cluster.
- **Higher throughput:** Express brokers offer up to 3x more throughput per broker than Standard brokers. For example, you can safely write data at up to 500 MBps with each m7g.16xlarge sized Express broker compared to 153.8 MBps on the equivalent Standard broker (both numbers assume sufficient bandwidth allocation towards background operations, such as replication and rebalancing).
- **Configured for high resilience:** Express brokers automatically offer various best practices to improve your cluster's resilience. These include guardrails on critical Apache Kafka configurations, throughput quotas, and capacity reservations for background operations and unplanned repairs. These capabilities make it safer and easier to run large scale Apache Kafka

applications. See the sections on [Express broker configurations](#) and [Amazon MSK Express broker quota](#) for more details.

- **No Maintenance windows:** There are no maintenance windows for Express brokers. Amazon MSK automatically updates your cluster hardware on an ongoing basis. See [Patching for Express brokers](#) for more details.

Additional information about Express brokers

- Express brokers work with Apache Kafka APIs, but don't yet fully support KStreams API.
- Express brokers are only available in a 3AZs configuration.
- Express brokers are only available on select instance sizes. See [Amazon MSK pricing](#) for the updated list.
- Express brokers are supported on Apache Kafka version 3.6.

Amazon MSK broker sizes

When you create an Amazon MSK Provisioned cluster you specify the size of brokers that you want it to have. Depending on the [broker type](#), Amazon MSK supports the following broker sizes.

Standard broker sizes

- kafka.t3.small
- kafka.m5.large, kafka.m5.xlarge, kafka.m5.2xlarge, kafka.m5.4xlarge, kafka.m5.8xlarge, kafka.m5.12xlarge, kafka.m5.16xlarge, kafka.m5.24xlarge
- kafka.m7g.large, kafka.m7g.xlarge, kafka.m7g.2xlarge, kafka.m7g.4xlarge, kafka.m7g.8xlarge, kafka.m7g.12xlarge, kafka.m7g.16xlarge

Express broker sizes

- express.m7g.large, express.m7g.xlarge, express.m7g.2xlarge, express.m7g.4xlarge, express.m7g.8xlarge, express.m7g.12xlarge, express.m7g.16xlarge

Note

Some broker sizes may not be available in certain AWS Regions. See the updated Broker Instance Pricing Tables on the [Amazon MSK pricing page](#) for the latest list of available instances by Region.

Other notes on broker sizes

- M7g brokers use AWS Graviton processors (custom Arm-based processors built by Amazon Web Services). M7g brokers offer improved price performance relative to comparable M5 instances. M7g brokers consume less power than comparable M5 instances.
- Amazon MSK supports M7g brokers on MSK Provisioned clusters running 2.8.2 and 3.3.2 and higher Kafka versions.
- M7g and M5 brokers have higher baseline throughput performance than T3 brokers and are recommended for production workloads. M7g and M5 brokers can also have more partitions per broker than T3 brokers. Use M7g or M5 brokers if you are running larger production-grade workloads or require a greater number of partitions. To learn more about M7g and M5 instance sizes, see [Amazon EC2 General Purpose Instances](#).
- T3 brokers have the ability to use CPU credits to temporarily burst performance. Use T3 brokers for low-cost development, if you are testing small to medium streaming workloads, or if you have low-throughput streaming workloads that experience temporary spikes in throughput. We recommend that you run a proof-of-concept test to determine if T3 brokers are sufficient for production or critical workload. To learn more about T3 broker sizes, see [Amazon EC2 T3 Instances](#).

For more information on how to choose broker sizes, see [Best practices for Standard and Express brokers](#).

Storage management for Standard brokers

Amazon MSK provides features to help you with storage management on your MSK clusters.

Note

With [Express brokers](#), you don't need to provision or manage any storage resources used for your data. This simplifies cluster management and eliminates one of the common causes

of operational issues with Apache Kafka clusters. You also spend less as you don't have to provision idle storage capacity and you only pay for what you use.

Standard broker type

With [Standard brokers](#) you can choose from a variety of storage options and capabilities. Amazon MSK provides features to help you with storage management on your MSK clusters.

For information about managing throughput, see [???](#).

Topics

- [Tiered storage for Standard brokers](#)
- [Scale up Amazon MSK Standard broker storage](#)
- [Manage storage throughput for Standard brokers in a Amazon MSK cluster](#)

Tiered storage for Standard brokers

Tiered storage is a low-cost storage tier for Amazon MSK that scales to virtually unlimited storage, making it cost-effective to build streaming data applications.

You can create an Amazon MSK cluster configured with tiered storage that balances performance and cost. Amazon MSK stores streaming data in a performance-optimized primary storage tier until it reaches the Apache Kafka topic retention limits. Then, Amazon MSK automatically moves data into the new low-cost storage tier.

When your application starts reading data from the tiered storage, you can expect an increase in read latency for the first few bytes. As you start reading the remaining data sequentially from the low-cost tier, you can expect latencies that are similar to the primary storage tier. You don't need to provision any storage for the low-cost tiered storage or manage the infrastructure. You can store any amount of data and pay only for what you use. This feature is compatible with the APIs introduced in [KIP-405: Kafka Tiered Storage](#).

For information about sizing, monitoring, and optimizing your MSK tiered storage cluster, see [Best practices for running production workloads using Amazon MSK tiered storage](#).

Here are some of the features of tiered storage:

- You can scale to virtually unlimited storage. You don't have to guess how to scale your Apache Kafka infrastructure.
- You can retain data longer in your Apache Kafka topics, or increase your topic storage, without the need to increase the number of brokers.
- It provides a longer duration safety buffer to handle unexpected delays in processing.
- You can reprocess old data in its exact production order with your existing stream processing code and Kafka APIs.
- Partitions rebalance faster because data on secondary storage doesn't require replication across broker disks.
- Data between brokers and the tiered storage moves within the VPC and doesn't travel through the internet.
- A client machine can use the same process to connect to new clusters with tiered storage enabled as it does to connect to a cluster without tiered storage enabled. See [Create a client machine](#).

Tiered storage requirements for Amazon MSK clusters

- You must use Apache Kafka client version 3.0.0 or higher to create a new topic with tiered storage enabled. To transition an existing topic to tiered storage, you can reconfigure a client machine that uses a Kafka client version lower than 3.0.0 (minimum supported Apache Kafka version is 2.8.2.tiered) to enable tiered storage. See [Step 4: Create a topic in the Amazon MSK cluster](#).
- The Amazon MSK cluster with tiered storage enabled must use version 3.6.0 or higher, or 2.8.2.tiered.

Tiered storage constraints and limitations for Amazon MSK clusters

Tiered storage has the following constraints and limitations:

- Make sure clients are not configured to `read_committed` when reading from the `remote_tier` in Amazon MSK, unless the application is actively using the transactions feature.
- Tiered storage isn't available in AWS GovCloud (US) regions.
- Tiered storage applies only to provisioned mode clusters.
- Tiered storage doesn't support broker size `t3.small`.

- The minimum retention period in low-cost storage is 3 days. There is no minimum retention period for primary storage.
- Tiered storage doesn't support Multiple Log directories on a broker (JBOD related features).
- Tiered storage doesn't support compacted topics. Make sure that all topics that have tiered storage turned on have their `cleanup.policy` configured to 'DELETE' only.
- Tiered storage cluster doesn't support altering the `log.cleanup.policy` policy for a topic after it's created.
- Tiered storage can be disabled for individual topics but not for the entire cluster. Once disabled, tiered storage cannot be re-enabled for a topic.
- If you use Amazon MSK version 2.8.2.tiered, you can migrate only to another tiered storage-supported Apache Kafka version. If you don't want to continue using a tiered storage-supported version, create a new MSK cluster and migrate your data to it.
- The `kafka-log-dirs` tool can't report tiered storage data size. The tool only reports the size of the log segments in primary storage.

For information about default settings and constraints you must be mindful of when you configure tiered storage at the topic level, see [Guidelines for Amazon MSK tiered storage topic-level configuration](#).

How log segments are copied to tiered storage for a Amazon MSK topic

When you enable tiered storage for a new or existing topic, Apache Kafka copies closed log segments from primary storage to tiered storage.

- Apache Kafka only copies closed log segments. It copies all messages within the log segment to tiered storage.
- Active segments are not eligible for tiering. The log segment size (`segment.bytes`) or the segment roll time (`segment.ms`) controls the rate of segment closure, and the rate Apache Kafka then copies them to tiered storage.

Retention settings for a topic with tiered storage enabled are different from settings for a topic without tiered storage enabled. The following rules control the retention of messages in topics with tiered storage enabled:

- You define retention in Apache Kafka with two settings: `log.retention.ms` (time) and `log.retention.bytes` (size). These settings determine the total duration and size of the data that

Apache Kafka retains in the cluster. Whether or not you enable tiered storage mode, you set these configurations at the cluster level. You can override the settings at the topic level with topic configurations.

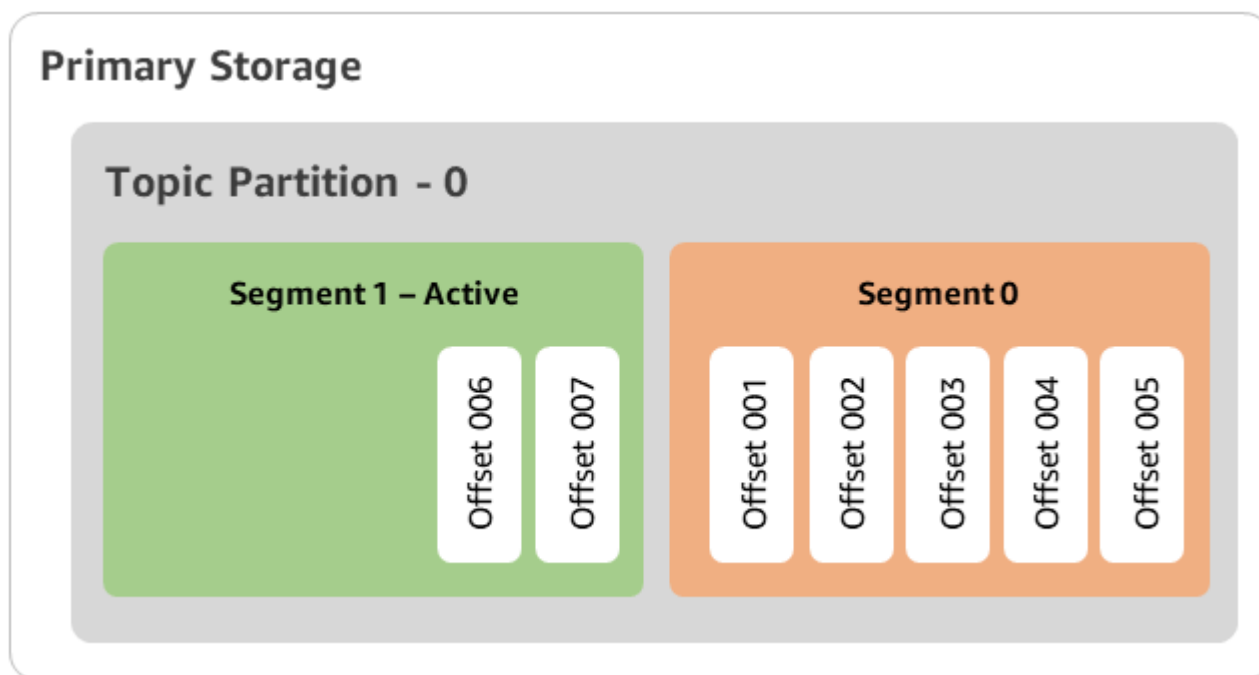
- When you enable tiered storage, you can additionally specify how long the primary high-performance storage tier stores data. For example, if a topic has overall retention (`log.retention.ms`) setting of 7 days and local retention (`local.retention.ms`) of 12 hours, then the cluster primary storage retains data for only the first 12 hours. The low-cost storage tier retains the data for the full 7 days.
- The usual retention settings apply to the full log. This includes its tiered and primary parts.
- The `local.retention.ms` or `local.retention.bytes` settings control the retention of messages in primary storage. When data has reached primary storage retention setting thresholds (`local.retention.ms/bytes`) on a full log, Apache Kafka copies the data in primary storage to tiered storage. The data is then eligible for expiration.
- When Apache Kafka copies a message in a log segment to tiered storage, it removes the message from the cluster based on `retention.ms` or `retention.bytes` settings.

Example Amazon MSK tiered storage scenario

This scenario illustrates how an existing topic that has messages in primary storage behaves when tiered storage is enabled. You enable tiered storage on this topic by when you set `remote.storage.enable` to `true`. In this example, `retention.ms` is set to 5 days and `local.retention.ms` is set to 2 days. The following is the sequence of events when a segment expires.

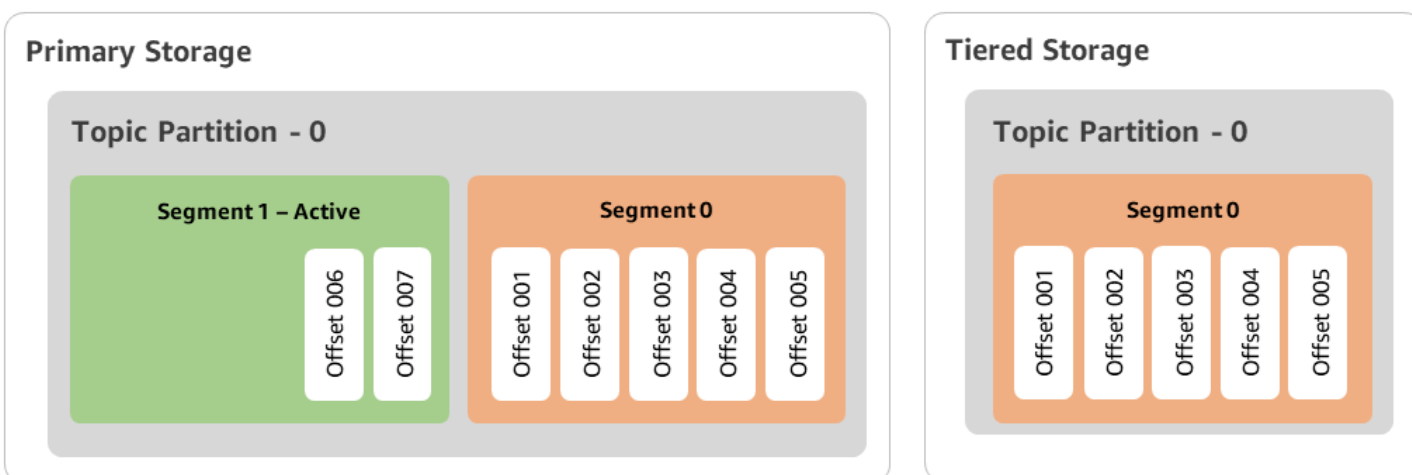
Time T0 - Before you enable tiered storage.

Before you enable tiered storage for this topic, there are two log segments. One of the segments is active for an existing topic partition 0.



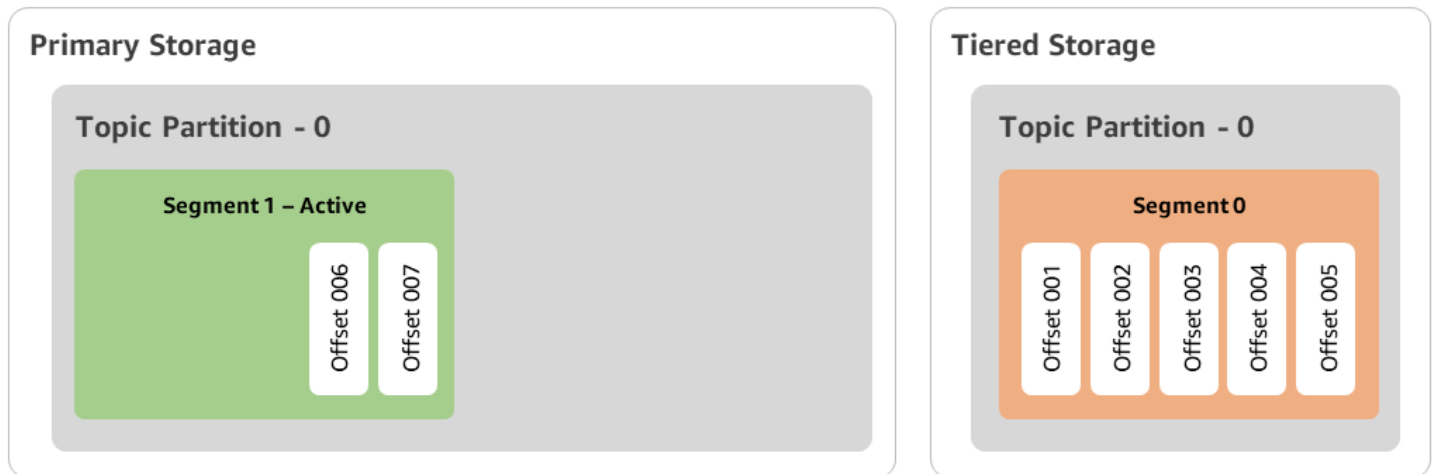
Time T1 (< 2 days) - Tiered storage enabled. Segment 0 copied to tiered storage.

After you enable tiered storage for this topic, Apache Kafka copies log segment 0 to tiered storage after the segment meets initial retention settings. Apache Kafka also retains the primary storage copy of segment 0. The active segment 1 is not eligible to copy over to tiered storage yet. In this timeline, Amazon MSK doesn't apply any of the retention settings yet for any of the messages in segment 0 and segment 1. (local.retention.bytes/ms, retention.ms/bytes)



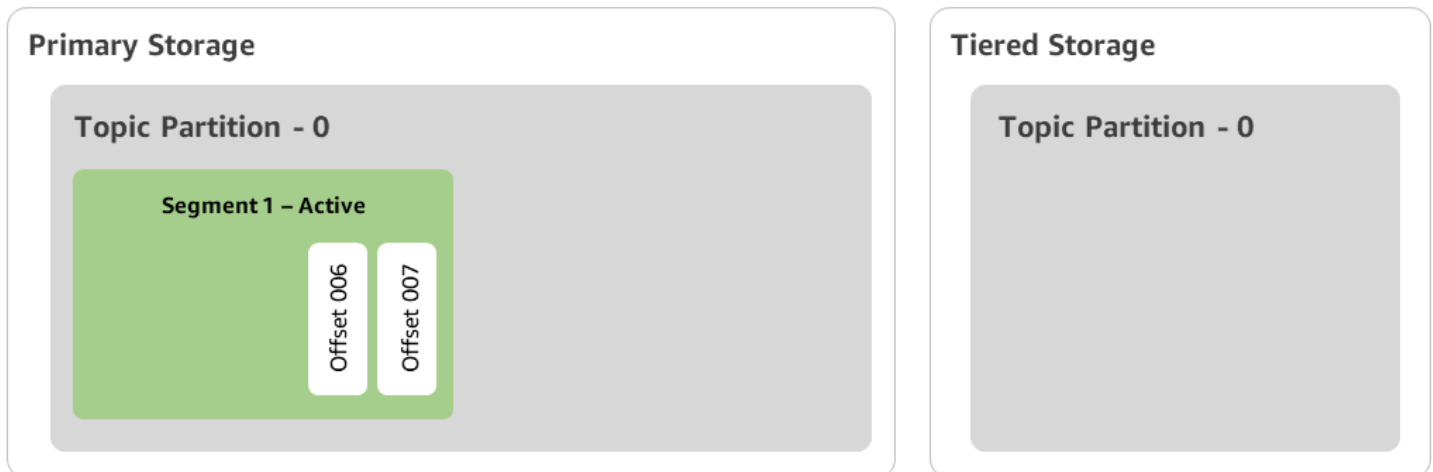
Time T2 - Local retention in effect.

After 2 days, primary retention settings take effect for the segment 0 that Apache Kafka copied to the tiered storage. The setting of `local.retention.ms` as 2 days determines this. Segment 0 now expires from the primary storage. Active segment 1 is neither eligible for expiration nor eligible to copy over to tiered storage yet.



Time T3 - Overall retention in effect.

After 5 days, retention settings take effect, and Kafka clears log segment 0 and associated messages from tiered storage. Segment 1 is neither eligible for expiration nor eligible to copy over to tiered storage yet because it is active. Segment 1 is not yet closed, so it is ineligible for segment roll.



Create a Amazon MSK cluster with tiered storage with the AWS Management Console

This process describes how to create a tiered storage Amazon MSK cluster using the AWS Management Console.

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. Choose **Create cluster**.
3. Choose **Custom create** for tiered storage.
4. Specify a name for the cluster.
5. In the **Cluster type**, select **Provisioned**.
6. Choose an Amazon Kafka version that supports tiered storage for Amazon MSK to use to create the cluster.
7. Specify a size of broker other than **kafka.t3.small**.
8. Select the number of brokers that you want Amazon MSK to create in each Availability Zone. The minimum is one broker per Availability Zone, and the maximum is 30 brokers per cluster.
9. Specify the number of zones that brokers are distributed across.
10. Specify the number of Apache Kafka brokers that are deployed per zone.
11. Select **Storage options**. This includes **Tiered storage and EBS storage** to enable tiered storage mode.
12. Follow the remaining steps in the cluster creation wizard. When complete, **Tiered storage and EBS storage** appears as the cluster storage mode in the **Review and create** view.
13. Select **Create cluster**.

Create an Amazon MSK cluster with tiered storage with the AWS CLI

To enable tiered storage on a cluster, create the cluster with the correct Apache Kafka version and attribute for tiered storage. Follow the code example below. Also, complete the steps in the next section to [Create a Kafka topic with tiered storage enabled with the AWS CLI](#).

See [create-cluster](#) for a complete list of supported attributes for cluster creation.

```
aws kafka create-cluster \  
  --cluster-name "MessagingCluster" \  
  --broker-node-group-info file://broker-node-group-info.json \  
  --number-of-broker-nodes 3 \  
  --kafka-version "3.6.0" \  
  --storage-mode "TIERED"
```

Create a Kafka topic with tiered storage enabled with the AWS CLI

To complete the process that you started when you created a cluster with the tiered storage enabled, also create a topic with tiered storage enabled with the attributes in the later code example. The attributes specifically for tiered storage are the following:

- `local.retention.ms` (for example, 10 mins) for time-based retention settings or `local.retention.bytes` for log segment size limits.
- `remote.storage.enable` set to `true` to enable tiered storage.

The following configuration uses `local.retention.ms`, but you can replace this attribute with `local.retention.bytes`. This attribute controls the amount of time that can pass or number of bytes that Apache Kafka can copy before Apache Kafka copies the data from primary to tiered storage. See [Topic-level configuration](#) for more details on supported configuration attributes.

Note

You must use the Apache Kafka client version 3.0.0 and above. These versions support a setting called `remote.storage.enable` only in those client versions of `kafka-topics.sh`. To enable tiered storage on an existing topic that uses an earlier version of Apache Kafka, see the section [Enabling tiered storage on an existing Amazon MSK topic](#).

```
bin/kafka-topics.sh --create --bootstrap-server $bs --replication-factor 2
--partitions 6 --topic MSKTutorialTopic --config remote.storage.enable=true
--config local.retention.ms=1000000 --config retention.ms=604800000 --config
segment.bytes=134217728
```

Enable and disable tiered storage on an existing Amazon MSK topic

These sections cover how to enable and disable tiered storage on a topic that you've already created. To create a new cluster and topic with tiered storage enabled, see [Creating a cluster with tiered storage using the AWS Management Console](#).

Enabling tiered storage on an existing Amazon MSK topic

To enable tiered storage on an existing topic, use the `alter` command syntax in the following example. When you enable tiered storage on an already existing topic, you aren't restricted to a certain Apache Kafka client version.

```
bin/kafka-configs.sh --bootstrap-server $bsrv --alter --entity-type topics
--entity-name msk-ts-topic --add-config 'remote.storage.enable=true,
local.retention.ms=604800000, retention.ms=15550000000'
```

Disable tiered storage on an existing Amazon MSK topic

To disable tiered storage on an existing topic, use the `alter` command syntax in the same order as when you enable tiered storage.

```
bin/kafka-configs.sh --bootstrap-server $bs --alter --entity-type topics --
entity-name MSKTutorialTopic --add-config 'remote.log.msk.disable.policy=Delete,
remote.storage.enable=false'
```

Note

When you disable tiered storage, you completely delete the topic data in tiered storage. Apache Kafka retains primary storage data, but it still applies the primary retention rules based on `local.retention.ms`. After you disable tiered storage on a topic, you can't re-enable it. If you want to disable tiered storage on an existing topic, you aren't restricted to a certain Apache Kafka client version.

Enable tiered storage on an existing Amazon MSK cluster using AWS CLI

Note

You can enable tiered storage only if your cluster's `log.cleanup.policy` is set to `delete`, as compacted topics are not supported on tiered storage. Later, you can configure an individual topic's `log.cleanup.policy` to `compact` if tiered storage is not enabled on that particular topic. See [Topic-level configuration](#) for more details on supported configuration attributes.

1. **Update the Kafka version** – Cluster versions aren't simple integers. To find the current version of the cluster, use the `DescribeCluster` operation or the `describe-cluster` AWS CLI command. An example version is `KTVPDKIKX0DER`.

```
aws kafka update-cluster-kafka-version --cluster-arn ClusterArn --current-version  
Current-Cluster-Version --target-kafka-version 3.6.0
```

2. Edit cluster storage mode. The following code example shows editing the cluster storage mode to TIERED using the [update-storage](#) API.

```
aws kafka update-storage --current-version Current-Cluster-Version --cluster-arn  
Cluster-arn --storage-mode TIERED
```

Update tiered storage on an existing Amazon MSK cluster using the console

This process describes how to update a tiered storage Amazon MSK cluster using the AWS Management Console.

Make sure the current Apache Kafka version of your MSK cluster is 2.8.2.tiered. Refer to [updating the Apache Kafka version](#) if you need to upgrade your MSK cluster to 2.8.2.tiered version.

Note

You can enable tiered storage only if your cluster's `log.cleanup.policy` is set to `delete`, as compacted topics are not supported on tiered storage. Later, you can configure an individual topic's `log.cleanup.policy` to `compact` if tiered storage is not enabled on that particular topic. See [Topic-level configuration](#) for more details on supported configuration attributes.

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. Go to the cluster summary page and choose **Properties**.
3. Go to the **Storage** section and choose **Edit cluster storage mode**.
4. Choose **Tiered storage and EBS storage** and **Save changes**.

Scale up Amazon MSK Standard broker storage

You can increase the amount of EBS storage per broker. You can't decrease the storage.

Storage volumes remain available during this scaling-up operation.

⚠ Important

When storage is scaled for an MSK cluster, the additional storage is made available right away. However, the cluster requires a cool-down period after every storage scaling event. Amazon MSK uses this cool-down period to optimize the cluster before it can be scaled again. This period can range from a minimum of 6 hours to over 24 hours, depending on the cluster's storage size and utilization and on traffic. This is applicable for both auto scaling events and manual scaling using the [UpdateBrokerStorage](#) operation. For information about right-sizing your storage, see [the section called “Best practices for Standard brokers”](#).

You can use tiered storage to scale up to unlimited amounts of storage for your broker. See, [Tiered storage for Standard brokers](#).

Topics

- [Automatic scaling for Amazon MSK clusters](#)
- [Manual scaling for Standard brokers](#)

Automatic scaling for Amazon MSK clusters

To automatically expand your cluster's storage in response to increased usage, you can configure an Application Auto-Scaling policy for Amazon MSK. In an auto-scaling policy, you set the target disk utilization and the maximum scaling capacity.

Before you use automatic scaling for Amazon MSK, you should consider the following:

- **⚠ Important**
A storage scaling action can occur only once every six hours.

We recommend that you start with a right-sized storage volume for your storage demands. For guidance on right-sizing your cluster, see [Right-size your cluster: Number of Standard brokers per cluster](#).

- Amazon MSK does not reduce cluster storage in response to reduced usage. Amazon MSK does not support decreasing the size of storage volumes. If you need to reduce the size of your cluster

storage, you must migrate your existing cluster to a cluster with smaller storage. For information about migrating a cluster, see [Migrate to Amazon MSK Cluster](#).

- Amazon MSK does not support automatic scaling in the Asia Pacific (Osaka) and Africa (Cape Town) Regions.
- When you associate an auto-scaling policy with your cluster, Amazon EC2 Auto Scaling automatically creates an Amazon CloudWatch alarm for target tracking. If you delete a cluster with an auto-scaling policy, this CloudWatch alarm persists. To delete the CloudWatch alarm, you should remove an auto-scaling policy from a cluster before you delete the cluster. To learn more about target tracking, see [Target tracking scaling policies for Amazon EC2 Auto Scaling](#) in the *Amazon EC2 Auto Scaling User Guide*.

Topics

- [Auto-scaling policy details for Amazon MSK](#)
- [Set up automatic scaling for your Amazon MSK cluster](#)

Auto-scaling policy details for Amazon MSK

An auto-scaling policy defines the following parameters for your cluster:

- **Storage Utilization Target:** The storage utilization threshold that Amazon MSK uses to trigger an auto-scaling operation. You can set the utilization target between 10% and 80% of the current storage capacity. We recommend that you set the Storage Utilization Target between 50% and 60%.
- **Maximum Storage Capacity:** The maximum scaling limit that Amazon MSK can set for your broker storage. You can set the maximum storage capacity up to 16 TiB per broker. For more information, see [Amazon MSK quota](#).

When Amazon MSK detects that your Maximum Disk Utilization metric is equal to or greater than the Storage Utilization Target setting, it increases your storage capacity by an amount equal to the larger of two numbers: 10 GiB or 10% of current storage. For example, if you have 1000 GiB, that amount is 100 GiB. The service checks your storage utilization every minute. Further scaling operations continue to increase storage by an amount equal to the larger of two numbers: 10 GiB or 10% of current storage.

To determine if auto-scaling operations have occurred, use the [ListClusterOperations](#) operation.

Set up automatic scaling for your Amazon MSK cluster

You can use the Amazon MSK console, the Amazon MSK API, or AWS CloudFormation to implement automatic scaling for storage. CloudFormation support is available through [Application Auto Scaling](#).

Note

You can't implement automatic scaling when you create a cluster. You must first create the cluster, and then create and enable an auto-scaling policy for it. However, you can create the policy while Amazon MSK service creates your cluster.

Topics

- [Set up automatic scaling using the Amazon MSK AWS Management Console](#)
- [Set up automatic scaling using the CLI](#)
- [Set up automatic-scaling for Amazon MSK using the API](#)

Set up automatic scaling using the Amazon MSK AWS Management Console

This process describes how to use the Amazon MSK console to implement automatic scaling for storage.

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. In the list of clusters, choose your cluster. This takes you to a page that lists details about the cluster.
3. In the **Auto scaling for storage** section, choose **Configure**.
4. Create and name an auto-scaling policy. Specify the storage utilization target, the maximum storage capacity, and the target metric.
5. Choose **Save changes**.

When you save and enable the new policy, the policy becomes active for the cluster. Amazon MSK then expands the cluster's storage when the storage utilization target is reached.

Set up automatic scaling using the CLI

This process describes how to use the Amazon MSK CLI to implement automatic scaling for storage.

1. Use the [RegisterScalableTarget](#) command to register a storage utilization target.
2. Use the [PutScalingPolicy](#) command to create an auto-expansion policy.

Set up automatic-scaling for Amazon MSK using the API

This process describes how to use the Amazon MSK API to implement automatic scaling for storage.

1. Use the [RegisterScalableTarget](#) API to register a storage utilization target.
2. Use the [PutScalingPolicy](#) API to create an auto-expansion policy.

Manual scaling for Standard brokers

To increase storage, wait for the cluster to be in the ACTIVE state. Storage scaling has a cool-down period of at least six hours between events. Even though the operation makes additional storage available right away, the service performs optimizations on your cluster that can take up to 24 hours or more. The duration of these optimizations is proportional to your storage size.

Scaling up broker storage using the AWS Management Console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. Choose the MSK cluster for which you want to update broker storage.
3. In the **Storage** section, choose **Edit**.
4. Specify the storage volume you want. You can only increase the amount of storage, you can't decrease it.
5. Choose **Save changes**.

Scaling up broker storage using the AWS CLI

Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see [the section called "List clusters"](#).

Replace *Current-Cluster-Version* with the current version of the cluster.

Important

Cluster versions aren't simple integers. To find the current version of the cluster, use the [DescribeCluster](#) operation or the [describe-cluster](#) AWS CLI command. An example version is KTVPDKIKX0DER.

The *Target-Volume-in-GiB* parameter represents the amount of storage that you want each broker to have. It is only possible to update the storage for all the brokers. You can't specify individual brokers for which to update storage. The value you specify for *Target-Volume-in-GiB* must be a whole number that is greater than 100 GiB. The storage per broker after the update operation can't exceed 16384 GiB.

```
aws kafka update-broker-storage --cluster-arn ClusterArn --current-version Current-Cluster-Version --target-broker-ebs-volume-info '{"KafkaBrokerNodeId": "All", "VolumeSizeGB": Target-Volume-in-GiB'
```

Scaling up broker storage using the API

To update a broker storage using the API, see [UpdateBrokerStorage](#).

Manage storage throughput for Standard brokers in a Amazon MSK cluster

For information on how to provision throughput using the Amazon MSK console, CLI, and API, see [???](#).

Topics

- [Amazon MSK broker throughput bottlenecks and maximum throughput settings](#)
- [Measure storage throughput of a Amazon MSK cluster](#)
- [Configuration update values for provisioned storage in a Amazon MSK cluster](#)
- [Provision storage throughput for Standard brokers in a Amazon MSK cluster](#)

Amazon MSK broker throughput bottlenecks and maximum throughput settings

There are multiple causes of bottlenecks in broker throughput: volume throughput, Amazon EC2 to Amazon EBS network throughput, and Amazon EC2 egress throughput. You can enable provisioned

storage throughput to adjust volume throughput. However, broker throughput limitations can be caused by Amazon EC2 to Amazon EBS network throughput and Amazon EC2 egress throughput.

Amazon EC2 egress throughput is impacted by the number of consumer groups and consumers per consumer groups. Also, both Amazon EC2 to Amazon EBS network throughput and Amazon EC2 egress throughput are higher for larger broker sizes.

For volume sizes of 10 GiB or larger, you can provision storage throughput of 250 MiB per second or greater. 250 MiB per second is the default. To provision storage throughput, you must choose broker size `kafka.m5.4xlarge` or larger (or `kafka.m7g.2xlarge` or larger), and you can specify maximum throughput as shown in the following table.

broker size	Maximum storage throughput (MiB/second)
<code>kafka.m5.4xlarge</code>	593
<code>kafka.m5.8xlarge</code>	850
<code>kafka.m5.12xlarge</code>	1000
<code>kafka.m5.16xlarge</code>	1000
<code>kafka.m5.24xlarge</code>	1000
<code>kafka.m7g.2xlarge</code>	312.5
<code>kafka.m7g.4xlarge</code>	625
<code>kafka.m7g.8xlarge</code>	1000
<code>kafka.m7g.12xlarge</code>	1000
<code>kafka.m7g.16xlarge</code>	1000

Measure storage throughput of a Amazon MSK cluster

You can use the `VolumeReadBytes` and `VolumeWriteBytes` metrics to measure the average storage throughput of a cluster. The sum of these two metrics gives the average storage throughput in bytes. To get the average storage throughput for a cluster, set these two metrics to SUM and the period to 1 minute, then use the following formula.

$$\text{Average storage throughput in MiB/s} = (\text{Sum}(\text{VolumeReadBytes}) + \text{Sum}(\text{VolumeWriteBytes})) / (60 * 1024 * 1024)$$

For information about the `VolumeReadBytes` and `VolumeWriteBytes` metrics, see [the section called “PER_BROKER Level monitoring”](#).

Configuration update values for provisioned storage in a Amazon MSK cluster

You can update your Amazon MSK configuration either before or after you turn on provisioned throughput. However, you won't see the desired throughput until you perform both actions: update the `num.replica.fetchers` configuration parameter and turn on provisioned throughput.

In the default Amazon MSK configuration, `num.replica.fetchers` has a value of 2. To update your `num.replica.fetchers`, you can use the suggested values from the following table. These values are for guidance purposes. We recommend that you adjust these values based on your use case.

broker size	num.replica.fetchers
kafka.m5.4xlarge	4
kafka.m5.8xlarge	8
kafka.m5.12xlarge	14
kafka.m5.16xlarge	16
kafka.m5.24xlarge	16

Your updated configuration may not take effect for up to 24 hours, and may take longer when a source volume is not fully utilized. However, transitional volume performance at least equals the performance of source storage volumes during the migration period. A fully-utilized 1 TiB volume typically takes about six hours to migrate to an updated configuration.

Provision storage throughput for Standard brokers in a Amazon MSK cluster

Amazon MSK brokers persist data on storage volumes. Storage I/O is consumed when producers write to the cluster, when data is replicated between brokers, and when consumers read data that

isn't in memory. The volume storage throughput is the rate at which data can be written into and read from a storage volume. Provisioned storage throughput is the ability to specify that rate for the brokers in your cluster.

You can specify the provisioned throughput rate in MiB per second for clusters whose brokers are of size `kafka.m5.4xlarge` or larger and if the storage volume is 10 GiB or greater. It is possible to specify provisioned throughput during cluster creation. You can also enable or disable provisioned throughput for a cluster that is in the `ACTIVE` state.

For information about managing throughput, see [???](#).

Topics

- [Provision Amazon MSK cluster storage throughput using the AWS Management Console](#)
- [Provision Amazon MSK cluster storage throughput using the AWS CLI](#)
- [Provision storage throughput when creating a Amazon MSK cluster using the API](#)

Provision Amazon MSK cluster storage throughput using the AWS Management Console

This process shows an example of how you can use the AWS Management Console to create a Amazon MSK cluster with provisioned throughput enabled.

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose **Create cluster**.
3. Choose **Custom create**.
4. Specify a name for the cluster.
5. In the **Storage** section, choose **Enable**.
6. Choose a value for storage throughput per broker.
7. Choose a VPC, zones and subnets, and a security group.
8. Choose **Next**.
9. At the bottom of the **Security** step, choose **Next**.
10. At the bottom of the **Monitoring and tags** step, choose **Next**.
11. Review the cluster settings, then choose **Create cluster**.

Provision Amazon MSK cluster storage throughput using the AWS CLI

This process shows an example of how you can use the AWS CLI to create a cluster with provisioned throughput enabled.

1. Copy the following JSON and paste it into a file. Replace the subnet IDs and security group ID placeholders with values from your account. Name the file `cluster-creation.json` and save it.

```
{
  "Provisioned": {
    "BrokerNodeGroupInfo": {
      "InstanceType": "kafka.m5.4xlarge",
      "ClientSubnets": [
        "Subnet-1-ID",
        "Subnet-2-ID"
      ],
      "SecurityGroups": [
        "Security-Group-ID"
      ],
      "StorageInfo": {
        "EbsStorageInfo": {
          "VolumeSize": 10,
          "ProvisionedThroughput": {
            "Enabled": true,
            "VolumeThroughput": 250
          }
        }
      }
    },
    "EncryptionInfo": {
      "EncryptionInTransit": {
        "InCluster": false,
        "ClientBroker": "PLAINTEXT"
      }
    },
    "KafkaVersion": "2.8.1",
    "NumberOfBrokerNodes": 2
  },
  "ClusterName": "provisioned-throughput-example"
}
```

2. Run the following AWS CLI command from the directory where you saved the JSON file in the previous step.

```
aws kafka create-cluster-v2 --cli-input-json file://cluster-creation.json
```

Provision storage throughput when creating a Amazon MSK cluster using the API

To configure provisioned storage throughput while creating a cluster, use [CreateClusterV2](#).

Security in Amazon MSK

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is 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. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to Amazon Managed Streaming for Apache Kafka, see [Amazon Web 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 company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon MSK. The following topics show you how to configure Amazon MSK to meet your security and compliance objectives. You also learn how to use other Amazon Web Services that help you to monitor and secure your Amazon MSK resources.

Topics

- [Data protection in Amazon Managed Streaming for Apache Kafka](#)
- [Authentication and authorization for Amazon MSK APIs](#)
- [Authentication and authorization for Apache Kafka APIs](#)
- [Changing an Amazon MSK cluster's security group](#)

- [Control access to Apache ZooKeeper nodes in your Amazon MSK cluster](#)
- [Compliance validation for Amazon Managed Streaming for Apache Kafka](#)
- [Resilience in Amazon Managed Streaming for Apache Kafka](#)
- [Infrastructure security in Amazon Managed Streaming for Apache Kafka](#)

Data protection in Amazon Managed Streaming for Apache Kafka

The AWS [shared responsibility model](#) applies to data protection in Amazon Managed Streaming for Apache Kafka. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see [Working with CloudTrail trails](#) in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon MSK or other AWS services using the console, API, AWS CLI, or AWS

SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Topics

- [Amazon MSK encryption](#)
- [Get started with Amazon MSK encryption](#)
- [Use Amazon MSK APIs with Interface VPC Endpoints](#)

Amazon MSK encryption

Amazon MSK provides data encryption options that you can use to meet strict data management requirements. The certificates that Amazon MSK uses for encryption must be renewed every 13 months. Amazon MSK automatically renews these certificates for all clusters. Express broker clusters remain in ACTIVE state when Amazon MSK starts the certificate-update operation. For standard brokers clusters, Amazon MSK sets the state of the cluster to MAINTENANCE when it starts the certificate-update operation. MSK sets it back to ACTIVE when the update is done. While a cluster is in the certificate-update operation, you can continue to produce and consume data, but you can't perform any update operations on it.

Amazon MSK encryption at rest

Amazon MSK integrates with [AWS Key Management Service](#) (KMS) to offer transparent server-side encryption. Amazon MSK always encrypts your data at rest. When you create an MSK cluster, you can specify the AWS KMS key that you want Amazon MSK to use to encrypt your data at rest. If you don't specify a KMS key, Amazon MSK creates an [AWS managed key](#) for you and uses it on your behalf. For more information about KMS keys, see [AWS KMS keys](#) in the *AWS Key Management Service Developer Guide*.

Amazon MSK encryption in transit

Amazon MSK uses TLS 1.2. By default, it encrypts data in transit between the brokers of your MSK cluster. You can override this default at the time you create the cluster.

For communication between clients and brokers, you must specify one of the following three settings:

- Only allow TLS encrypted data. This is the default setting.

- Allow both plaintext, as well as TLS encrypted data.
- Only allow plaintext data.

Amazon MSK brokers use public AWS Certificate Manager certificates. Therefore, any truststore that trusts Amazon Trust Services also trusts the certificates of Amazon MSK brokers.

While we highly recommend enabling in-transit encryption, it can add additional CPU overhead and a few milliseconds of latency. Most use cases aren't sensitive to these differences, however, and the magnitude of impact depends on the configuration of your cluster, clients, and usage profile.

Get started with Amazon MSK encryption

When creating an MSK cluster, you can specify encryption settings in JSON format. The following is an example.

```
{
  "EncryptionAtRest": {
    "DataVolumeKMSKeyId": "arn:aws:kms:us-east-1:123456789012:key/abcdabcd-1234-
abcd-1234-abcd123e8e8e"
  },
  "EncryptionInTransit": {
    "InCluster": true,
    "ClientBroker": "TLS"
  }
}
```

For `DataVolumeKMSKeyId`, you can specify a [customer managed key](#) or the AWS managed key for MSK in your account (`alias/aws/kafka`). If you don't specify `EncryptionAtRest`, Amazon MSK still encrypts your data at rest under the AWS managed key. To determine which key your cluster is using, send a GET request or invoke the `DescribeCluster` API operation.

For `EncryptionInTransit`, the default value of `InCluster` is `true`, but you can set it to `false` if you don't want Amazon MSK to encrypt your data as it passes between brokers.

To specify the encryption mode for data in transit between clients and brokers, set `ClientBroker` to one of three values: `TLS`, `TLS_PLAINTEXT`, or `PLAINTEXT`.

Topics

- [Specify encryption settings when creating a Amazon MSK cluster](#)
- [Test Amazon MSK TLS encryption](#)

Specify encryption settings when creating a Amazon MSK cluster

This process describes how to specify encryption settings when creating a Amazon MSK cluster.

Specify encryption settings when creating a cluster

1. Save the contents of the previous example in a file and give the file any name that you want. For example, call it `encryption-settings.json`.
2. Run the `create-cluster` command and use the `encryption-info` option to point to the file where you saved your configuration JSON. The following is an example. Replace `{YOUR MSK VERSION}` with a version that matches the Apache Kafka client version. For information on how to find your MSK cluster version, see [To find the version of your MSK cluster](#). Be aware that using an Apache Kafka client version that is not the same as your MSK cluster version may lead to Apache Kafka data corruption, loss and down time.

```
aws kafka create-cluster --cluster-name "ExampleClusterName" --broker-node-group-info file://brokernodegroupinfo.json --encryption-info file://encryptioninfo.json --kafka-version "{YOUR MSK VERSION}" --number-of-broker-nodes 3
```

The following is an example of a successful response after running this command.

```
{
  "ClusterArn": "arn:aws:kafka:us-east-1:123456789012:cluster/SecondTLSTest/abcdabcd-1234-abcd-1234-abcd123e8e8e",
  "ClusterName": "ExampleClusterName",
  "State": "CREATING"
}
```

Test Amazon MSK TLS encryption

This process describes how to test TLS encryption on Amazon MSK.

To test TLS encryption

1. Create a client machine following the guidance in [the section called "Create a client machine"](#).
2. Install Apache Kafka on the client machine.
3. In this example we use the JVM truststore to talk to the MSK cluster. To do this, first create a folder named `/tmp` on the client machine. Then, go to the `bin` folder of the Apache Kafka installation, and run the following command. (Your JVM path might be different.)

```
cp /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.201.b09-0.amzn2.x86_64/jre/lib/security/
cacerts /tmp/kafka.client.truststore.jks
```

4. While still in the `bin` folder of the Apache Kafka installation on the client machine, create a text file named `client.properties` with the following contents.

```
security.protocol=SSL
ssl.truststore.location=/tmp/kafka.client.truststore.jks
```

5. Run the following command on a machine that has the AWS CLI installed, replacing *clusterARN* with the ARN of your cluster.

```
aws kafka get-bootstrap-brokers --cluster-arn clusterARN
```

A successful result looks like the following. Save this result because you need it for the next step.

```
{
  "BootstrapBrokerStringTls": "a-1.example.g7oein.c2.kafka.us-
east-1.amazonaws.com:0123,a-3.example.g7oein.c2.kafka.us-
east-1.amazonaws.com:0123,a-2.example.g7oein.c2.kafka.us-east-1.amazonaws.com:0123"
}
```

6. Run the following command, replacing *BootstrapBrokerStringTls* with one of the broker endpoints that you obtained in the previous step.

```
<path-to-your-kafka-installation>/bin/kafka-console-producer.sh --broker-
list BootstrapBrokerStringTls --producer.config client.properties --topic
TLSTestTopic
```

7. Open a new command window and connect to the same client machine. Then, run the following command to create a console consumer.

```
<path-to-your-kafka-installation>/bin/kafka-console-consumer.sh --bootstrap-
server BootstrapBrokerStringTls --consumer.config client.properties --topic
TLSTestTopic
```

8. In the producer window, type a text message followed by a return, and look for the same message in the consumer window. Amazon MSK encrypted this message in transit.

For more information about configuring Apache Kafka clients to work with encrypted data, see [Configuring Kafka Clients](#).

Use Amazon MSK APIs with Interface VPC Endpoints

You can use an Interface VPC Endpoint, powered by AWS PrivateLink, to prevent traffic between your Amazon VPC and Amazon MSK APIs from leaving the Amazon network. Interface VPC Endpoints don't require an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. [AWS PrivateLink](#) is an AWS technology that enables private communication between AWS services using an elastic network interface with private IPs in your Amazon VPC. For more information, see [Amazon Virtual Private Cloud](#) and [Interface VPC Endpoints \(AWS PrivateLink\)](#).

Your applications can connect with Amazon MSK Provisioned and MSK Connect APIs using AWS PrivateLink. To get started, create an Interface VPC Endpoint for your Amazon MSK API to start traffic flowing from and to your Amazon VPC resources through the Interface VPC Endpoint. FIPS-enabled Interface VPC endpoints are available for US Regions. For more information, see [Create an Interface Endpoint](#).

Using this feature, your Apache Kafka clients can dynamically fetch the connection strings to connect with MSK Provisioned or MSK Connect resources without traversing the internet to retrieve the connection strings.

When creating an Interface VPC Endpoint, choose one of the following service name endpoints:

For MSK Provisioned:

- `com.amazonaws.region.kafka`
- `com.amazonaws.region.kafka-fips` (FIPS-enabled)

Where `region` is your region name. Choose this service name to work with MSK Provisioned-compatible APIs. For more information, see [Operations](#) in the <https://docs.aws.amazon.com/msk/1.0/apireference/>.

For MSK Connect:

- `com.amazonaws.region.kafkaconnect`

Where `region` is your region name. Choose this service name to work with MSK Connect-compatible APIs. For more information, see [Actions](#) in the *Amazon MSK Connect API Reference*.

For more information, including step-by-step instructions to create an interface VPC endpoint, see [Creating an interface endpoint](#) in the *AWS PrivateLink Guide*.

Control access to VPC endpoints for Amazon MSK Provisioned or MSK Connect APIs

VPC endpoint policies let you control access by either attaching a policy to a VPC endpoint or by using additional fields in a policy that is attached to an IAM user, group, or role to restrict access to occur only through the specified VPC endpoint. Use the appropriate example policy to define access permissions for either MSK Provisioned or MSK Connect service.

If you don't attach a policy when you create an endpoint, Amazon VPC attaches a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace IAM identity-based policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *AWS PrivateLink Guide*.

MSK Provisioned — VPC policy example

Read-only access

This sample policy can be attached to a VPC endpoint. (For more information, see *Controlling Access to Amazon VPC Resources*). It restricts actions to only listing and describing operations through the VPC endpoint to which it is attached.

```
{
  "Statement": [
    {
      "Sid": "MSKReadOnly",
      "Principal": "*",
      "Action": [
        "kafka:List*",
        "kafka:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

MSK Provisioned — VPC endpoint policy example

Restrict access to a specific MSK cluster

This sample policy can be attached to a VPC endpoint. It restricts access to a specific Kafka cluster through the VPC endpoint to which it is attached.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificCluster",
      "Principal": "*",
      "Action": "kafka:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kafka:us-east-1:123456789012:cluster/MyCluster"
    }
  ]
}
```

MSK Connect — VPC endpoint policy example

List connectors and create a new connector

The following is an example of an endpoint policy for MSK Connect. This policy allows the specified role to list connectors and create a new connector.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MSKConnectPermissions",
      "Effect": "Allow",
      "Action": [
        "kafkaconnect:ListConnectors",
        "kafkaconnect:CreateConnector"
      ],
      "Resource": "*",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:role/<ExampleRole>"
        ]
      }
    }
  ]
}
```

MSK Connect — VPC endpoint policy example

Allows only requests from a specific IP address in the specified VPC

The following example shows a policy that only allows requests coming from a specified IP address in the specified VPC to succeed. Requests from other IP addresses will fail.

```
{
  "Statement": [
    {
      "Action": "kafkaconnect:*",
      "Effect": "Allow",
      "Principal": "*",
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "aws:VpcSourceIp": "192.0.2.123"
        },
        "StringEquals": {
          "aws:SourceVpc": "vpc-555555555555"
        }
      }
    }
  ]
}
```

Authentication and authorization for Amazon MSK APIs

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 Amazon MSK resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [How Amazon MSK works with IAM](#)
- [Amazon MSK identity-based policy examples](#)
- [Service-linked roles for Amazon MSK](#)
- [AWS managed policies for Amazon MSK](#)
- [Troubleshoot Amazon MSK identity and access](#)

How Amazon MSK works with IAM

Before you use IAM to manage access to Amazon MSK, you should understand what IAM features are available to use with Amazon MSK. To get a high-level view of how Amazon MSK and other AWS services work with IAM, see [AWS Services That Work with IAM](#) in the *IAM User Guide*.

Topics

- [Amazon MSK identity-based policies](#)
- [Amazon MSK resource-based policies](#)
- [Authorization based on Amazon MSK tags](#)
- [Amazon MSK IAM roles](#)

Amazon MSK identity-based policies

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. Amazon MSK supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON Policy Elements Reference](#) in the *IAM User Guide*.

Actions for Amazon MSK identity-based policies

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon MSK use the following prefix before the action: `kafka:`. For example, to grant someone permission to describe an MSK cluster with the Amazon MSK `DescribeCluster` API operation, you include the `kafka:DescribeCluster` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon MSK defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": ["kafka:action1", "kafka:action2"]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "kafka:Describe*"
```

To see a list of Amazon MSK actions, see [Actions, resources, and condition keys for Amazon Managed Streaming for Apache Kafka](#) in the *IAM User Guide*.

Resources for Amazon MSK identity-based policies

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

The Amazon MSK instance resource has the following ARN:

```
arn:${Partition}:kafka:${Region}:${Account}:cluster/${ClusterName}/${UUID}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\) and AWS Service Namespaces](#).

For example, to specify the `CustomerMessages` instance in your statement, use the following ARN:

```
"Resource": "arn:aws:kafka:us-east-1:123456789012:cluster/CustomerMessages/abcd1234-abcd-dcba-4321-a1b2abcd9f9f-2"
```

To specify all instances that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:kafka:us-east-1:123456789012:cluster/*"
```

Some Amazon MSK actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": ["resource1", "resource2"]
```

To see a list of Amazon MSK resource types and their ARNs, see [Resources Defined by Amazon Managed Streaming for Apache Kafka](#) in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by Amazon Managed Streaming for Apache Kafka](#).

Condition keys for Amazon MSK identity-based policies

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Condition element (or Condition *block*) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

Amazon MSK defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see [AWS Global Condition Context Keys](#) in the *IAM User Guide*.

To see a list of Amazon MSK condition keys, see [Condition Keys for Amazon Managed Streaming for Apache Kafka](#) in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see [Actions Defined by Amazon Managed Streaming for Apache Kafka](#).

Examples for Amazon MSK identity-based policies

To view examples of Amazon MSK identity-based policies, see [Amazon MSK identity-based policy examples](#).

Amazon MSK resource-based policies

Amazon MSK supports a cluster policy (also known as a resource-based policy) for use with Amazon MSK clusters. You can use a cluster policy to define which IAM principals have cross-account permissions to set up private connectivity to your Amazon MSK cluster. When used with IAM client authentication, you can also use the cluster policy to granularly define Kafka data plane permissions for the connecting clients.

To view an example of how to configure a cluster policy, refer to [Step 2: Attach a cluster policy to the MSK cluster](#).

Authorization based on Amazon MSK tags

You can attach tags to Amazon MSK clusters. To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `kafka:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For information about tagging Amazon MSK resources, see [the section called "Tag a cluster"](#).

You can only control cluster access with the help of tags. To tag topics and consumer groups, you need to add a separate statement in your policies without tags.

To view example of an identity-based policy for limiting access to a cluster based on the tags on that cluster, see [Accessing Amazon MSK clusters based on tags](#).

You can use conditions in your identity-based policy to control access to Amazon MSK resources based on tags. The following example shows a policy that allows a user to describe the cluster,

get its bootstrap brokers, list its broker nodes, update it, and delete it. However, this policy grants permission only if the cluster tag `Owner` has the value of that user's `username`. The second statement in the following policy allows access to the topics on the cluster. The first statement in this policy doesn't authorize any topic access.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccessClusterIfOwner",
      "Effect": "Allow",
      "Action": [
        "kafka:Describe*",
        "kafka:Get*",
        "kafka:List*",
        "kafka:Update*",
        "kafka:Delete*"
      ],
      "Resource": "arn:aws:kafka:us-east-1:123456789012:cluster/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Owner": "${aws:username}"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:*Topic*",
        "kafka-cluster:WriteData",
        "kafka-cluster:ReadData"
      ],
      "Resource": [
        "arn:aws:kafka:us-east-1:123456789012:topic/*"
      ]
    }
  ]
}
```

Amazon MSK IAM roles

An [IAM role](#) is an entity within your Amazon Web Services account that has specific permissions.

Using temporary credentials with Amazon MSK

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as [AssumeRole](#) or [GetFederationToken](#).

Amazon MSK supports using temporary credentials.

Service-linked roles

[Service-linked roles](#) allow Amazon Web Services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An administrator can view but not edit the permissions for service-linked roles.

Amazon MSK supports service-linked roles. For details about creating or managing Amazon MSK service-linked roles, [the section called “Service-linked roles”](#).

Amazon MSK identity-based policy examples

By default, IAM users and roles don't have permission to execute Amazon MSK API actions. An administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating Policies on the JSON Tab](#) in the *IAM User Guide*.

Topics

- [Policy best practices](#)
- [Allow users to view their own permissions](#)
- [Accessing one Amazon MSK cluster](#)
- [Accessing Amazon MSK clusters based on tags](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon MSK resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [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*.

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": "*"
  }
]
}

```

Accessing one Amazon MSK cluster

In this example, you want to grant an IAM user in your Amazon Web Services account access to one of your clusters, `purchaseQueriesCluster`. This policy allows the user to describe the cluster, get its bootstrap brokers, list its broker nodes, and update it.

```

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

```

```

    "Effect": "Allow",
    "Action": [
        "kafka:Describe*",
        "kafka:Get*",
        "kafka:List*",
        "kafka:Update*"
    ],
    "Resource": "arn:aws:kafka:us-east-1:012345678012:cluster/
purchaseQueriesCluster/abcdefab-1234-abcd-5678-cdef0123ab01-2"
}
]
}

```

Accessing Amazon MSK clusters based on tags

You can use conditions in your identity-based policy to control access to Amazon MSK resources based on tags. This example shows how you might create a policy that allows the user to describe the cluster, get its bootstrap brokers, list its broker nodes, update it, and delete it. However, permission is granted only if the cluster tag `Owner` has the value of that user's user name.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccessClusterIfOwner",
      "Effect": "Allow",
      "Action": [
        "kafka:Describe*",
        "kafka:Get*",
        "kafka:List*",
        "kafka:Update*",
        "kafka:Delete*"
      ],
      "Resource": "arn:aws:kafka:us-east-1:012345678012:cluster/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Owner": "${aws:username}"
        }
      }
    }
  ]
}

```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to update an MSK cluster, the cluster must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise, he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see [IAM JSON Policy Elements: Condition](#) in the *IAM User Guide*.

Service-linked roles for Amazon MSK

Amazon MSK uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon MSK. Service-linked roles are predefined by Amazon MSK and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon MSK easier because you do not have to manually add the necessary permissions. Amazon MSK defines the permissions of its service-linked roles. Unless defined otherwise, only Amazon MSK can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see [Amazon Web Services That Work with IAM](#), and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Topics

- [Service-linked role permissions for Amazon MSK](#)
- [Create a service-linked role for Amazon MSK](#)
- [Edit a service-linked role for Amazon MSK](#)
- [Supported Regions for Amazon MSK service-linked roles](#)

Service-linked role permissions for Amazon MSK

Amazon MSK uses the service-linked role named **AWSServiceRoleForKafka**. Amazon MSK uses this role to access your resources and perform operations such as:

- `*NetworkInterface` – create and manage network interfaces in the customer account that make cluster brokers accessible to clients in the customer VPC.

- `*VpcEndpoints` – manage VPC endpoints in the customer account that make cluster brokers accessible to clients in the customer VPC using AWS PrivateLink. Amazon MSK uses permissions to `DescribeVpcEndpoints`, `ModifyVpcEndpoint` and `DeleteVpcEndpoints`.
- `secretsmanager` – manage client credentials with AWS Secrets Manager.
- `GetCertificateAuthorityCertificate` – retrieve the certificate for your private certificate authority.

This service-linked role is attached to the following managed policy: `KafkaServiceRolePolicy`. For updates to this policy, see [KafkaServiceRolePolicy](#).

The `AWSServiceRoleForKafka` service-linked role trusts the following services to assume the role:

- `kafka.amazonaws.com`

The role permissions policy allows Amazon MSK to complete the following actions on resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:AttachNetworkInterface",
        "ec2>DeleteNetworkInterface",
        "ec2:DetachNetworkInterface",
        "ec2:DescribeVpcEndpoints",
        "acm-pca:GetCertificateAuthorityCertificate",
        "secretsmanager:ListSecrets"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:ModifyVpcEndpoint"
      ],
      "Resource": "arn:*:ec2:*:*:subnet/*"
    }
  ]
}
```

```

    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DeleteVpcEndpoints",
        "ec2:ModifyVpcEndpoint"
      ],
      "Resource": "arn:*:ec2:*:*:vpc-endpoint/*",
      "Condition": {
        "StringEquals": {
          "ec2:ResourceTag/AWSMSKManaged": "true"
        },
        "StringLike": {
          "ec2:ResourceTag/ClusterArn": "*"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetResourcePolicy",
        "secretsmanager:PutResourcePolicy",
        "secretsmanager>DeleteResourcePolicy",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "*",
      "Condition": {
        "ArnLike": {
          "secretsmanager:SecretId": "arn:*:secretsmanager:*:*:secret:AmazonMSK_*"
        }
      }
    }
  ]
}

```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Create a service-linked role for Amazon MSK

You don't need to create a service-linked role manually. When you create an Amazon MSK cluster in the AWS Management Console, the AWS CLI, or the AWS API, Amazon MSK creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create an Amazon MSK cluster, Amazon MSK creates the service-linked role for you again.

Edit a service-linked role for Amazon MSK

Amazon MSK does not allow you to edit the `AWSServiceRoleForKafka` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Supported Regions for Amazon MSK service-linked roles

Amazon MSK supports using service-linked roles in all of the Regions where the service is available. For more information, see [AWS Regions and Endpoints](#).

AWS managed policies for Amazon MSK

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining [customer managed policies](#) that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see [AWS managed policies](#) in the *IAM User Guide*.

AWS managed policy: AmazonMSKFullAccess

This policy grants administrative permissions that allow a principal full access to all Amazon MSK actions. The permissions in this policy are grouped as follows:

- The Amazon MSK permissions allow all Amazon MSK actions.
- **Amazon EC2 permissions** – in this policy are required to validate the passed resources in an API request. This is to make sure Amazon MSK is able to successfully use the resources with a cluster. The rest of the Amazon EC2 permissions in this policy allow Amazon MSK to create AWS resources that are needed to make it possible for you to connect to your clusters.
- **AWS KMS permissions** – are used during API calls to validate the passed resources in a request. They are required for Amazon MSK to be able to use the passed key with the Amazon MSK cluster.
- **CloudWatch Logs, Amazon S3, and Amazon Data Firehose permissions** – are required for Amazon MSK to be able to ensure that the log delivery destinations are reachable, and that they are valid for broker log use.
- **IAM permissions** – are required for Amazon MSK to be able to create a service-linked role in your account and to allow you to pass a service execution role to Amazon MSK.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kafka:*",
      "ec2:DescribeSubnets",
      "ec2:DescribeVpcs",
      "ec2:DescribeSecurityGroups",
      "ec2:DescribeRouteTables",
      "ec2:DescribeVpcEndpoints",
      "ec2:DescribeVpcAttribute",
      "kms:DescribeKey",
      "kms:CreateGrant",
      "logs:CreateLogDelivery",
      "logs:GetLogDelivery",
      "logs:UpdateLogDelivery",
      "logs>DeleteLogDelivery",
      "logs:ListLogDeliveries",
      "logs:PutResourcePolicy",
```

```

    "logs:DescribeResourcePolicies",
    "logs:DescribeLogGroups",
    "S3:GetBucketPolicy",
    "firehose:TagDeliveryStream"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:CreateVpcEndpoint"
  ],
  "Resource": [
    "arn:*:ec2:*:*:vpc/*",
    "arn:*:ec2:*:*:subnet/*",
    "arn:*:ec2:*:*:security-group/*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:CreateVpcEndpoint"
  ],
  "Resource": [
    "arn:*:ec2:*:*:vpc-endpoint/*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/AWSMSKManaged": "true"
    },
    "StringLike": {
      "aws:RequestTag/ClusterArn": "*"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:CreateTags"
  ],
  "Resource": "arn:*:ec2:*:*:vpc-endpoint/*",
  "Condition": {
    "StringEquals": {
      "ec2:CreateAction": "CreateVpcEndpoint"
    }
  }
}

```

```

    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "ec2:DeleteVpcEndpoints"
    ],
    "Resource": "arn:*:ec2:*:*:vpc-endpoint/*",
    "Condition": {
      "StringEquals": {
        "ec2:ResourceTag/AWSMSKManaged": "true"
      },
      "StringLike": {
        "ec2:ResourceTag/ClusterArn": "*"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "iam:PassedToService": "kafka.amazonaws.com"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/kafka.amazonaws.com/AWSServiceRoleForKafka*",
    "Condition": {
      "StringLike": {
        "iam:AWSServiceName": "kafka.amazonaws.com"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:AttachRolePolicy",
      "iam:PutRolePolicy"
    ]
  }
}

```

```

    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/kafka.amazonaws.com/
AWSServiceRoleForKafka*"
  },
  {
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/delivery.logs.amazonaws.com/
AWSServiceRoleForLogDelivery*",
    "Condition": {
      "StringLike": {
        "iam:AWSServiceName": "delivery.logs.amazonaws.com"
      }
    }
  }
]
}

```

AWS managed policy: AmazonMSKReadOnlyAccess

This policy grants read-only permissions that allow users to view information in Amazon MSK. Principals with this policy attached can't make any updates or delete existing resources, nor can they create new Amazon MSK resources. For example, principals with these permissions can view the list of clusters and configurations associated with their account, but cannot change the configuration or settings of any clusters. The permissions in this policy are grouped as follows:

- **Amazon MSK permissions** – allow you to list Amazon MSK resources, describe them, and get information about them.
- **Amazon EC2 permissions** – are used to describe the Amazon VPC, subnets, security groups, and ENIs that are associated with a cluster.
- **AWS KMS permission** – is used to describe the key that is associated with the cluster.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "kafka:Describe*",
        "kafka:List*",

```

```

        "kafka:Get*",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "kms:DescribeKey"
    ],
    "Effect": "Allow",
    "Resource": "*"
}
]
}

```

AWS managed policy: `KafkaServiceRolePolicy`

You can't attach `KafkaServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Amazon MSK to perform actions such as managing VPC endpoints (connectors) on MSK clusters, managing network interfaces, and managing cluster credentials with AWS Secrets Manager. For more information, see [the section called "Service-linked roles"](#).

AWS managed policy: `AWSMSKReplicatorExecutionRole`

The `AWSMSKReplicatorExecutionRole` policy grants permissions to the Amazon MSK replicator to replicate data between MSK clusters. The permissions in this policy are grouped as follows:

- **cluster** – Grants the Amazon MSK Replicator permissions to connect to the cluster using IAM authentication. Also grants permissions to describe and alter the cluster.
- **topic** – Grants the Amazon MSK Replicator permissions to describe, create, and alter a topic, and to alter the topic's dynamic configuration.
- **consumer group** – Grants the Amazon MSK Replicator permissions to describe and alter consumer groups, to read and write data from an MSK cluster, and to delete internal topics created by the replicator.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ClusterPermissions",
      "Effect": "Allow",
      "Action": [

```

```

    "kafka-cluster:Connect",
    "kafka-cluster:DescribeCluster",
    "kafka-cluster:AlterCluster",
    "kafka-cluster:DescribeTopic",
    "kafka-cluster:CreateTopic",
    "kafka-cluster:AlterTopic",
    "kafka-cluster:WriteData",
    "kafka-cluster:ReadData",
    "kafka-cluster:AlterGroup",
    "kafka-cluster:DescribeGroup",
    "kafka-cluster:DescribeTopicDynamicConfiguration",
    "kafka-cluster:AlterTopicDynamicConfiguration",
    "kafka-cluster:WriteDataIdempotently"
  ],
  "Resource": [
    "arn:aws:kafka:*:*:cluster/*"
  ]
},
{
  "Sid": "TopicPermissions",
  "Effect": "Allow",
  "Action": [
    "kafka-cluster:DescribeTopic",
    "kafka-cluster:CreateTopic",
    "kafka-cluster:AlterTopic",
    "kafka-cluster:WriteData",
    "kafka-cluster:ReadData",
    "kafka-cluster:DescribeTopicDynamicConfiguration",
    "kafka-cluster:AlterTopicDynamicConfiguration",
    "kafka-cluster:AlterCluster"
  ],
  "Resource": [
    "arn:aws:kafka:*:*:topic/*/*"
  ]
},
{
  "Sid": "GroupPermissions",
  "Effect": "Allow",
  "Action": [
    "kafka-cluster:AlterGroup",
    "kafka-cluster:DescribeGroup"
  ],
  "Resource": [
    "arn:aws:kafka:*:*:group/*/*"
  ]
}

```

```

    ]
  }
]
}

```

Amazon MSK updates to AWS managed policies

View details about updates to AWS managed policies for Amazon MSK since this service began tracking these changes.

Change	Description	Date
WriteDataIdempotently permission added to AWSMSKReplicatorExecutionRole – Update to an existing policy	Amazon MSK added WriteDataIdempotently permission to AWSMSKReplicatorExecutionRole policy to support data replication between MSK clusters.	March 12, 2024
AWSMSKReplicatorExecutionRole – New policy	Amazon MSK added AWSMSKReplicatorExecutionRole policy to support Amazon MSK Replicator.	December 4, 2023
AmazonMSKFullAccess – Update to an existing policy	Amazon MSK added permissions to support Amazon MSK Replicator.	September 28, 2023
KafkaServiceRolePolicy – Update to an existing policy	Amazon MSK added permissions to support multi-VPC private connectivity.	March 8, 2023
AmazonMSKFullAccess – Update to an existing policy	Amazon MSK added new Amazon EC2 permissions to make it possible to connect to a cluster.	November 30, 2021
AmazonMSKFullAccess – Update to an existing policy	Amazon MSK added a new permission to allow it to	November 19, 2021

Change	Description	Date
	describe Amazon EC2 route tables.	
Amazon MSK started tracking changes	Amazon MSK started tracking changes for its AWS managed policies.	November 19, 2021

Troubleshoot Amazon MSK identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon MSK and IAM.

Topics

- [I Am not authorized to perform an action in Amazon MSK](#)

I Am not authorized to perform an action in Amazon MSK

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 sign-in credentials.

The following example error occurs when the mateojackson IAM user tries to use the console to delete a cluster but does not have `kafka:DeleteCluster` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
kafka:DeleteCluster on resource: purchaseQueriesCluster
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `purchaseQueriesCluster` resource using the `kafka:DeleteCluster` action.

Authentication and authorization for Apache Kafka APIs

You can use IAM to authenticate clients and to allow or deny Apache Kafka actions. Alternatively, you can use TLS or SASL/SCRAM to authenticate clients, and Apache Kafka ACLs to allow or deny actions.

For information on how to control who can perform [Amazon MSK operations](#) on your cluster, see [the section called “Authentication and authorization for Amazon MSK APIs”](#).

Topics

- [IAM access control](#)
- [Mutual TLS client authentication for Amazon MSK](#)
- [Sign-in credentials authentication with AWS Secrets Manager](#)
- [Apache Kafka ACLs](#)

IAM access control

IAM access control for Amazon MSK enables you to handle both authentication and authorization for your MSK cluster. This eliminates the need to use one mechanism for authentication and another for authorization. For example, when a client tries to write to your cluster, Amazon MSK uses IAM to check whether that client is an authenticated identity and also whether it is authorized to produce to your cluster. IAM access control works for Java and non-Java clients, including Kafka clients written in Python, Go, JavaScript, and .NET. IAM access control for non-Java clients is available for MSK clusters with Kafka version 2.7.1 or above.

Amazon MSK logs access events so you can audit them.

To make IAM access control possible, Amazon MSK makes minor modifications to Apache Kafka source code. These modifications won't cause a noticeable difference in your Apache Kafka experience.

Important

IAM access control doesn't apply to Apache ZooKeeper nodes. For information about how you can control access to those nodes, see [Control access to Apache ZooKeeper nodes in your Amazon MSK cluster](#).

Important

The `allow.everyone.if.no.acl.found` Apache Kafka setting has no effect if your cluster uses IAM access control.

⚠ Important

You can invoke Apache Kafka ACL APIs for an MSK cluster that uses IAM access control. However, Apache Kafka ACLs have no effect on authorization for IAM identities. You must use IAM policies to control access for IAM identities.

How IAM access control for Amazon MSK works

To use IAM access control for Amazon MSK, perform the following steps, which are described in detail in these topics:

- [Create a Amazon MSK cluster that uses IAM access control](#)
- [Configure clients for IAM access control](#)
- [Create authorization policies for the IAM role](#)
- [Get the bootstrap brokers for IAM access control](#)

Create a Amazon MSK cluster that uses IAM access control

This section explains how you can use the AWS Management Console, the API, or the AWS CLI to create a Amazon MSK cluster that uses IAM access control. For information about how to turn on IAM access control for an existing cluster, see [Update security settings of a Amazon MSK cluster](#).

Use the AWS Management Console to create a cluster that uses IAM access control

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. Choose **Create cluster**.
3. Choose **Create cluster with custom settings**.
4. In the **Authentication** section, choose **IAM access control**.
5. Complete the rest of the workflow for creating a cluster.

Use the API or the AWS CLI to create a cluster that uses IAM access control

- To create a cluster with IAM access control enabled, use the [CreateCluster](#) API or the [create-cluster](#) CLI command, and pass the following JSON for the `ClientAuthentication` parameter: `"ClientAuthentication": { "Sasl": { "Iam": { "Enabled": true } } }`.

Configure clients for IAM access control

To enable clients to communicate with an MSK cluster that uses IAM access control, you can use either of these mechanisms:

- Non-Java client configuration using SASL_OAUTHBEARER mechanism
- Java client configuration using SASL_OAUTHBEARER mechanism or AWS_MSK_IAM mechanism

Use the SASL_OAUTHBEARER mechanism to configure IAM

1. Edit your client.properties configuration file using the following Python Kafka client example. Configuration changes are similar in other languages.

```
from kafka import KafkaProducer
from kafka.errors import KafkaError
from kafka.sasl.oauth import AbstractTokenProvider
import socket
import time
from aws_msk_iam_sasl_signer import MSKAuthTokenProvider

class MSKTokenProvider():
    def token(self):
        token, _ = MSKAuthTokenProvider.generate_auth_token('<my AWS Region>')
        return token

tp = MSKTokenProvider()

producer = KafkaProducer(
    bootstrap_servers='<myBootstrapString>',
    security_protocol='SASL_SSL',
    sasl_mechanism='OAUTHBEARER',
    sasl_oauth_token_provider=tp,
    client_id=socket.gethostname(),
)

topic = "<my-topic>"
while True:
    try:
        inp=input(">")
        producer.send(topic, inp.encode())
        producer.flush()
        print("Produced!")
```

```
except Exception:
    print("Failed to send message:", e)

producer.close()
```

2. Download the helper library for your chosen configuration language and follow the instructions in the *Getting started* section of that language library's homepage.
 - JavaScript: <https://github.com/aws/aws-msk-iam-sasl-signer-js#getting-started>
 - Python: <https://github.com/aws/aws-msk-iam-sasl-signer-python#get-started>
 - Go: <https://github.com/aws/aws-msk-iam-sasl-signer-go#getting-started>
 - .NET: <https://github.com/aws/aws-msk-iam-sasl-signer-net#getting-started>
 - JAVA: SASL_OAUTHBEARER support for Java is available through the [aws-msk-iam-auth](#) jar file

Use the MSK custom AWS_MSK_IAM mechanism to configure IAM

1. Add the following to the `client.properties` file. Replace `<PATH_TO_TRUST_STORE_FILE>` with the fully-qualified path to the trust store file on the client.

Note

If you don't want to use a specific certificate, you can remove `ssl.truststore.location=<PATH_TO_TRUST_STORE_FILE>` from your `client.properties` file. When you don't specify a value for `ssl.truststore.location`, the Java process uses the default certificate.

```
ssl.truststore.location=<PATH_TO_TRUST_STORE_FILE>
security.protocol=SASL_SSL
sasl.mechanism=AWS_MSK_IAM
sasl.jaas.config=software.amazon.msk.auth.iam.IAMLoginModule required;
sasl.client.callback.handler.class=software.amazon.msk.auth.iam.IAMClientCallbackHandler
```

To use a named profile that you created for AWS credentials, include `awsProfileName="your profile name";` in your client configuration file. For information about named profiles, see [Named profiles](#) in the AWS CLI documentation.

2. Download the latest stable [aws-msk-iam-auth](#) JAR file, and place it in the class path. If you use Maven, add the following dependency, adjusting the version number as needed:

```
<dependency>
  <groupId>software.amazon.msk</groupId>
  <artifactId>aws-msk-iam-auth</artifactId>
  <version>1.0.0</version>
</dependency>
```

The Amazon MSK client plugin is open-sourced under the Apache 2.0 license.

Create authorization policies for the IAM role

Attach an authorization policy to the IAM role that corresponds to the client. In an authorization policy, you specify which actions to allow or deny for the role. If your client is on an Amazon EC2 instance, associate the authorization policy with the IAM role for that Amazon EC2 instance. Alternatively, you can configure your client to use a named profile, and then you associate the authorization policy with the role for that named profile. [Configure clients for IAM access control](#) describes how to configure a client to use a named profile.

For information about how to create an IAM policy, see [Creating IAM policies](#).

The following is an example authorization policy for a cluster named MyTestCluster. To understand the semantics of the Action and Resource elements, see [Semantics of IAM authorization policy actions and resources](#).

Important

Changes that you make to an IAM policy are reflected in the IAM APIs and the AWS CLI immediately. However, it can take noticeable time for the policy change to take effect. In most cases, policy changes take effect in less than a minute. Network conditions may sometimes increase the delay.

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

```

        "Action": [
            "kafka-cluster:Connect",
            "kafka-cluster:AlterCluster",
            "kafka-cluster:DescribeCluster"
        ],
        "Resource": [
            "arn:aws:kafka:us-east-1:0123456789012:cluster/MyTestCluster/abcd1234-0123-abcd-5678-1234abcd-1"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "kafka-cluster:*Topic*",
            "kafka-cluster:WriteData",
            "kafka-cluster:ReadData"
        ],
        "Resource": [
            "arn:aws:kafka:us-east-1:123456789012:topic/MyTestCluster/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "kafka-cluster:AlterGroup",
            "kafka-cluster:DescribeGroup"
        ],
        "Resource": [
            "arn:aws:kafka:us-east-1:123456789012:group/MyTestCluster/*"
        ]
    }
]
}

```

To learn how to create a policy with action elements that correspond to common Apache Kafka use cases, like producing and consuming data, see [Common use cases for client authorization policy](#).

For Kafka versions 2.8.0 and above, the **WriteDataIdempotently** permission is deprecated ([KIP-679](#)). By default, `enable.idempotence = true` is set. Therefore, for Kafka versions 2.8.0 and above, IAM doesn't offer the same functionality as Kafka ACLs. It isn't possible to `WriteDataIdempotently` to a topic by only providing `WriteData` access to that topic. This doesn't affect the case when `WriteData` is provided to **ALL** topics. In that case,

`WriteDataIdempotently` is allowed. This is due to differences in implementation of IAM logic and how the Kafka ACLs are implemented. Additionally, writing to a topic idempotently also requires access to `transactional-ids`.

To work around this, we recommend using a policy similar to the following policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:Connect",
        "kafka-cluster:AlterCluster",
        "kafka-cluster:DescribeCluster",
        "kafka-cluster:WriteDataIdempotently"
      ],
      "Resource": [
        "arn:aws:kafka:us-east-1:123456789012:cluster/MyTestCluster/abcd1234-0123-abcd-5678-1234abcd-1"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:*Topic*",
        "kafka-cluster:WriteData",
        "kafka-cluster:ReadData"
      ],
      "Resource": [
        "arn:aws:kafka:us-east-1:123456789012:topic/MyTestCluster/abcd1234-0123-abcd-5678-1234abcd-1/TestTopic",
        "arn:aws:kafka:us-east-1:123456789012:transactional-id/MyTestCluster/abcd1234-0123-abcd-5678-1234abcd-1/*"
      ]
    }
  ]
}
```

In this case, `WriteData` allows writes to `TestTopic`, while `WriteDataIdempotently` allows idempotent writes to the cluster. This policy also adds access to the `transactional-id` resources that will be needed.

Because `WriteDataIdempotently` is a cluster level permission, you can't use it at the topic level. If `WriteDataIdempotently` is restricted to the topic level, this policy won't work.

Get the bootstrap brokers for IAM access control

See [Get the bootstrap brokers for an Amazon MSK cluster](#).

Semantics of IAM authorization policy actions and resources

This section explains the semantics of the action and resource elements that you can use in an IAM authorization policy. For an example policy, see [Create authorization policies for the IAM role](#).

Authorization policy actions

The following table lists the actions that you can include in an authorization policy when you use IAM access control for Amazon MSK. When you include in your authorization policy an action from the *Action* column of the table, you must also include the corresponding actions from the *Required actions* column.

Action	Description	Required actions	Required resources	Applicable to serverless clusters
<code>kafka-cluster:Connect</code>	Grants permission to connect and authenticate to the cluster.	None	cluster	Yes
<code>kafka-cluster:DescribeCluster</code>	Grants permission to describe various aspects of the cluster, equivalent to Apache Kafka's DESCRIBE CLUSTER ACL.	<code>kafka-cluster:Connect</code>	cluster	Yes
<code>kafka-cluster:AlterCluster</code>	Grants permission to alter various aspects of	<code>kafka-cluster:Connect</code>	cluster	No

Action	Description	Required actions	Required resources	Applicable to serverless clusters
	the cluster, equivalent to Apache Kafka's ALTER CLUSTER ACL.	kafka-cluster:DescribeCluster		
kafka-cluster:DescribeClusterDynamicConfiguration	Grants permission to describe the dynamic configuration of a cluster, equivalent to Apache Kafka's DESCRIBE_CONFIGS CLUSTER ACL.	kafka-cluster:Connect	cluster	No
kafka-cluster:AlterClusterDynamicConfiguration	Grants permission to alter the dynamic configuration of a cluster, equivalent to Apache Kafka's ALTER_CONFIGS CLUSTER ACL.	kafka-cluster:Connect kafka-cluster:DescribeClusterDynamicConfiguration	cluster	No

Action	Description	Required actions	Required resources	Applicable to serverless clusters
<code>kafka-cluster:WriteDataIdempotently</code>	Grants permission to write data idempotently on a cluster, equivalent to Apache Kafka's IDEMPOTENT_WRITE CLUSTER ACL.	<code>kafka-cluster:Connect</code> <code>kafka-cluster:WriteData</code>	cluster	Yes
<code>kafka-cluster:CreateTopic</code>	Grants permission to create topics on a cluster, equivalent to Apache Kafka's CREATE CLUSTER/TOPIC ACL.	<code>kafka-cluster:Connect</code>	topic	Yes
<code>kafka-cluster:DescribeTopic</code>	Grants permission to describe topics on a cluster, equivalent to Apache Kafka's DESCRIBE TOPIC ACL.	<code>kafka-cluster:Connect</code>	topic	Yes

Action	Description	Required actions	Required resources	Applicable to serverless clusters
<code>kafka-cluster:AlterTopic</code>	Grants permission to alter topics on a cluster, equivalent to Apache Kafka's ALTER TOPIC ACL.	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeTopic</code>	topic	Yes
<code>kafka-cluster>DeleteTopic</code>	Grants permission to delete topics on a cluster, equivalent to Apache Kafka's DELETE TOPIC ACL.	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeTopic</code>	topic	Yes
<code>kafka-cluster:DescribeTopicDynamicConfiguration</code>	Grants permission to describe the dynamic configuration of topics on a cluster, equivalent to Apache Kafka's DESCRIBE_CONFIGS TOPIC ACL.	<code>kafka-cluster:Connect</code>	topic	Yes

Action	Description	Required actions	Required resources	Applicable to serverless clusters
<code>kafka-cluster:AlterTopicDynamicConfiguration</code>	Grants permission to alter the dynamic configuration of topics on a cluster, equivalent to Apache Kafka's ALTER_CONFIGS TOPIC ACL.	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeTopicDynamicConfiguration</code>	topic	Yes
<code>kafka-cluster:ReadData</code>	Grants permission to read data from topics on a cluster, equivalent to Apache Kafka's READ TOPIC ACL.	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeTopic</code> <code>kafka-cluster:AlterGroup</code>	topic	Yes
<code>kafka-cluster:WriteData</code>	Grants permission to write data to topics on a cluster, equivalent to Apache Kafka's WRITE TOPIC ACL	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeTopic</code>	topic	Yes

Action	Description	Required actions	Required resources	Applicable to serverless clusters
kafka-cluster:DescribeGroup	Grants permission to describe groups on a cluster, equivalent to Apache Kafka's DESCRIBE GROUP ACL.	kafka-cluster:Connect	group	Yes
kafka-cluster:AlterGroup	Grants permission to join groups on a cluster, equivalent to Apache Kafka's READ GROUP ACL.	kafka-cluster:Connect kafka-cluster:DescribeGroup	group	Yes
kafka-cluster>DeleteGroup	Grants permission to delete groups on a cluster, equivalent to Apache Kafka's DELETE GROUP ACL.	kafka-cluster:Connect kafka-cluster:DescribeGroup	group	Yes

Action	Description	Required actions	Required resources	Applicable to serverless clusters
kafka-cluster:DescribeTransactionalId	Grants permission to describe transactional IDs on a cluster, equivalent to Apache Kafka's DESCRIBE_TRANSACTIONAL_ID ACL.	kafka-cluster:Connect	transactional-id	Yes
kafka-cluster:AlterTransactionalId	Grants permission to alter transactional IDs on a cluster, equivalent to Apache Kafka's WRITE_TRANSACTIONAL_ID ACL.	kafka-cluster:Connect kafka-cluster:DescribeTransactionalId kafka-cluster:WriteData	transactional-id	Yes

You can use the asterisk (*) wildcard any number of times in an action after the colon. The following are examples.

- kafka-cluster:*Topic stands for kafka-cluster:CreateTopic, kafka-cluster:DescribeTopic, kafka-cluster:AlterTopic, and kafka-cluster>DeleteTopic. It doesn't include kafka-cluster:DescribeTopicDynamicConfiguration or kafka-cluster:AlterTopicDynamicConfiguration.
- kafka-cluster:* stands for all permissions.

Authorization policy resources

The following table shows the four types of resources that you can use in an authorization policy when you use IAM access control for Amazon MSK. You can get the cluster Amazon Resource Name (ARN) from the AWS Management Console or by using the [DescribeCluster](#) API or the [describe-cluster](#) AWS CLI command. You can then use the cluster ARN to construct topic, group, and transactional ID ARNs. To specify a resource in an authorization policy, use that resource's ARN.

Resource	ARN format
Cluster	<code>arn:aws:kafka:<i>region</i>:<i>account-id</i> :cluster/<i>cluster-name</i> /<i>cluster-uuid</i></code>
Topic	<code>arn:aws:kafka:<i>region</i>:<i>account-id</i> :topic/<i>cluster-name</i> /<i>cluster-uuid</i> /<i>topic-name</i></code>
Group	<code>arn:aws:kafka:<i>region</i>:<i>account-id</i> :group/<i>cluster-name</i> /<i>cluster-uuid</i> /<i>group-name</i></code>
Transactional ID	<code>arn:aws:kafka:<i>region</i>:<i>account-id</i> :transactional-id/<i>cluster-name</i> /<i>cluster-uuid</i> /<i>transactional-id</i></code>

You can use the asterisk (*) wildcard any number of times anywhere in the part of the ARN that comes after `:cluster/`, `:topic/`, `:group/`, and `:transactional-id/`. The following are some examples of how you can use the asterisk (*) wildcard to refer to multiple resources:

- `arn:aws:kafka:us-east-1:0123456789012:topic/MyTestCluster/*`: all the topics in any cluster named MyTestCluster, regardless of the cluster's UUID.
- `arn:aws:kafka:us-east-1:0123456789012:topic/MyTestCluster/abcd1234-0123-abcd-5678-1234abcd-1/*_test`: all topics whose name ends with `"_test"` in the cluster whose name is MyTestCluster and whose UUID is abcd1234-0123-abcd-5678-1234abcd-1.
- `arn:aws:kafka:us-east-1:0123456789012:transactional-id/MyTestCluster/*/5555abcd-1111-abcd-1234-abcd1234-1`: all transactions whose transactional ID is 5555abcd-1111-abcd-1234-abcd1234-1, across all incarnations of a cluster named MyTestCluster in your account. This means that if you create a cluster named MyTestCluster, then delete it, and then create another cluster by the same name, you can use this resource ARN to represent the same transactions ID on both clusters. However, the deleted cluster isn't accessible.

Common use cases for client authorization policy

The first column in the following table shows some common use cases. To authorize a client to carry out a given use case, include the required actions for that use case in the client's authorization policy, and set Effect to Allow.

For information about all the actions that are part of IAM access control for Amazon MSK, see [Semantics of IAM authorization policy actions and resources](#).

Note

Actions are denied by default. You must explicitly allow every action that you want to authorize the client to perform.

Use case	Required actions
Admin	<code>kafka-cluster:*</code>
Create a topic	<code>kafka-cluster:Connect</code> <code>kafka-cluster:CreateTopic</code>
Produce data	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeTopic</code> <code>kafka-cluster:WriteData</code>
Consume data	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeTopic</code> <code>kafka-cluster:DescribeGroup</code> <code>kafka-cluster:AlterGroup</code> <code>kafka-cluster:ReadData</code>
Produce data idempotently	<code>kafka-cluster:Connect</code>

Use case	Required actions
	<code>kafka-cluster:DescribeTopic</code> <code>kafka-cluster:WriteData</code> <code>kafka-cluster:WriteDataIdempotently</code>
Produce data transactionally	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeTopic</code> <code>kafka-cluster:WriteData</code> <code>kafka-cluster:DescribeTransactionalId</code> <code>kafka-cluster:AlterTransactionalId</code>
Describe the configuration of a cluster	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeClusterDynamicConfiguration</code>
Update the configuration of a cluster	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeClusterDynamicConfiguration</code> <code>kafka-cluster:AlterClusterDynamicConfiguration</code>
Describe the configuration of a topic	<code>kafka-cluster:Connect</code> <code>kafka-cluster:DescribeTopicDynamicConfiguration</code>

Use case	Required actions
Update the configuration of a topic	kafka-cluster:Connect kafka-cluster:DescribeTopic DynamicConfiguration kafka-cluster:AlterTopicDynamicConfiguration
Alter a topic	kafka-cluster:Connect kafka-cluster:DescribeTopic kafka-cluster:AlterTopic

Mutual TLS client authentication for Amazon MSK

You can enable client authentication with TLS for connections from your applications to your Amazon MSK brokers. To use client authentication, you need an AWS Private CA. The AWS Private CA can be either in the same AWS account as your cluster, or in a different account. For information about AWS Private CAs, see [Creating and Managing a AWS Private CA](#).

Note

TLS authentication is not currently available in the Beijing and Ningxia Regions.

Amazon MSK doesn't support certificate revocation lists (CRLs). To control access to your cluster topics or block compromised certificates, use Apache Kafka ACLs and AWS security groups. For information about using Apache Kafka ACLs, see [the section called "Apache Kafka ACLs"](#).

This topic contains the following sections:

- [Create a Amazon MSK cluster that supports client authentication](#)
- [Set up a client to use authentication](#)
- [Produce and consume messages using authentication](#)

Create a Amazon MSK cluster that supports client authentication

This procedure shows you how to enable client authentication using a AWS Private CA.

Note

We highly recommend using independent AWS Private CA for each MSK cluster when you use mutual TLS to control access. Doing so will ensure that TLS certificates signed by PCAs only authenticate with a single MSK cluster.

1. Create a file named `clientauthinfo.json` with the following contents. Replace *Private-CA-ARN* with the ARN of your PCA.

```
{
  "Tls": {
    "CertificateAuthorityArnList": ["Private-CA-ARN"]
  }
}
```

2. Create a file named `brokernodegroupinfo.json` as described in [the section called “Create a provisioned Amazon MSK cluster using the AWS CLI”](#).
3. Client authentication requires that you also enable encryption in transit between clients and brokers. Create a file named `encryptioninfo.json` with the following contents. Replace *KMS-Key-ARN* with the ARN of your KMS key. You can set `ClientBroker` to `TLS` or `TLS_PLAINTEXT`.

```
{
  "EncryptionAtRest": {
    "DataVolumeKMSKeyId": "KMS-Key-ARN"
  },
  "EncryptionInTransit": {
    "InCluster": true,
    "ClientBroker": "TLS"
  }
}
```

For more information about encryption, see [the section called “Amazon MSK encryption”](#).

4. On a machine where you have the AWS CLI installed, run the following command to create a cluster with authentication and in-transit encryption enabled. Save the cluster ARN provided in the response.

```
aws kafka create-cluster --cluster-name "AuthenticationTest" --broker-node-group-info file://brokernodegroupinfo.json --encryption-info file://encryptioninfo.json --client-authentication file://clientauthinfo.json --kafka-version "{YOUR KAFKA VERSION}" --number-of-broker-nodes 3
```

Set up a client to use authentication

This process describes how to set up an Amazon EC2 instance to use as a client to use authentication.

This process describes how to produce and consume messages using authentication by creating a client machine, creating a topic, and configuring the required security settings.

1. Create an Amazon EC2 instance to use as a client machine. For simplicity, create this instance in the same VPC you used for the cluster. See [the section called "Create a client machine"](#) for an example of how to create such a client machine.
2. Create a topic. For an example, see the instructions under [the section called "Create a topic"](#).
3. On a machine where you have the AWS CLI installed, run the following command to get the bootstrap brokers of the cluster. Replace *Cluster-ARN* with the ARN of your cluster.

```
aws kafka get-bootstrap-brokers --cluster-arn Cluster-ARN
```

Save the string associated with `BootstrapBrokerStringTls` in the response.

4. On your client machine, run the following command to use the JVM trust store to create your client trust store. If your JVM path is different, adjust the command accordingly.

```
cp /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.201.b09-0.amzn2.x86_64/jre/lib/security/cacerts kafka.client.truststore.jks
```

5. On your client machine, run the following command to create a private key for your client. Replace *Distinguished-Name*, *Example-Alias*, *Your-Store-Pass*, and *Your-Key-Pass* with strings of your choice.

```
keytool -genkey -keystore kafka.client.keystore.jks -validity 300 -storepass Your-Store-Pass -keypass Your-Key-Pass -dname "CN=Distinguished-Name" -alias Example-Alias -storetype pkcs12 -keyalg rsa
```

6. On your client machine, run the following command to create a certificate request with the private key you created in the previous step.

```
keytool -keystore kafka.client.keystore.jks -certreq -file client-cert-sign-request -alias Example-Alias -storepass Your-Store-Pass -keypass Your-Key-Pass
```

7. Open the `client-cert-sign-request` file and ensure that it starts with `-----BEGIN CERTIFICATE REQUEST-----` and ends with `-----END CERTIFICATE REQUEST-----`. If it starts with `-----BEGIN NEW CERTIFICATE REQUEST-----`, delete the word `NEW` (and the single space that follows it) from the beginning and the end of the file.
8. On a machine where you have the AWS CLI installed, run the following command to sign your certificate request. Replace *Private-CA-ARN* with the ARN of your PCA. You can change the validity value if you want. Here we use 300 as an example.

```
aws acm-pca issue-certificate --certificate-authority-arn Private-CA-ARN --csr fileb://client-cert-sign-request --signing-algorithm "SHA256WITHRSA" --validity Value=300,Type="DAYS"
```

Save the certificate ARN provided in the response.

Note

To retrieve your client certificate, use the `acm-pca get-certificate` command and specify your certificate ARN. For more information, see [get-certificate](#) in the *AWS CLI Command Reference*.

9. Run the following command to get the certificate that AWS Private CA signed for you. Replace *Certificate-ARN* with the ARN you obtained from the response to the previous command.

```
aws acm-pca get-certificate --certificate-authority-arn Private-CA-ARN --certificate-arn Certificate-ARN
```

10. From the JSON result of running the previous command, copy the strings associated with `Certificate` and `CertificateChain`. Paste these two strings in a new file named `signed-`

certificate-from-acm. Paste the string associated with Certificate first, followed by the string associated with CertificateChain. Replace the `\n` characters with new lines. The following is the structure of the file after you paste the certificate and certificate chain in it.

```
-----BEGIN CERTIFICATE-----  
...  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
...  
-----END CERTIFICATE-----  
-----BEGIN CERTIFICATE-----  
...  
-----END CERTIFICATE-----
```

11. Run the following command on the client machine to add this certificate to your keystore so you can present it when you talk to the MSK brokers.

```
keytool -keystore kafka.client.keystore.jks -import -file signed-certificate-from-acm -alias Example-Alias -storepass Your-Store-Pass -keypass Your-Key-Pass
```

12. Create a file named `client.properties` with the following contents. Adjust the truststore and keystore locations to the paths where you saved `kafka.client.truststore.jks`. Substitute your Kafka client version for the `{YOUR KAFKA VERSION}` placeholders.

```
security.protocol=SSL  
ssl.truststore.location=/tmp/kafka_2.12-{YOUR KAFKA VERSION}/  
kafka.client.truststore.jks  
ssl.keystore.location=/tmp/kafka_2.12-{YOUR KAFKA VERSION}/  
kafka.client.keystore.jks  
ssl.keystore.password=Your-Store-Pass  
ssl.key.password=Your-Key-Pass
```

Produce and consume messages using authentication

This process describes how to produce and consume messages using authentication.

1. Run the following command to create a topic. The file named `client.properties` is the one you created in the previous procedure.

```
<path-to-your-kafka-installation>/bin/kafka-topics.sh --create --bootstrap-server BootstrapBroker-String --replication-factor 3 --partitions 1 --topic ExampleTopic --command-config client.properties
```

2. Run the following command to start a console producer. The file named `client.properties` is the one you created in the previous procedure.

```
<path-to-your-kafka-installation>/bin/kafka-console-producer.sh --bootstrap-server BootstrapBroker-String --topic ExampleTopic --producer.config client.properties
```

3. In a new command window on your client machine, run the following command to start a console consumer.

```
<path-to-your-kafka-installation>/bin/kafka-console-consumer.sh --bootstrap-server BootstrapBroker-String --topic ExampleTopic --consumer.config client.properties
```

4. Type messages in the producer window and watch them appear in the consumer window.

Sign-in credentials authentication with AWS Secrets Manager

You can control access to your Amazon MSK clusters using sign-in credentials that are stored and secured using AWS Secrets Manager. Storing user credentials in Secrets Manager reduces the overhead of cluster authentication such as auditing, updating, and rotating credentials. Secrets Manager also lets you share user credentials across clusters.

After you associate a secret with an MSK cluster, MSK syncs the credential data periodically.

This topic contains the following sections:

- [How sign-in credentials authentication works](#)
- [Set up SASL/SCRAM authentication for an Amazon MSK cluster](#)
- [Working with users](#)
- [Limitations when using SCRAM secrets](#)

How sign-in credentials authentication works

Sign-in credentials authentication for Amazon MSK uses SASL/SCRAM (Simple Authentication and Security Layer/ Salted Challenge Response Mechanism) authentication. To set up sign-in credentials authentication for a cluster, you create a Secret resource in [AWS Secrets Manager](#), and associate sign-in credentials with that secret.

SASL/SCRAM is defined in [RFC 5802](#). SCRAM uses secured hashing algorithms, and does not transmit plaintext sign-in credentials between client and server.

Note

When you set up SASL/SCRAM authentication for your cluster, Amazon MSK turns on TLS encryption for all traffic between clients and brokers.

Set up SASL/SCRAM authentication for an Amazon MSK cluster

To set up a secret in AWS Secrets Manager, follow the [Creating and Retrieving a Secret](#) tutorial in the [AWS Secrets Manager User Guide](#).

Note the following requirements when creating a secret for an Amazon MSK cluster:

- Choose **Other type of secrets (e.g. API key)** for the secret type.
- Your secret name must begin with the prefix **AmazonMSK_**.
- You must either use an existing custom AWS KMS key or create a new custom AWS KMS key for your secret. Secrets Manager uses the default AWS KMS key for a secret by default.

Important

A secret created with the default AWS KMS key cannot be used with an Amazon MSK cluster.

- Your sign-in credential data must be in the following format to enter key-value pairs using the **Plaintext** option.

```
{
  "username": "alice",
  "password": "alice-secret"
```

```
}
```

- Record the ARN (Amazon Resource Name) value for your secret.

⚠ Important

You can't associate a Secrets Manager secret with a cluster that exceeds the limits described in [the section called “ Right-size your cluster: Number of partitions per Standard broker”](#).

- If you use the AWS CLI to create the secret, specify a key ID or ARN for the `kms-key-id` parameter. Don't specify an alias.
- To associate the secret with your cluster, use either the Amazon MSK console, or the [BatchAssociateScramSecret](#) operation.

⚠ Important

When you associate a secret with a cluster, Amazon MSK attaches a resource policy to the secret that allows your cluster to access and read the secret values that you defined. You should not modify this resource policy. Doing so can prevent your cluster from accessing your secret. If you make any changes to the Secrets resource policy and/ or the KMS key used for secret encryption, make sure you re-associate the secrets to your MSK cluster. This will make sure that your cluster can continue accessing your secret.

The following example JSON input for the `BatchAssociateScramSecret` operation associates a secret with a cluster:

```
{
  "clusterArn" : "arn:aws:kafka:us-west-2:0123456789019:cluster/SalesCluster/
abcd1234-abcd-cafe-abab-9876543210ab-4",
  "secretArnList": [
    "arn:aws:secretsmanager:us-west-2:0123456789019:secret:AmazonMSK_MyClusterSecret"
  ]
}
```

Connecting to your cluster with sign-in credentials

After you create a secret and associate it with your cluster, you can connect your client to the cluster. The following example steps demonstrate how to connect a client to a cluster that uses SASL/SCRAM authentication, and how to produce to and consume from an example topic.

1. Run the following command on a machine that has the AWS CLI installed, replacing *clusterARN* with the ARN of your cluster.

```
aws kafka get-bootstrap-brokers --cluster-arn clusterARN
```

2. On your client machine, create a JAAS configuration file that contains the user credentials stored in your secret. For example, for the user **alice**, create a file called `users_jaas.conf` with the following content.

```
KafkaClient {  
    org.apache.kafka.common.security.scram.ScramLoginModule required  
    username="alice"  
    password="alice-secret";  
};
```

3. Use the following command to export your JAAS config file as a `KAFKA_OPTS` environment parameter.

```
export KAFKA_OPTS=-Djava.security.auth.login.config=<path-to-jaas-file>/  
users_jaas.conf
```

4. Create a file named `kafka.client.truststore.jks` in a `./tmp` directory.
5. (Optional) Use the following command to copy the JDK key store file from your JVM `cacerts` folder into the `kafka.client.truststore.jks` file that you created in the previous step. Replace *JDKFolder* with the name of the JDK folder on your instance. For example, your JDK folder might be named `java-1.8.0-openjdk-1.8.0.201.b09-0.amzn2.x86_64`.

```
cp /usr/lib/jvm/JDKFolder/lib/security/cacerts /tmp/kafka.client.truststore.jks
```

6. In the `bin` directory of your Apache Kafka installation, create a client properties file called `client_sasl.properties` with the following contents. This file defines the SASL mechanism and protocol.

```
security.protocol=SASL_SSL
```

```
sasl.mechanism=SCRAM-SHA-512
```

7. To create an example topic, run the following command, replacing *BootstrapServerString* with one of the broker endpoints that you obtained in the previous step.

```
<path-to-your-kafka-installation>/bin/kafka-topics.sh --create --bootstrap-server BootstrapBrokerStringSaslScram --command-config client_sasl.properties --replication-factor 3 --partitions 1 --topic ExampleTopicName
```

8. Retrieve your bootstrap brokers string with the following command. Replace *ClusterArn* with the Amazon Resource Name (ARN) of your cluster:

```
aws kafka get-bootstrap-brokers --cluster-arn ClusterArn
```

From the JSON result of the command, save the value associated with the string named `BootstrapBrokerStringSaslScram`.

9. To produce to the example topic that you created, run the following command on your client machine. Replace *BootstrapBrokerStringSaslScram* with the value that you retrieved in the previous step.

```
<path-to-your-kafka-installation>/bin/kafka-console-producer.sh --broker-list BootstrapBrokerStringSaslScram --topic ExampleTopicName --producer.config client_sasl.properties
```

10. To consume from the topic you created, run the following command on your client machine. Replace *BootstrapBrokerStringSaslScram* with the value that you obtained previously.

```
<path-to-your-kafka-installation>/bin/kafka-console-consumer.sh --bootstrap-server BootstrapBrokerStringSaslScram --topic ExampleTopicName --from-beginning --consumer.config client_sasl.properties
```

Working with users

Creating users: You create users in your secret as key-value pairs. When you use the **Plaintext** option in the Secrets Manager console, you should specify sign-in credential data in the following format.

```
{  
  "username": "alice",
```

```
"password": "alice-secret"
}
```

Revoking user access: To revoke a user's credentials to access a cluster, we recommend that you first remove or enforce an ACL on the cluster, and then disassociate the secret. This is because of the following:

- Removing a user does not close existing connections.
- Changes to your secret take up to 10 minutes to propagate.

For information about using an ACL with Amazon MSK, see [Apache Kafka ACLs](#).

For clusters using ZooKeeper mode, we recommend that you restrict access to your ZooKeeper nodes to prevent users from modifying ACLs. For more information, see [Control access to Apache ZooKeeper nodes in your Amazon MSK cluster](#).

Limitations when using SCRAM secrets

Note the following limitations when using SCRAM secrets:

- Amazon MSK only supports SCRAM-SHA-512 authentication.
- An Amazon MSK cluster can have up to 1000 users.
- You must use an AWS KMS key with your Secret. You cannot use a Secret that uses the default Secrets Manager encryption key with Amazon MSK. For information about creating a KMS key, see [Creating symmetric encryption KMS keys](#).
- You can't use an asymmetric KMS key with Secrets Manager.
- You can associate up to 10 secrets with a cluster at a time using the [BatchAssociateScramSecret](#) operation.
- The name of secrets associated with an Amazon MSK cluster must have the prefix **AmazonMSK_**.
- Secrets associated with an Amazon MSK cluster must be in the same Amazon Web Services account and AWS region as the cluster.

Apache Kafka ACLs

Apache Kafka has a pluggable authorizer and ships with an out-of-box authorizer implementation. Amazon MSK enables this authorizer in the `server.properties` file on the brokers.

Apache Kafka ACLs have the format "Principal P is [Allowed/Denied] Operation O From Host H on any Resource R matching ResourcePattern RP". If RP doesn't match a specific resource R, then R has no associated ACLs, and therefore no one other than super users is allowed to access R. To change this Apache Kafka behavior, you set the property `allow.everyone.if.no.acl.found` to true. Amazon MSK sets it to true by default. This means that with Amazon MSK clusters, if you don't explicitly set ACLs on a resource, all principals can access this resource. If you enable ACLs on a resource, only the authorized principals can access it. If you want to restrict access to a topic and authorize a client using TLS mutual authentication, add ACLs using the Apache Kafka authorizer CLI. For more information about adding, removing, and listing ACLs, see [Kafka Authorization Command Line Interface](#).

Because Amazon MSK configures brokers as super users, they can access all topics. This helps the brokers to replicate messages from the primary partition whether or not the `allow.everyone.if.no.acl.found` property is defined for the cluster's configuration.

To add or remove read and write access to a topic

1. Add your brokers to the ACL table to allow them to read from all topics that have ACLs in place. To grant your brokers read access to a topic, run the following command on a client machine that can communicate with the MSK cluster.

Replace *Distinguished-Name* with the DNS of any of your cluster's bootstrap brokers, then replace the string before the first period in this distinguished name by an asterisk (*). For example, if one of your cluster's bootstrap brokers has the DNS `b-6.mytestcluster.67281x.c4.kafka.us-east-1.amazonaws.com`, replace *Distinguished-Name* in the following command with `*.mytestcluster.67281x.c4.kafka.us-east-1.amazonaws.com`. For information on how to get the bootstrap brokers, see [the section called "Get the bootstrap brokers"](#).

```
<path-to-your-kafka-installation>/bin/kafka-acls.sh --bootstrap-server
BootstrapServerString --add --allow-principal "User:CN=Distinguished-Name" --
operation Read --group=* --topic Topic-Name
```

2. To grant a client application read access to a topic, run the following command on your client machine. If you use mutual TLS authentication, use the same *Distinguished-Name* you used when you created the private key.

```
<path-to-your-kafka-installation>/bin/kafka-acls.sh --bootstrap-server  
BootstrapServerString --add --allow-principal "User:CN=Distinguished-Name" --  
operation Read --group=* --topic Topic-Name
```

To remove read access, you can run the same command, replacing `--add` with `--remove`.

3. To grant write access to a topic, run the following command on your client machine. If you use mutual TLS authentication, use the same *Distinguished-Name* you used when you created the private key.

```
<path-to-your-kafka-installation>/bin/kafka-acls.sh --bootstrap-server  
BootstrapServerString --add --allow-principal "User:CN=Distinguished-Name" --  
operation Write --topic Topic-Name
```

To remove write access, you can run the same command, replacing `--add` with `--remove`.

Changing an Amazon MSK cluster's security group

This page explains how to change the security group of an existing MSK cluster. You might need to change a cluster's security group in order to provide access to a certain set of users or to limit access to the cluster. For information about security groups, see [Security groups for your VPC](#) in the Amazon VPC user guide.

1. Use the [ListNodes](#) API or the [list-nodes](#) command in the AWS CLI to get a list of the brokers in your cluster. The results of this operation include the IDs of the elastic network interfaces (ENIs) that are associated with the brokers.
2. Sign in to the AWS Management Console and open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
3. Using the dropdown list near the top-right corner of the screen, select the Region in which the cluster is deployed.
4. In the left pane, under **Network & Security**, choose **Network Interfaces**.
5. Select the first ENI that you obtained in the first step. Choose the **Actions** menu at the top of the screen, then choose **Change Security Groups**. Assign the new security group to this ENI. Repeat this step for each of the ENIs that you obtained in the first step.

 **Note**

Changes that you make to a cluster's security group using the Amazon EC2 console aren't reflected in the MSK console under **Network settings**.

6. Configure the new security group's rules to ensure that your clients have access to the brokers. For information about setting security group rules, see [Adding, Removing, and Updating Rules](#) in the Amazon VPC user guide.

 **Important**

If you change the security group that is associated with the brokers of a cluster, and then add new brokers to that cluster, Amazon MSK associates the new brokers with the original security group that was associated with the cluster when the cluster was created. However, for a cluster to work correctly, all of its brokers must be associated with the same security group. Therefore, if you add new brokers after changing the security group, you must follow the previous procedure again and update the ENIs of the new brokers.

Control access to Apache ZooKeeper nodes in your Amazon MSK cluster

For security reasons you can limit access to the Apache ZooKeeper nodes that are part of your Amazon MSK cluster. To limit access to the nodes, you can assign a separate security group to them. You can then decide who gets access to that security group.

 **Important**

This section does not apply for clusters running in KRaft mode. See [the section called "KRaft mode"](#).

This topic contains the following sections:

- [To place your Apache ZooKeeper nodes in a separate security group](#)
- [Using TLS security with Apache ZooKeeper](#)

To place your Apache ZooKeeper nodes in a separate security group

To limit access to Apache ZooKeeper nodes, you can assign a separate security group to them. You can choose who has access to this new security group by setting security group rules.

1. Get the Apache ZooKeeper connection string for your cluster. To learn how, see [the section called "ZooKeeper mode"](#). The connection string contains the DNS names of your Apache ZooKeeper nodes.
2. Use a tool like `host` or `ping` to convert the DNS names you obtained in the previous step to IP addresses. Save these IP addresses because you need them later in this procedure.
3. Sign in to the AWS Management Console and open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
4. In the left pane, under **NETWORK & SECURITY**, choose **Network Interfaces**.
5. In the search field above the table of network interfaces, type the name of your cluster, then type return. This limits the number of network interfaces that appear in the table to those interfaces that are associated with your cluster.
6. Select the check box at the beginning of the row that corresponds to the first network interface in the list.
7. In the details pane at the bottom of the page, look for the **Primary private IPv4 IP**. If this IP address matches one of the IP addresses you obtained in the first step of this procedure, this means that this network interface is assigned to an Apache ZooKeeper node that is part of your cluster. Otherwise, deselect the check box next to this network interface, and select the next network interface in the list. The order in which you select the network interfaces doesn't matter. In the next steps, you will perform the same operations on all network interfaces that are assigned to Apache ZooKeeper nodes, one by one.
8. When you select a network interface that corresponds to an Apache ZooKeeper node, choose the **Actions** menu at the top of the page, then choose **Change Security Groups**. Assign a new security group to this network interface. For information about creating security groups, see [Creating a Security Group](#) in the Amazon VPC documentation.
9. Repeat the previous step to assign the same new security group to all the network interfaces that are associated with the Apache ZooKeeper nodes of your cluster.
10. You can now choose who has access to this new security group. For information about setting security group rules, see [Adding, Removing, and Updating Rules](#) in the Amazon VPC documentation.

Using TLS security with Apache ZooKeeper

You can use TLS security for encryption in transit between your clients and your Apache ZooKeeper nodes. To implement TLS security with your Apache ZooKeeper nodes, do the following:

- Clusters must use Apache Kafka version 2.5.1 or later to use TLS security with Apache ZooKeeper.
- Enable TLS security when you create or configure your cluster. Clusters created with Apache Kafka version 2.5.1 or later with TLS enabled automatically use TLS security with Apache ZooKeeper endpoints. For information about setting up TLS security, see [Get started with Amazon MSK encryption](#).
- Retrieve the TLS Apache ZooKeeper endpoints using the [DescribeCluster](#) operation.
- Create an Apache ZooKeeper configuration file for use with the `kafka-configs.sh` and [kafka-acls.sh](#) tools, or with the ZooKeeper shell. With each tool, you use the `--zk-tls-config-file` parameter to specify your Apache ZooKeeper config.

The following example shows a typical Apache ZooKeeper configuration file:

```
zookeeper.ssl.client.enable=true
zookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty
zookeeper.ssl.keystore.location=kafka.jks
zookeeper.ssl.keystore.password=test1234
zookeeper.ssl.truststore.location=truststore.jks
zookeeper.ssl.truststore.password=test1234
```

- For other commands (such as `kafka-topics`), you must use the `KAFKA_OPTS` environment variable to configure Apache ZooKeeper parameters. The following example shows how to configure the `KAFKA_OPTS` environment variable to pass Apache ZooKeeper parameters into other commands:

```
export KAFKA_OPTS="
-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty
-Dzookeeper.client.secure=true
-Dzookeeper.ssl.trustStore.location=/home/ec2-user/kafka.client.truststore.jks
-Dzookeeper.ssl.trustStore.password=changeit"
```

After you configure the `KAFKA_OPTS` environment variable, you can use CLI commands normally. The following example creates an Apache Kafka topic using the Apache ZooKeeper configuration from the `KAFKA_OPTS` environment variable:

```
<path-to-your-kafka-installation>/bin/kafka-topics.sh --create --  
zookeeper ZooKeeperTLSConnectString --replication-factor 3 --partitions 1 --topic  
AWSKafkaTutorialTopic
```

Note

The names of the parameters you use in your Apache ZooKeeper configuration file and those you use in your KAFKA_OPTS environment variable are not consistent. Pay attention to which names you use with which parameters in your configuration file and KAFKA_OPTS environment variable.

For more information about accessing your Apache ZooKeeper nodes with TLS, see [KIP-515: Enable ZK client to use the new TLS supported authentication](#).

Compliance validation for Amazon Managed Streaming for Apache Kafka

Third-party auditors assess the security and compliance of Amazon Managed Streaming for Apache Kafka as part of AWS compliance programs. These include PCI and HIPAA BAA.

For a list of AWS services in scope of specific compliance programs, see [Amazon Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using Amazon MSK is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.

- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon Managed Streaming for Apache Kafka

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 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](#).

Infrastructure security in Amazon Managed Streaming for Apache Kafka

As a managed service, Amazon Managed Streaming for Apache Kafka 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 Amazon MSK through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Amazon MSK Provisioned configuration

Amazon MSK provides default configurations for brokers, topics, and metadata nodes. You can also create custom configurations and use them to create new MSK clusters or to update existing clusters. An MSK configuration consists of a set of properties and their corresponding values.

Depending on the broker type you use in your cluster, there are a different set of configuration defaults and a different set of configurations you can modify. See the sections below for more details on how to configure your Standard and Express brokers.

Topics

- [Standard broker configurations](#)
- [Express broker configurations](#)
- [Broker configuration operations](#)

Standard broker configurations

This section describes configuration properties for Standard brokers.

Topics

- [Custom Amazon MSK configurations](#)
- [Default Amazon MSK configuration](#)
- [Guidelines for Amazon MSK tiered storage topic-level configuration](#)

Custom Amazon MSK configurations

You can use Amazon MSK to create a custom MSK configuration where you set the following properties. Properties that you don't set explicitly get the values they have in [the section called "Default Amazon MSK configuration"](#). For more information about configuration properties, see [Apache Kafka Configuration](#).

Apache Kafka configuration properties

Name	Description
<code>allow.everyone.if.no.acl.found</code>	If you want to set this property to <code>false</code> , first make sure you define Apache Kafka ACLs for your cluster. If you set this property to <code>false</code> and you don't first define Apache Kafka ACLs, you lose access to the cluster. If that happens, you can update the configuration again and set this property to <code>true</code> to regain access to the cluster.

Name	Description
auto.create.topics.enable	Enables topic auto-creation on the server.
compression.type	The final compression type for a given topic. You can set this property to the standard compression codecs (gzip, snappy, lz4, and zstd). It additionally accepts uncompressed. This value is equivalent to no compression. If you set the value to producer, it means retain the original compression codec that the producer sets.
connections.max.idle.ms	Idle connections timeout in milliseconds. The server socket processor threads close the connections that are idle for more than the value that you set for this property.
default.replication.factor	The default replication factor for automatically created topics.
delete.topic.enable	Enables the delete topic operation. If you turn off this setting, you can't delete a topic through the admin tool.
group.initial.rebalance.delay.ms	Amount of time the group coordinator waits for more data consumers to join a new group before the group coordinator performs the first rebalance. A longer delay means potentially fewer rebalances, but this increases the time until processing begins.
group.max.session.timeout.ms	Maximum session timeout for registered consumers. Longer timeouts give consumers more time to process messages between heartbeats at the cost of a longer time to detect failures.

Name	Description
<code>group.min.session.timeout.ms</code>	Minimum session timeout for registered consumers. Shorter timeouts result in quicker failure detection at the cost of more frequent consumer heartbeats. This can overwhelm broker resources.
<code>leader.imbalance.per.broker.percentage</code>	The ratio of leader imbalance allowed per broker. The controller triggers a leader balance if it exceeds this value per broker. This value is specified in percentage.
<code>log.cleaner.delete.retention.ms</code>	Amount of time that you want Apache Kafka to retain deleted records. The minimum value is 0.
<code>log.cleaner.min.cleanable.ratio</code>	This configuration property can have values between 0 and 1. This value determines how frequently the log compactor attempts to clean the log (if log compaction is enabled). By default, Apache Kafka avoids cleaning a log if more than 50% of the log has been compacted. This ratio bounds the maximum space that the log wastes with duplicates (at 50%, this means at most 50% of the log could be duplicates). A higher ratio means fewer, more efficient cleanings, but more wasted space in the log.
<code>log.cleanup.policy</code>	The default cleanup policy for segments beyond the retention window. A comma-separated list of valid policies. Valid policies are <code>delete</code> and <code>compact</code> . For Tiered Storage enabled clusters, valid policy is <code>delete</code> only.
<code>log.flush.interval.messages</code>	Number of messages that accumulate on a log partition before messages are flushed to disk.

Name	Description
<code>log.flush.interval.ms</code>	Maximum time in milliseconds that a message in any topic remains in memory before flushed to disk. If you don't set this value, the value in <code>log.flush.scheduler.interval.ms</code> is used. The minimum value is 0.
<code>log.message.timestamp.difference.max.ms</code>	This configuration is deprecated in Kafka 3.6.0. Two configurations, <code>log.message.timestamp.before.max.ms</code> and <code>log.message.timestamp.after.max.ms</code> , have been added. The maximum time difference between the timestamp when a broker receives a message and the timestamp specified in the message. If <code>log.message.timestamp.type=CreateTime</code> , a message is rejected if the difference in timestamp exceeds this threshold. This configuration is ignored if <code>log.message.timestamp.type=LogAppendTime</code> .
<code>log.message.timestamp.type</code>	Specifies if the timestamp in the message is the message creation time or the log append time. The allowed values are <code>CreateTime</code> and <code>LogAppendTime</code> .
<code>log.retention.bytes</code>	Maximum size of the log before deleting it.
<code>log.retention.hours</code>	Number of hours to keep a log file before deleting it, tertiary to the <code>log.retention.ms</code> property.
<code>log.retention.minutes</code>	Number of minutes to keep a log file before deleting it, secondary to <code>log.retention.ms</code> property. If you don't set this value, the value in <code>log.retention.hours</code> is used.

Name	Description
log.retention.ms	Number of milliseconds to keep a log file before deleting it (in milliseconds). If not set, the value in log.retention.minutes is used.
log.roll.ms	Maximum time before a new log segment is rolled out (in milliseconds). If you don't set this property, the value in log.roll.hours is used. The minimum possible value for this property is 1.
log.segment.bytes	Maximum size of a single log file.
max.incremental.fetch.session.cache.slots	Maximum number of incremental fetch sessions that are maintained.
message.max.bytes	<p>Largest record batch size that Kafka allows. If you increase this value and there are consumers older than 0.10.2, you must also increase the fetch size of the consumers so that they can fetch record batches this large.</p> <p>The latest message format version always groups messages into batches for efficiency. Previous message format versions don't group uncompressed records into batches, and in such a case, this limit only applies to a single record.</p> <p>You can set this value per topic with the topic level max.message.bytes config.</p>

Name	Description
min.insync.replicas	<p>When a producer sets acks to "all" (or "-1"), the value in min.insync.replicas specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful. If this minimum cannot be met, the producer raises an exception (either <code>NotEnoughReplicas</code> or <code>NotEnoughReplicasAfterAppend</code>).</p> <p>You can use values in min.insync.replicas and acks to enforce greater durability guarantees. For example, you might create a topic with a replication factor of 3, set min.insync.replicas to 2, and produce with acks of "all". This ensures that the producer raises an exception if a majority of replicas don't receive a write.</p>
num.io.threads	The number of threads that the server uses for processing requests, which may include disk I/O.
num.network.threads	The number of threads that the server uses to receive requests from the network and send responses to it.
num.partitions	Default number of log partitions per topic.
num.recovery.threads.per.data.dir	The number of threads per data directory to be used to recover logs at startup and to flush them at shutdown.
num.replica.fetchers	The number of fetcher threads used to replicate messages from a source broker. If you increase this value, you can increase the degree of I/O parallelism in the follower broker.

Name	Description
<code>offsets.retention.minutes</code>	After a consumer group loses all its consumers (that is, it becomes empty) its offsets are kept for this retention period before getting discarded. For standalone consumers (that is, those that use manual assignment), offsets expire after the time of the last commit plus this retention period.
<code>offsets.topic.replication.factor</code>	The replication factor for the offsets topic. Set this value higher to ensure availability. Internal topic creation fails until the cluster size meets this replication factor requirement.
<code>replica.fetch.max.bytes</code>	Number of bytes of messages to attempt to fetch for each partition. This is not an absolute maximum. If the first record batch in the first non-empty partition of the fetch is larger than this value, the record batch is returned to ensure progress. The <code>message.max.bytes</code> (broker config) or <code>max.message.bytes</code> (topic config) defines the maximum record batch size that the broker accepts.
<code>replica.fetch.response.max.bytes</code>	The maximum number of bytes expected for the entire fetch response. Records are fetched in batches, and if the first record batch in the first non-empty partition of the fetch is larger than this value, the record batch will still be returned to ensure progress. This isn't an absolute maximum. The <code>message.max.bytes</code> (broker config) or <code>max.message.bytes</code> (topic config) properties specify the maximum record batch size that the broker accepts.

Name	Description
replica.lag.time.max.ms	<p>If a follower hasn't sent any fetch requests or hasn't consumed up to the leader's log end offset for at least this number of milliseconds, the leader removes the follower from the ISR.</p> <p>MinValue: 10000</p> <p>MaxValue = 30000</p>
replica.selector.class	<p>The fully-qualified class name that implements <code>ReplicaSelector</code>. The broker uses this value to find the preferred read replica. If you use Apache Kafka version 2.4.1 or higher, and want to allow consumers to fetch from the closest replica, set this property to <code>org.apache.kafka.common.replica.RackAwareReplicaSelector</code>. For more information, see the section called “Apache Kafka version 2.4.1 (use 2.4.1.1 instead)”.</p>
replica.socket.receive.buffer.bytes	<p>The socket receive buffer for network requests.</p>
socket.receive.buffer.bytes	<p>The <code>SO_RCVBUF</code> buffer of the socket server sockets. The minimum value that you can set for this property is -1. If the value is -1, Amazon MSK uses the OS default.</p>
socket.request.max.bytes	<p>The maximum number of bytes in a socket request.</p>
socket.send.buffer.bytes	<p>The <code>SO_SNDBUF</code> buffer of the socket server sockets. The minimum value that you can set for this property is -1. If the value is -1, Amazon MSK uses the OS default.</p>

Name	Description
<code>transaction.max.timeout.ms</code>	Maximum timeout for transactions. If the requested transaction time of a client exceeds this value, the broker returns an error in <code>InitProducerIdRequest</code> . This prevents a client from too large of a timeout, and this can stall consumers that read from topics included in the transaction.
<code>transaction.state.log.min.isr</code>	Overridden <code>min.insync.replicas</code> configuration for the transaction topic.
<code>transaction.state.log.replication.factor</code>	The replication factor for the transaction topic. Set this property to a higher value to increase availability. Internal topic creation fails until the cluster size meets this replication factor requirement.
<code>transactional.id.expiration.ms</code>	The time in milliseconds that the transaction coordinator waits to receive any transaction status updates for the current transaction before the coordinator expires its transactional ID. This setting also influences producer ID expiration because it causes producer IDs expire when this time elapses after the last write with the given producer ID. Producer IDs might expire sooner if the last write from the producer ID is deleted because of the retention settings for the topic. The minimum value for this property is 1 millisecond.
<code>unclean.leader.election.enable</code>	Indicates if replicas not in the ISR set should serve as leader as a last resort, even though this might result in data loss.

Name	Description
zookeeper.connection.timeout.ms	<p>ZooKeeper mode clusters. Maximum time that the client waits to establish a connection to ZooKeeper. If you don't set this value, the value in <code>zookeeper.session.timeout.ms</code> is used.</p> <p>MinValue = 6000</p> <p>MaxValue (inclusive) = 18000</p> <p>We recommend that you set this value to 10,000 on T3.small to avoid cluster downtime.</p>
zookeeper.session.timeout.ms	<p>ZooKeeper mode clusters. The Apache ZooKeeper session timeout in milliseconds.</p> <p>MinValue = 6000</p> <p>MaxValue (inclusive) = 18000</p>

To learn how you can create a custom MSK configuration, list all configurations, or describe them, see [the section called “Broker configuration operations”](#). To create an MSK cluster with a custom MSK configuration, or to update a cluster with a new custom configuration, see [the section called “Key features and concepts”](#).

When you update your existing MSK cluster with a custom MSK configuration, Amazon MSK does rolling restarts when necessary, and uses best practices to minimize customer downtime. For example, after Amazon MSK restarts each broker, Amazon MSK tries to let the broker catch up on data that the broker might have missed during the configuration update before it moves to the next broker.

Dynamic Amazon MSK configuration

In addition to the configuration properties that Amazon MSK provides, you can dynamically set cluster-level and broker-level configuration properties that don't require a broker restart. You can dynamically set some configuration properties. These are the properties not marked as read-only in the table under [Broker Configs](#) in the Apache Kafka documentation. For information on

dynamic configuration and example commands, see [Updating Broker Configs](#) in the Apache Kafka documentation.

Note

You can set the `advertised.listeners` property, but not the `listeners` property.

Topic-level Amazon MSK configuration

You can use Apache Kafka commands to set or modify topic-level configuration properties for new and existing topics. For more information on topic-level configuration properties and examples on how to set them, see [Topic-Level Configs](#) in the Apache Kafka documentation.

Default Amazon MSK configuration

When you create an MSK cluster and don't specify a custom MSK configuration, Amazon MSK creates and uses a default configuration with the values shown in the following table. For properties that aren't in this table, Amazon MSK uses the defaults associated with your version of Apache Kafka. For a list of these default values, see [Apache Kafka Configuration](#).

Default configuration values

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
<code>allow.everyone.if.no.acl.found</code>	If no resource patterns match a specific resource, the resource has no associated ACLs. In this case, if you set this property to <code>true</code> , all users can access the resource, not just the super users.	<code>true</code>	<code>true</code>

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
auto.create.topics.enable	Enables autocreation of a topic on the server.	false	false
auto.leader.rebalance.enable	Enables auto leader balancing. A background thread checks and initiates leader balance at regular intervals, if necessary.	true	true
default.replication.factor	Default replication factors for automatically created topics.	3 for clusters in 3 Availability Zones, and 2 for clusters in 2 Availability Zones.	3 for clusters in 3 Availability Zones, and 2 for clusters in 2 Availability Zones.

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
local.retention.bytes	<p>The maximum size of local log segments for a partition before it deletes the old segments. If you don't set this value, the value in log.retention.bytes is used. The effective value should always be less than or equal to the log.retention.bytes value. The default value of -2 indicates that there is no limit on local retention. This corresponds to the retention.ms/bytes setting of -1. The properties local.retention.ms and local.retention.bytes are similar to log.retention as they are used to determine how long the log segments should remain in local storage. Existing log.retention.* configurations are retention</p>	-2 for unlimited	-2 for unlimited

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
	configurations for the topic partition. This includes both local and remote storage. Valid values: integers in [-2; +Inf]		

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
local.retention.ms	<p>The number of milliseconds to retain the local log segment before deletion. If you don't set this value, Amazon MSK uses the value in <code>log.retention.ms</code>. The effective value should always be less than or equal to the <code>log.retention.bytes</code> value. The default value of -2 indicates that there is no limit on local retention. This corresponds to the <code>retention.ms/bytes</code> setting of -1. The values <code>local.retention.ms</code> and <code>local.retention.bytes</code> are similar to <code>log.retention</code>. MSK uses this configuration to determine how long the log segments should remain in local storage. Existing <code>log.retention.*</code> configurations are</p>	-2 for unlimited	-2 for unlimited

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
	retention configurations for the topic partition. This includes both local and remote storage. Valid values are integers greater than 0.		

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
log.message.timestamp.difference.max.ms	This configuration is deprecated in Kafka 3.6.0. Two configurations, <code>log.message.timestamp.before.max.ms</code> and <code>log.message.timestamp.after.max.ms</code> , have been added. The maximum difference allowed between the timestamp when a broker receives a message and the timestamp specified in the message. If <code>log.message.timestamp.type=CreateTime</code> , a message will be rejected if the difference in timestamp exceeds this threshold. This configuration is ignored if <code>log.message.timestamp.type=LogAppendTime</code> . The maximum timestamp difference allowed	9223372036854775807	86400000 for Kafka 2.8.2.tiered and Kafka 3.7.x tiered.

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
	should be no greater than log.retention.ms to avoid unnecessarily frequent log rolling.		
log.segment.bytes	The maximum size of a single log file.	1073741824	134217728

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
min.insync.replicas	<p>When a producer sets the value of acks (acknowledgement producer gets from Kafka broker) to "all" (or "-1"), the value in min.insync.replicas specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful. If this value doesn't meet this minimum, the producer raises an exception (either <code>NotEnoughReplicas</code> or <code>NotEnoughReplicasAfterAppend</code>).</p> <p>When you use the values in min.insync.replicas and acks together, you can enforce greater durability guarantees. For example, you might create a topic with a replication factor of 3, set min.insync.replica</p>	2 for clusters in 3 Availability Zones, and 1 for clusters in 2 Availability Zones.	2 for clusters in 3 Availability Zones, and 1 for clusters in 2 Availability Zones.

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
	s to 2, and produce with acks of "all". This ensures that the producer raises an exception if a majority of replicas don't receive a write.		
num.io.threads	Number of threads that the server uses to produce requests, which may include disk I/O.	8	max(8, vCPUs) where vCPUs depends on the instance size of broker
num.network.threads	Number of threads that the server uses to receive requests from the network and send responses to the network.	5	max(5, vCPUs / 2) where vCPUs depends on the instance size of broker
num.partitions	Default number of log partitions per topic.	1	1
num.replica.fetchers	Number of fetcher threads used to replicate messages from a source broker.If you increase this value, you can increase the degree of I/O parallelism in the follower broker.	2	max(2, vCPUs / 4) where vCPUs depends on the instance size of broker

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
remote.log.msk.disable.policy	Used with remote.storage.enable to disable tiered storage. Set this policy to Delete, to indicate that data in tiered storage is deleted when you set remote.storage.enable to false.	N/A	None
remote.log.reader.threads	Remote log reader thread pool size, which is used in scheduling tasks to fetch data from remote storage.	N/A	$\max(10, \text{vCPUs} * 0.67)$ where vCPUs depends on the instance size of broker

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
remote.storage.enable	Enables tiered (remote) storage for a topic if set to true. Disables topic level tiered storage if set to false and remote.log.ms.delete.policy is set to Delete. When you disable tiered storage, you delete data from remote storage. When you disable tiered storage for a topic, you can't enable it again.	false	false
replica.lag.time.max.ms	If a follower hasn't sent any fetch requests or hasn't consumed up to the leader's log end offset for at least this number of milliseconds, the leader removes the follower from the ISR.	30000	30000

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
retention.ms	<p>Mandatory field. Minimum time is 3 days. There is no default because the setting is mandatory.</p> <p>Amazon MSK uses the retention.ms value with local.retention.ms to determine when data moves from local to tiered storage. The local.retention.ms value specifies when to move data from local to tiered storage. The retention.ms value specifies when to remove data from tiered storage (that is, removed from the cluster). Valid values: integers in [-1; +Inf]</p>	Minimum 259,200,000 milliseconds (3 days). -1 for infinite retention.	Minimum 259,200,000 milliseconds (3 days). -1 for infinite retention.
socket.receive.buffer.bytes	The SO_RCVBUF buffer of the socket server sockets. If the value is -1, the OS default is used.	102400	102400

Name	Description	Default value for non-tiered storage cluster	Default value for tiered storage-enabled cluster
socket.request.max.bytes	Maximum number of bytes in a socket request.	104857600	104857600
socket.send.buffer.bytes	The SO_SNDBUF buffer of the socket sever sockets. If the value is -1, the OS default is used.	102400	102400
unclean.leader.election.enable	Indicates if you want replicas not in the ISR set to serve as leader as a last resort, even though this might result in data loss.	true	false
zookeeper.session.timeout.ms	The Apache ZooKeeper session timeout in milliseconds.	18000	18000
zookeeper.set.acl	The set client to use secure ACLs.	false	false

For information on how to specify custom configuration values, see [the section called “Custom Amazon MSK configurations”](#).

Guidelines for Amazon MSK tiered storage topic-level configuration

The following are default settings and limitations when you configure tiered storage at the topic level.

- Amazon MSK doesn't support smaller log segment sizes for topics with tiered storage activated. If you want to create a segment, there is a minimum log segment size of 48 MiB, or a minimum segment roll time of 10 minutes. These values map to the `segment.bytes` and `segment.ms` properties.
- The value of `local.retention.ms/bytes` can't equal or exceed the `retention.ms/bytes`. This is the tiered storage retention setting.
- The default value for `local.retention.ms/bytes` is -2. This means that the `retention.ms` value is used for `local.retention.ms/bytes`. In this case, data remains in both local storage and tiered storage (one copy in each), and they expire together. For this option, a copy of the local data is persisted to the remote storage. In this case, the data read from consume traffic comes from the local storage.
- The default value for `retention.ms` is 7 days. There is no default size limit for `retention.bytes`.
- The minimum value for `retention.ms/bytes` is -1. This means infinite retention.
- The minimum value for `local.retention.ms/bytes` is -2. This means infinite retention for local storage. It matches with the `retention.ms/bytes` setting as -1.
- The topic-level configuration `retention.ms` is mandatory for topics with tiered storage activated. The minimum `retention.ms` is 3 days.

For more information about tiered storage constraints, see [Tiered storage constraints and limitations for Amazon MSK clusters](#).

Express broker configurations

Apache Kafka has hundreds of broker configurations that you can use to tune the performance of your MSK Provisioned cluster. Setting erroneous or sub-optimal values can affect cluster reliability and performance. Express brokers improve the availability and durability of your MSK Provisioned clusters by setting optimal values for critical configurations and protecting them from common misconfiguration. There are three categories of configurations based on read and write access: [read/write \(editable\)](#), [read only](#), and non-read/write configurations. Some configurations still use Apache Kafka's default value for the Apache Kafka version the cluster is running. We mark those as Apache Kafka Default.

Topics

- [Custom MSK Express broker configurations \(Read/Write access\)](#)
- [Express brokers read-only configurations](#)

Custom MSK Express broker configurations (Read/Write access)

You can update read/write broker configurations either by using Amazon MSK's [update configuration feature](#) or using Apache Kafka's AlterConfig API. Apache Kafka broker configurations are either static or dynamic. Static configurations require a broker restart for the configuration to be applied, while dynamic configurations do not need a broker restart. For more information about configuration properties and update modes, see [Updating broker configs](#).

Topics

- [Static configurations on MSK Express brokers](#)
- [Dynamic configurations on Express Brokers](#)
- [Topic-level configurations on Express Brokers](#)

Static configurations on MSK Express brokers

You can use Amazon MSK to create a custom MSK configuration file to set the following static properties. Amazon MSK sets and manages all other properties that you do not set. You can create and update static configuration files from the MSK console or using the [configurations command](#).

Property	Description	Default Value
allow.everyone.if.no.acl.found	If you want to set this property to false, first make sure you define Apache Kafka ACLs for your cluster. If you set this property to false and you don't first define Apache Kafka ACLs, you lose access to the cluster. If that happens, you can update the configuration again and set this property to true to regain access to the cluster.	true
auto.create.topics.enable	Enables autocreation of a topic on the server.	false

Property	Description	Default Value
compression.type	<p>Specify the final compression type for a given topic. This configuration accepts the standard compression codecs: gzip, snappy, lz4, zstd.</p> <p>This configuration additionally accepts uncompressed , which is equivalent to no compression; and producer, which means retain the original compression codec set by the producer.</p>	Apache Kafka Default
connections.max.idle.ms	Idle connections timeout in milliseconds. The server socket processor threads close the connections that are idle for more than the value that you set for this property.	Apache Kafka Default
delete.topic.enable	Enables the delete topic operation. If you turn off this setting, you can't delete a topic through the admin tool.	Apache Kafka Default

Property	Description	Default Value
<code>group.initial.rebalance.delay.ms</code>	Amount of time the group coordinator waits for more data consumers to join a new group before the group coordinator performs the first rebalance. A longer delay means potentially fewer rebalances, but this increases the time until processing begins.	Apache Kafka Default
<code>group.max.session.timeout.ms</code>	Maximum session timeout for registered consumers . Longer timeouts give consumers more time to process messages between heartbeats at the cost of a longer time to detect failures.	Apache Kafka Default
<code>leader.imbalance.per.broker.percentage</code>	The ratio of leader imbalance allowed per broker. The controller triggers a leader balance if it exceeds this value per broker. This value is specified in percentage.	Apache Kafka Default
<code>log.cleanup.policy</code>	The default cleanup policy for segments beyond the retention window. A comma-separated list of valid policies. Valid policies are <code>delete</code> and <code>compact</code> . For tiered storage-enabled clusters, valid policy is <code>delete</code> only.	Apache Kafka Default

Property	Description	Default Value
<code>log.message.timestamp.after.max.ms</code>	<p>The allowable timestamp difference between the message timestamp and the broker's timestamp. The message timestamp can be later than or equal to the broker's timestamp, with the maximum allowable difference determined by the value set in this configuration.</p> <p>If <code>log.message.timestamp.type=CreateTime</code> , the message will be rejected if the difference in timestamps exceeds this specified threshold. This configuration is ignored if <code>log.message.timestamp.type=LogAppendTime</code> .</p>	86400000 (24 * 60 * 60 * 1000 ms, that is, 1 day)

Property	Description	Default Value
<code>log.message.timestamp.before.max.ms</code>	<p>The allowable timestamp difference between the broker's timestamp and the message timestamp. The message timestamp can be earlier than or equal to the broker's timestamp, with the maximum allowable difference determined by the value set in this configuration.</p> <p>If <code>log.message.timestamp.type=CreateTime</code> , the message will be rejected if the difference in timestamps exceeds this specified threshold. This configuration is ignored if <code>log.message.timestamp.type=LogAppendTime</code> .</p>	86400000 (24 * 60 * 60 * 1000 ms, that is, 1 day)
<code>log.message.timestamp.type</code>	Specifies if the timestamp in the message is the message creation time or the log append time. The allowed values are <code>CreateTime</code> and <code>LogAppendTime</code> .	Apache Kafka Default
<code>log.retention.bytes</code>	Maximum size of the log before deleting it.	Apache Kafka Default
<code>log.retention.ms</code>	Number of milliseconds to keep a log file before deleting it.	Apache Kafka Default

Property	Description	Default Value
max.connections.per.ip	The maximum number of connections allowed from each IP address. This can be set to 0 if there are overrides configured using the <code>max.connections.per.ip.overrides</code> property. New connections from the IP address are dropped if the limit is reached.	Apache Kafka Default
max.incremental.fetch.session.cache.slots	Maximum number of incremental fetch sessions that are maintained.	Apache Kafka Default

Property	Description	Default Value
message.max.bytes	<p>Largest record batch size that Kafka allows. If you increase this value and there are consumers older than 0.10.2, you must also increase the fetch size of the consumers so that they can fetch record batches this large.</p> <p>The latest message format version always groups messages into batches for efficiency. Previous message format versions don't group uncompressed records into batches, and in such a case, this limit only applies to a single record. You can set this value per topic with the topic level <code>max.message.bytes</code> config.</p>	Apache Kafka Default
num.partitions	Default number of partitions per topic.	1

Property	Description	Default Value
offsets.retention.minutes	After a consumer group loses all its consumers (that is, it becomes empty) its offsets are kept for this retention period before getting discarded. For standalone consumers (that is, those that use manual assignment), offsets expire after the time of the last commit plus this retention period.	Apache Kafka Default
replica.fetch.max.bytes	Number of bytes of messages to attempt to fetch for each partition. This is not an absolute maximum. If the first record batch in the first non-empty partition of the fetch is larger than this value, the record batch is returned to ensure progress. The message.max.bytes (broker config) or max.message.bytes (topic config) defines the maximum record batch size that the broker accepts.	Apache Kafka Default

Property	Description	Default Value
replica.selector.class	The fully-qualified class name that implements ReplicaSelector. The broker uses this value to find the preferred read replica. If you want to allow consumers to fetch from the closest replica, set this property to <code>org.apache.kafka.common.replica.RackAwareReplicaSelector</code> .	Apache Kafka Default
socket.receive.buffer.bytes	The SO_RCVBUF buffer of the socket server sockets. If the value is -1, the OS default is used.	102400
socket.request.max.bytes	Maximum number of bytes in a socket request.	104857600
socket.send.buffer.bytes	The SO_SNDBUF buffer of the socket server sockets. If the value is -1, the OS default is used.	102400

Property	Description	Default Value
transaction.max.timeout.ms	Maximum timeout for transactions. If the requested transaction time of a client exceeds this value, the broker returns an error in <code>InitProducerIdRequest</code> . This prevents a client from too large of a timeout, and this can stall consumers that read from topics included in the transaction.	Apache Kafka Default
transactional.id.expiration.ms	The time in milliseconds that the transaction coordinator waits to receive any transaction status updates for the current transaction before the coordinator expires its transactional ID. This setting also influences producer ID expiration because it causes producer IDs to expire when this time elapses after the last write with the given producer ID. Producer IDs might expire sooner if the last write from the producer ID is deleted because of the retention settings for the topic. The minimum value for this property is 1 millisecond.	Apache Kafka Default

Dynamic configurations on Express Brokers

You can use Apache Kafka AlterConfig API or the `Kafka-configs.sh` tool to edit the following dynamic configurations. Amazon MSK sets and manages all other properties that you do not set. You can dynamically set cluster-level and broker-level configuration properties that don't require a broker restart.

Property	Description	Default value
<code>advertise.listeners</code>	Listeners to publish for clients to use, if different than the <code>listeners</code> config property. In IaaS environments, this may need to be different from the interface to which the broker binds. If this is not set, the value for <code>listeners</code> will be used. Unlike <code>listeners</code> , it is not valid to advertise the <code>0.0.0.0</code> meta-address. Also unlike <code>listeners</code> , there can	<code>null</code>

Property	Description	Default value
	<p>be duplicate d ports in this property, so that one listener can be configure d to advertise another listener's address. This can be useful in some cases where external load balancers are used.</p> <p>This property is set at a per- broker level.</p>	

Property	Description	Default value
compression.type	The final compression type for a given topic. You can set this property to the standard compression codecs (gzip, snappy, lz4, and zstd). It additionally accepts uncompressed. This value is equivalent to no compression. If you set the value to producer, it means retain the original compression codec that the producer sets.	Apache Kafka Default

Property	Description	Default value
log.cleaner.delete.retention.ms	The amount of time to retain delete tombstone markers for log compacted topics. This setting also gives a bound on the time in which a consumer must complete a read if they begin from offset 0 to ensure that they get a valid snapshot of the final stage. Else, delete tombstone s might be collected before they complete their scan.	86400000 (24 * 60 * 60 * 1000 ms, that is, 1 day), Apache Kafka Default

Property	Description	Default value
log.cleaner.min.compaction.lag.ms	The minimum time a message will remain uncompact ed in the log. This setting is only applicabl e for logs that are being compacted.	0, Apache Kafka Default
log.cleaner.max.compaction.lag.ms	The maximum time a message will remain ineligible for compactio n in the log. This setting is only applicabl e for logs that are being compacted. This configura tion would be bounded in the range of [7 days, Long.Max].	9223372036854775807, Apache Kafka Default

Property	Description	Default value
log.cleanup.policy	The default cleanup policy for segments beyond the retention window. A comma-separated list of valid policies. Valid policies are delete and compact. For tiered storage-enabled clusters, valid policy is delete only.	Apache Kafka Default

Property	Description	Default value
<code>log.message.timestamp.after.max.ms</code>	<p>The allowable difference between the message timestamp and the broker's timestamp. The message timestamp can be later than or equal to the broker's timestamp, with the maximum allowable difference determined by the value set in this configuration. If <code>log.message.timestamp.type=CreateTime</code>, the message will be rejected if the difference in timestamps exceeds this specified threshold. This</p>	86400000 (24 * 60 * 60 * 1000 ms, that is, 1 day)

Property	Description	Default value
	configuration is ignored if <code>log.message.timestamp.type=LogAppendTime</code> .	

Property	Description	Default value
<code>log.message.timestamp.before.max.ms</code>	The allowable timestamp difference between the broker's timestamp and the message timestamp. The message timestamp can be earlier than or equal to the broker's timestamp, with the maximum allowable difference determined by the value set in this configuration. If <code>log.message.timestamp.type=CreateTime</code> , the message will be rejected if the difference in timestamps exceeds this specified threshold. This	86400000 (24 * 60 * 60 * 1000 ms, that is, 1 day)

Property	Description	Default value
	configuration is ignored if <code>log.message.timestamp.type=LogAppendTime</code> .	
<code>log.message.timestamp.type</code>	Specifies if the timestamp in the message is the message creation time or the log append time. The allowed values are <code>CreateTime</code> and <code>LogAppendTime</code> .	Apache Kafka Default
<code>log.retention.bytes</code>	Maximum size of the log before deleting it.	Apache Kafka Default
<code>log.retention.ms</code>	Number of milliseconds to keep a log file before deleting it.	Apache Kafka Default

Property	Description	Default value
max.connection.creation.rate	The maximum connection creation rate allowed in the broker at any time.	Apache Kafka Default
max.connections	The maximum number of connections allowed in the broker at any time. This limit is applied in addition to any per-ip limits configured using <code>max.connections.per.ip</code> .	Apache Kafka Default

Property	Description	Default value
max.connections.per.ip	The maximum number of connections allowed from each ip address. This can be set to 0 if there are overrides configured using max.connections.per.ip.overrides property. New connections from the ip address are dropped if the limit is reached.	Apache Kafka Default

Property	Description	Default value
max.connections.per.ip.overrides	A comma-separated list of per-ip or hostname overrides to the default maximum number of connections. An example value is hostName: 100,127.0.0.1:200	Apache Kafka Default

Property	Description	Default value
message.max.bytes	Largest record batch size that Kafka allows. If you increase this value and there are consumers older than 0.10.2, you must also increase the fetch size of the consumers so that they can fetch record batches this large. The latest message format version always groups messages into batches for efficiency. Previous message format versions don't group uncompressed records into batches, and in such a case, this limit only applies to a	Apache Kafka Default

Property	Description	Default value
	single record. You can set this value per topic with the topic level max.messa ge.bytes config.	

Property	Description	Default value
producer.id.expiration.ms	The time in ms that a topic partition leader will wait before expiring producer IDs. Producer IDs will not expire while a transaction associated to them is still ongoing. Note that producer IDs may expire sooner if the last write from the producer ID is deleted due to the topic's retention settings. Setting this value the same or higher than delivery.timeout.ms can help prevent expiration during retries and protect against message	Apache Kafka Default

Property	Description	Default value
	duplication, but the default should be reasonable for most use cases.	

Topic-level configurations on Express Brokers

You can use Apache Kafka commands to set or modify topic-level configuration properties for new and existing topics. If you can't give any topic-level, configuration, Amazon MSK uses the broker default. As with broker-level configurations, Amazon MSK protects some of the topic-level configuration properties from change. Examples include `replication.factor`, `min.insync.replicas` and `unclean.leader.election.enable`. If you try to create a topic with a replication factor value other than 3, Amazon MSK will create the topic with a replication factor of 3 by default. For more information on topic-level configuration properties and examples on how to set them, see [Topic-Level Configs](#) in the Apache Kafka documentation.

Property	Description
<code>cleanup.policy</code>	This config designates the retention policy to use on log segments. The "delete" policy (which is the default) will discard old segments when their retention time or size limit has been reached. The "compact" policy will enable log compaction, which retains the latest value for each key. It is also possible to specify both policies in a comma-separated list (for example, "delete,compact"). In this case, old segments will be discarded per the retention time and size configuration, while retained segments will be compacted. Compaction on Express brokers is triggered after the data in a partition reaches 256 MB.

Property	Description
<code>compression.type</code>	Specify the final compression type for a given topic. This configuration accepts the standard compression codecs (gzip, snappy, lz4, zstd). It additionally accepts uncompressed which is equivalent to no compression; and producer which means retain the original compression codec set by the producer.
<code>delete.retention.ms</code>	<p>The amount of time to retain delete tombstone markers for log compacted topics. This setting also gives a bound on the time in which a consumer must complete a read if they begin from offset 0 to ensure that they get a valid snapshot of the final stage. Else, delete tombstones might be collected before they complete their scan.</p> <p>The default value for this setting is 86400000 (24 * 60 * 60 * 1000 ms, that is, 1 day), Apache Kafka Default</p>
<code>max.message.bytes</code>	The largest record batch size allowed by Kafka (after compression, if compression is enabled). If this is increased and there are consumers older than 0.10.2, the consumers' fetch size must also be increased so that they can fetch record batches this large. In the latest message format version, records are always grouped into batches for efficiency. In previous message format versions, uncompressed records are not grouped into batches and this limit only applies to a single record in that case. This can be set per topic with the topic level <code>max.message.bytes.config</code> .

Property	Description
<code>message.timestamp.after.max.ms</code>	This configuration sets the allowable timestamp difference between the message timestamp and the broker's timestamp. The message timestamp can be later than or equal to the broker's timestamp, with the maximum allowable difference determined by the value set in this configuration. If <code>message.timestamp.type=CreateTime</code> , the message will be rejected if the difference in timestamps exceeds this specified threshold. This configuration is ignored if <code>message.timestamp.type=LogAppendTime</code> .
<code>message.timestamp.before.max.ms</code>	This configuration sets the allowable timestamp difference between the broker's timestamp and the message timestamp. The message timestamp can be earlier than or equal to the broker's timestamp, with the maximum allowable difference determined by the value set in this configuration. If <code>message.timestamp.type=CreateTime</code> , the message will be rejected if the difference in timestamps exceeds this specified threshold. This configuration is ignored if <code>message.timestamp.type=LogAppendTime</code> .
<code>message.timestamp.type</code>	Define whether the timestamp in the message is message create time or log append time. The value should be either <code>CreateTime</code> or <code>LogAppendTime</code> .

Property	Description
<code>min.compaction.lag.ms</code>	<p>The minimum time a message will remain uncompactd in the log. This setting is only applicable for logs that are being compacted.</p> <p>The default value for this setting is 0, Apache Kafka Default</p>
<code>max.compaction.lag.ms</code>	<p>The maximum time a message will remain ineligible for compaction in the log. This setting is only applicable for logs that are being compacted. This configuration would be bounded in the range of [7 days, Long.Max].</p> <p>The default value for this setting is 9223372036854775807, Apache Kafka Default.</p>
<code>retention.bytes</code>	<p>This configuration controls the maximum size a partition (which consists of log segments) can grow to before we will discard old log segments to free up space if we are using the "delete" retention policy. By default there is no size limit only a time limit. Since this limit is enforced at the partition level, multiply it by the number of partitions to compute the topic retention in bytes. Additionally, <code>retention.bytes</code> configuration operates independently of <code>segment.ms</code> and <code>segment.bytes</code> configurations. Moreover, it triggers the rolling of new segment if the <code>retention.bytes</code> is configured to zero.</p>

Property	Description
<code>retention.ms</code>	This configuration controls the maximum time we will retain a log before we will discard old log segments to free up space if we are using the "delete" retention policy. This represents an SLA on how soon consumers must read their data. If set to -1, no time limit is applied. Additionally, <code>retention.ms</code> configuration operates independently of <code>segment.ms</code> and <code>segment.bytes</code> configurations. Moreover, it triggers the rolling of new segment if the <code>retention.ms</code> condition is satisfied.

Express brokers read-only configurations

Amazon MSK sets the values for these configurations and protects them from change that may affect the availability of your cluster. These values may change depending on the Apache Kafka version running on the cluster, so remember to check the values from your specific cluster. Here are some examples.

Express brokers read-only configurations

Property	Description	Express Broker Value
<code>broker.id</code>	The broker id for this server.	1,2,3...
<code>broker.rack</code>	Rack of the broker. This will be used in rack aware replication assignment for fault tolerance. Examples: <code>`RACK1`</code> , <code>`us-east-1d`</code>	AZ ID or Subnet ID
<code>default.replication.factor</code>	Default replication factors for all topics.	3

Property	Description	Express Broker Value
fetch.max.bytes	The maximum number of bytes we will return for a fetch request.	Apache Kafka Default
group.max.size	The maximum number of consumers that a single consumer group can accommodate.	Apache Kafka Default
inter.broker.listener.name	Name of listener used for communication between brokers.	REPLICATION_SECURE or REPLICATION
inter.broker.protocol.version	Specifies which version of the inter-broker protocol is used.	Apache Kafka Default
listeners	Listener List - Comma-separated list of URIs we will listen on and the listener names. You can set the <code>advertised.listeners</code> property, but not the <code>listeners</code> property.	MSK-generated
log.message.format.version	Specify the message format version the broker will use to append messages to the logs.	Apache Kafka Default

Property	Description	Express Broker Value
<code>min.insync.replicas</code>	<p>When a producer sets <code>acks</code> to <code>all</code> (or <code>-1</code>), the value in <code>min.insync.replicas</code> specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful. If this minimum cannot be met, the producer raises an exception (either <code>NotEnoughReplicas</code> or <code>NotEnoughReplicasAfterAppend</code>).</p> <p>You can use value of <code>acks</code> from your producer to enforce greater durability guarantees. By setting <code>acks</code> to <code>"all"</code>. This ensures that the producer raises an exception if a majority of replicas don't receive a write.</p>	2
<code>num.io.threads</code>	Number of threads that the server uses to produce requests, which may include disk I/O. (m7g.large, 8), (m7g.xlarge, 8), (m7g.2xlarge, 16), (m7g.4xlarge, 32), (m7g.8xlarge, 64), (m7g.12xlarge, 96), (m7g.16xlarge, 128)	Based on instance type. $\text{=Math.max}(8, 2 * \text{vCPUs})$

Property	Description	Express Broker Value
num.network.threads	Number of threads that the server uses to receive requests from the network and send responses to the network. (m7g.large, 8), (m7g.xlarge, 8), (m7g.2xlarge, 8), (m7g.4xlarge, 16), (m7g.8xlarge, 32), (m7g.12xlarge, 48), (m7g.16xlarge, 64)	Based on instance type. =Math.max(8, vCPUs)
replica.fetch.response.max.bytes	The maximum number of bytes expected for the entire fetch response. Records are fetched in batches, and if the first record batch in the first non-empty partition of the fetch is larger than this value, the record batch will still be returned to ensure progress. This isn't an absolute maximum. The <code>message.max.bytes</code> (broker config) or <code>max.message.bytes</code> (topic config) properties specify the maximum record batch size that the broker accepts.	Apache Kafka Default

Property	Description	Express Broker Value
request.timeout.ms	The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses, the client will resend the request if necessary or fail the request if retries are exhausted.	Apache Kafka Default
transaction.state.log.min.isr	Overridden <code>min.insync.replicas</code> configuration for the transaction topic.	2
transaction.state.log.replication.factor	The replication factor for the transaction topic.	Apache Kafka Default
unclean.leader.election.enable	Allows replicas not in the ISR set to serve as leader as a last resort, even though this might result in data loss.	FALSE

Broker configuration operations

Apache Kafka broker configurations are either static or dynamic. Static configurations require a broker restart for the configuration to be applied. Dynamic configurations do not need a broker restart for the configuration to be updated. For more information about configuration properties and update modes, see [Apache Kafka Configuration](#).

This topic describes how to create custom MSK configurations and how to perform operations on them. For information about how to use MSK configurations to create or update clusters, see [the section called “Key features and concepts”](#).

Topics

- [Create a configuration](#)

- [Update configuration](#)
- [Delete configuration](#)
- [Get configuration metadata](#)
- [Get details about configuration revision](#)
- [List configurations in your account for the current Region](#)
- [Amazon MSK configuration states](#)

Create a configuration

This process describes how to create a custom Amazon MSK configuration and how to perform operations on it.

1. Create a file where you specify the configuration properties that you want to set and the values that you want to assign to them. The following are the contents of an example configuration file.

```
auto.create.topics.enable = true  
  
log.roll.ms = 604800000
```

2. Run the following AWS CLI command, and replace *config-file-path* with the path to the file where you saved your configuration in the previous step.

Note

The name that you choose for your configuration must match the following regex:
`"^[0-9A-Za-z][0-9A-Za-z-]{0,}$"`.

```
aws kafka create-configuration --name "ExampleConfigurationName" --description  
"Example configuration description." --kafka-versions "1.1.1" --server-properties  
fileb://config-file-path
```

The following is an example of a successful response after you run this command.

```
{
```

```
"Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/
abcdabcd-1234-abcd-1234-abcd123e8e8e-1",
"CreationTime": "2019-05-21T19:37:40.626Z",
"LatestRevision": {
  "CreationTime": "2019-05-21T19:37:40.626Z",
  "Description": "Example configuration description.",
  "Revision": 1
},
"Name": "ExampleConfigurationName"
}
```

3. The previous command returns an Amazon Resource Name (ARN) for your new configuration. Save this ARN because you need it to refer to this configuration in other commands. If you lose your configuration ARN, you can list all the configurations in your account to find it again.

Update configuration

This process describes how to update a custom Amazon MSK configuration.

1. Create a file where you specify the configuration properties that you want to update and the values that you want to assign to them. The following are the contents of an example configuration file.

```
auto.create.topics.enable = true

min.insync.replicas = 2
```

2. Run the following AWS CLI command, and replace *config-file-path* with the path to the file where you saved your configuration in the previous step.

Replace *configuration-arn* with the ARN that you obtained when you created the configuration. If you didn't save the ARN when you created the configuration, you can use the `list-configurations` command to list all configuration in your account. The configuration that you want in the list appears in the response. The ARN of the configuration also appears in that list.

```
aws kafka update-configuration --arn configuration-arn --description "Example
configuration revision description." --server-properties fileb://config-file-path
```

3. The following is an example of a successful response after you run this command.

```
{
  "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/
abcdabcd-1234-abcd-1234-abcd123e8e8e-1",
  "LatestRevision": {
    "CreationTime": "2020-08-27T19:37:40.626Z",
    "Description": "Example configuration revision description.",
    "Revision": 2
  }
}
```

Delete configuration

The following procedure shows how to delete a configuration that isn't attached to a cluster. You can't delete a configuration that's attached to a cluster.

1. To run this example, replace *configuration-arn* with the ARN that you obtained when you created the configuration. If you didn't save the ARN when you created the configuration, you can use the `list-configurations` command to list all configuration in your account. The configuration that you want in the list appears in the response. The ARN of the configuration also appears in that list.

```
aws kafka delete-configuration --arn configuration-arn
```

2. The following is an example of a successful response after you run this command.

```
{
  "arn": " arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/
abcdabcd-1234-abcd-1234-abcd123e8e8e-1",
  "state": "DELETING"
}
```

Get configuration metadata

The following procedure shows how to describe an Amazon MSK configuration to get metadata about the configuration.

1. The following command returns metadata about the configuration. To get a detailed description of the configuration, run the `describe-configuration-revision`.

To run this example, replace *configuration-arn* with the ARN that you obtained when you created the configuration. If you didn't save the ARN when you created the configuration, you can use the `list-configurations` command to list all configuration in your account. The configuration that you want in the list appears in the response. The ARN of the configuration also appears in that list.

```
aws kafka describe-configuration --arn configuration-arn
```

2. The following is an example of a successful response after you run this command.

```
{
  "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/abcdabcd-
abcd-1234-abcd-abcd123e8e8e-1",
  "CreationTime": "2019-05-21T00:54:23.591Z",
  "Description": "Example configuration description.",
  "KafkaVersions": [
    "1.1.1"
  ],
  "LatestRevision": {
    "CreationTime": "2019-05-21T00:54:23.591Z",
    "Description": "Example configuration description.",
    "Revision": 1
  },
  "Name": "SomeTest"
}
```

Get details about configuration revision

This process gets you a detailed description of the Amazon MSK configuration revision.

If you use the `describe-configuration` command to describe an MSK configuration, you see the metadata of the configuration. To get a description of the configuration, use the command, `describe-configuration-revision`.

- Run the following command and replace *configuration-arn* with the ARN that you obtained when you created the configuration. If you didn't save the ARN when you created the configuration, you can use the `list-configurations` command to list all configuration in your account. The configuration that you want in the list that appears in the response. The ARN of the configuration also appears in that list.

```
aws kafka describe-configuration-revision --arn configuration-arn --revision 1
```

The following is an example of a successful response after you run this command.

```
{
  "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/abcdabcd-
abcd-1234-abcd-abcd123e8e8e-1",
  "CreationTime": "2019-05-21T00:54:23.591Z",
  "Description": "Example configuration description.",
  "Revision": 1,
  "ServerProperties":
    "YXV0by5jcmVhdGUudG9waWNzLmVuYWJsZSA9IHRydWUKCgp6b29rZWVwZXIuY29ubmVjdGlvbi50aW1lb3V0Lm1zI
}
```

The value of `ServerProperties` is encoded with base64. If you use a base64 decoder (for example, <https://www.base64decode.org/>) to decode it manually, you get the contents of the original configuration file that you used to create the custom configuration. In this case, you get the following:

```
auto.create.topics.enable = true

log.roll.ms = 604800000
```

List configurations in your account for the current Region

This process describes how to list all Amazon MSK configurations in your account for the current AWS Region.

- Run the following command.

```
aws kafka list-configurations
```

The following is an example of a successful response after you run this command.

```
{
  "Configurations": [
    {
```

```

        "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/
abcdabcd-abcd-1234-abcd-abcd123e8e8e-1",
        "CreationTime": "2019-05-21T00:54:23.591Z",
        "Description": "Example configuration description.",
        "KafkaVersions": [
            "1.1.1"
        ],
        "LatestRevision": {
            "CreationTime": "2019-05-21T00:54:23.591Z",
            "Description": "Example configuration description.",
            "Revision": 1
        },
        "Name": "SomeTest"
    },
    {
        "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/
abcdabcd-1234-abcd-1234-abcd123e8e8e-1",
        "CreationTime": "2019-05-03T23:08:29.446Z",
        "Description": "Example configuration description.",
        "KafkaVersions": [
            "1.1.1"
        ],
        "LatestRevision": {
            "CreationTime": "2019-05-03T23:08:29.446Z",
            "Description": "Example configuration description.",
            "Revision": 1
        },
        "Name": "ExampleConfigurationName"
    }
]
}

```

Amazon MSK configuration states

An Amazon MSK configuration can be in one of the following states. To perform an operation on a configuration, the configuration must be in the `ACTIVE` or `DELETE_FAILED` state:

- `ACTIVE`
- `DELETING`
- `DELETE_FAILED`

Patching

Patching on MSK Provisioned clusters

Periodically, Amazon MSK updates software on the brokers in your cluster. Maintenance includes planned updates or unplanned repairs. Planned maintenance includes operating system updates, security updates, and other software updates required to maintain the health, security, and performance of your cluster. We perform unplanned maintenance to resolve sudden infrastructure degradation. We perform maintenance on Standard and Express brokers, but the experiences are different.

Patching for Standard brokers

Updates to your Standard brokers have no impact on your applications' writes and reads if you follow [best practices](#).

Amazon MSK uses rolling updates for software to maintain high availability of your clusters. During this process, brokers are rebooted one at a time, and Kafka automatically moves leadership to another online broker. Kafka clients have built-in mechanisms to automatically detect the change in leadership for the partitions and continue to write and read data into a MSK cluster. Follow [Best practices for Apache Kafka clients](#) for smooth operation of your cluster at all times, including during patching.

Following a broker going offline, it is normal to see transient disconnect errors on your clients. You will also observe for a brief window (up to 2 mins, typically less) some spikes in p99 read and write latency (typically high milliseconds, up to ~2 seconds). These spikes are expected and are caused by the client re-reconnecting to a new leader broker; it does not impact your produce or consume and will resolve following the re-connect. For more information, see [Broker offline and client failover](#).

You will also observe an increase in the metric `UnderReplicatedPartitions`, which is expected as the partitions on the broker that was shut down are no longer replicating data. This has no impact on applications' writes and reads as replicas for these partitions that are hosted on other brokers are now serving the requests.

After the software update, when the broker comes back online, it needs to "catch up" on the messages produced while it was offline. During catch up, you may also observe an increase in usage of the volume throughput and CPU. These should have no impact on writes and reads into the cluster if you have enough CPU, memory, network, and volume resources on your brokers.

Patching for Express brokers

There are no maintenance windows for Express brokers. Amazon MSK automatically updates your cluster on an ongoing basis in a time distributed manner, meaning you can expect occasional and singular broker reboots across the month. This ensures you do not need to make any plans or accommodations around one-time cluster-wide maintenance windows. As always, traffic will remain uninterrupted during a broker reboot as leadership will change to other brokers that will continue serving requests.

Express brokers come configured with best practice settings and guardrails that make your cluster resilient to load changes that may occur during maintenance. Amazon MSK sets throughput quotas on your Express brokers to mitigate the impact of overloading your cluster which can lead to issues during broker restarts. These improvements eliminate the need for advance notifications, planning, and maintenance windows when you use Express brokers.

Express brokers always replicate your data three ways so your clients automatically failover during reboots. You don't need to worry about topics becoming unavailable because of replication factor set to 1 or 2. Also, catch up for a restarted Express broker is faster than on Standard brokers. The faster patching speed on Express brokers means that there will be minimal planning disruption to any control plane activities you may have scheduled for your cluster.

As with all Apache Kafka applications, there is still a shared client-server contract for clients connecting to Express brokers. It's still critical to configure your clients to handle leadership failover between brokers. Follow the [Best practices for Apache Kafka clients](#) for a smooth operation of your cluster at all times, including during patching. Following a broker restart, it is normal to see transient [disconnect errors on your clients](#). This will not affect your produce and consume as follower brokers will take over partition leadership. Your Apache Kafka clients will automatically fail-over and start sending requests to the new leader brokers.

Broker offline and client failover

Kafka allows for an offline broker; a single offline broker in a healthy and balanced cluster following best practices will not see impact or cause failure to produce or consume. This is because another broker will take over partition leadership and because the Kafka client lib will automatically fail-over and start sending requests to the new leader brokers.

Client server contract

This results in a shared contract between the client library and server-side behavior; the server must successfully assign one or more new leaders and the client must change brokers to send requests to the new leaders in a timely manner.

Kafka uses exceptions to control this flow:

An example procedure

1. Broker A enters an offline state.
2. Kafka client receives an exception (typically network disconnect or `not_leader_for_partition`).
3. These exceptions trigger the Kafka client to update its metadata so that it knows about the latest leaders.
4. Kafka client resumes sending requests to the new partition leaders on other brokers.

This process typically takes less than 2 seconds with the vended Java client and default configurations. The client side errors are verbose and repetitive but not cause for concern, as denoted by the "WARN" level.

Example: Exception 1

```
10:05:25.306 [kafka-producer-network-thread | producer-1] WARN
o.a.k.c.producer.internals.Sender - [Producer clientId=producer-1] Got
error produce response with correlation id 864845 on topic-partition
msk-test-topic-1-0, retrying (2147483646 attempts left). Error:
NETWORK_EXCEPTION. Error Message: Disconnected from node 2
```

Example: Exception 2

```
10:05:25.306 [kafka-producer-network-thread | producer-1] WARN
o.a.k.c.producer.internals.Sender - [Producer clientId=producer-1] Received
invalid metadata error in produce request on partition msk-test-topic-1-41
due to org.apache.kafka.common.errors.NotLeaderOrFollowerException: For
requests intended only for the leader, this error indicates that the broker
is not the current leader. For requests intended for any replica, this
error indicates that the broker is not a replica of the topic partition..
Going to request metadata update now"
```

Kafka clients will automatically resolve these errors typically within 1 second and at most 3 seconds. This presents as produce/consume latency at p99 in client side metrics (typically high

milliseconds in the 100's). Any longer than this typically indicates an issue with client configuration or server-side controller load. Please see the troubleshooting section.

A successful fail-over can be verified by checking the `BytesInPerSec` and `LeaderCount` metrics increase on other brokers which proves that the traffic and leadership moved as expected. You will also observe an increase in the `UnderReplicatedPartitions` metric, which is expected when replicas are offline with the shutdown broker.

Troubleshooting

The above flow can be disrupted by breaking the client-server contract. The most common reasons for issue include:

- Misconfiguration or incorrect usage of Kafka client libs.
- Unexpected default behaviours and bugs with 3rd party client libs.
- Overloaded controller resulting in slower partition leader assignment.
- New controller is being elected resulting in slower partition leader assignment.

In order to ensure correct behaviour to handle leadership fail-over, we recommend:

- Server side [best practices](#) must be followed to ensure that the controller broker is scaled appropriately to avoid slow leadership assignment.
- Client libraries must have retries enabled to ensure that client handles the failover.
- Client libraries must have `retry.backoff.ms` configured (default 100) to avoid connection/request storms.
- Client libraries must set `request.timeout.ms` and `delivery.timeout.ms` to values inline with the applications' SLA. Higher values will result in slower fail-over for certain failure types.
- Client libraries must ensure that `bootstrap.servers` contains at least 3 random brokers to avoid an availability impact on initial discovery.
- Some client libraries are lower level than others and expect the application developer to implement retry logic and exception handling themselves. Please refer to client lib specific documentation for example usage, and ensure that correct reconnect/retry logic is followed.
- We recommend monitoring client side latency for produces, successful request count, and error count for non-retryable errors.
- We have observed that older 3rd party go lang and ruby libraries remain verbose during an entire broker offline time period despite produces and consume requests being unaffected. We

recommend you always monitor your business level metrics besides request metrics for success and errors to determine if there is real impact vs noise in your logs.

- Customers should not alarm on transient exceptions for network/not_leader as they are normal, non-impacting, and expected as part of the kafka protocol.
- Customers should not alarm on UnderReplicatedPartitions as they are normal, non-impacting, and expected during a single offline broker.

Amazon MSK logging

You can deliver Apache Kafka broker logs to one or more of the following destination types: Amazon CloudWatch Logs, Amazon S3, Amazon Data Firehose. You can also log Amazon MSK API calls with AWS CloudTrail.

Note

Broker logs are not available on Express brokers.

Broker logs

Broker logs enable you to troubleshoot your Apache Kafka applications and to analyze their communications with your MSK cluster. You can configure your new or existing MSK cluster to deliver INFO-level broker logs to one or more of the following types of destination resources: a CloudWatch log group, an S3 bucket, a Firehose delivery stream. Through Firehose you can then deliver the log data from your delivery stream to OpenSearch Service. You must create a destination resource before you configure your cluster to deliver broker logs to it. Amazon MSK doesn't create these destination resources for you if they don't already exist. For information about these three types of destination resources and how to create them, see the following documentation:

- [Amazon CloudWatch Logs](#)
- [Amazon S3](#)
- [Amazon Data Firehose](#)

Required permissions

To configure a destination for Amazon MSK broker logs, the IAM identity that you use for Amazon MSK actions must have the permissions described in the [AWS managed policy: AmazonMSKFullAccess](#) policy.

To stream broker logs to an S3 bucket, you also need the `s3:PutBucketPolicy` permission. For information about S3 bucket policies, see [How Do I Add an S3 Bucket Policy?](#) in the Amazon S3 User Guide. For information about IAM policies in general, see [Access Management](#) in the IAM User Guide.

Required KMS key policy for use with SSE-KMS buckets

If you enabled server-side encryption for your S3 bucket using AWS KMS-managed keys (SSE-KMS) with a customer managed key, add the following to the key policy for your KMS key so that Amazon MSK can write broker files to the bucket.

```
{
  "Sid": "Allow Amazon MSK to use the key.",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "delivery.logs.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": "*"
}
```

Configure broker logs using the AWS Management Console

If you are creating a new cluster, look for the **Broker log delivery** heading in the **Monitoring** section. You can specify the destinations to which you want Amazon MSK to deliver your broker logs.

For an existing cluster, choose the cluster from your list of clusters, then choose the **Properties** tab. Scroll down to the **Log delivery** section and then choose its **Edit** button. You can specify the destinations to which you want Amazon MSK to deliver your broker logs.

Configure broker logs using the AWS CLI

When you use the `create-cluster` or the `update-monitoring` commands, you can optionally specify the `logging-info` parameter and pass to it a JSON structure like the following example. In this JSON, all three destination types are optional.

```
{
  "BrokerLogs": {
    "S3": {
      "Bucket": "amzn-s3-demo-bucket",
      "Prefix": "ExamplePrefix",
      "Enabled": true
    },
    "Firehose": {
      "DeliveryStream": "ExampleDeliveryStreamName",
      "Enabled": true
    },
    "CloudWatchLogs": {
      "Enabled": true,
      "LogGroup": "ExampleLogGroupName"
    }
  }
}
```

Configure broker logs using the API

You can specify the optional `loggingInfo` structure in the JSON that you pass to the [CreateCluster](#) or [UpdateMonitoring](#) operations.

Note

By default, when broker logging is enabled, Amazon MSK logs INFO level logs to the specified destinations. However, users of Apache Kafka 2.4.X and later can dynamically set the broker log level to any of the [log4j log levels](#). For information about dynamically setting the broker log level, see [KIP-412: Extend Admin API to support dynamic application log levels](#). If you dynamically set the log level to DEBUG or TRACE, we recommend using Amazon S3 or Firehose as the log destination. If you use CloudWatch Logs as a log

destination and you dynamically enable DEBUG or TRACE level logging, Amazon MSK may continuously deliver a sample of logs. This can significantly impact broker performance and should only be used when the INFO log level is not verbose enough to determine the root cause of an issue.

Log API calls with AWS CloudTrail

Note

AWS CloudTrail logs are available for Amazon MSK only when you use [IAM access control](#).

Amazon MSK is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon MSK. CloudTrail captures API calls for as events. The calls captured include calls from the Amazon MSK console and code calls to the Amazon MSK API operations. It also captures Apache Kafka actions such as creating and altering topics and groups.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon MSK. 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 Amazon MSK or the Apache Kafka action, 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, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

Amazon MSK information in CloudTrail

CloudTrail is enabled on your Amazon Web Services account when you create the account. When supported event activity occurs in an MSK cluster, 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 Amazon Web Services account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your Amazon Web Services account, including events for Amazon MSK, 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 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 Amazon 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](#)

Amazon MSK logs all [Amazon MSK operations](#) as events in CloudTrail log files. In addition, it logs the following Apache Kafka actions.

- kafka-cluster:DescribeClusterDynamicConfiguration
- kafka-cluster:AlterClusterDynamicConfiguration
- kafka-cluster:CreateTopic
- kafka-cluster:DescribeTopicDynamicConfiguration
- kafka-cluster:AlterTopic
- kafka-cluster:AlterTopicDynamicConfiguration
- kafka-cluster>DeleteTopic

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 user 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](#).

Example: Amazon MSK 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 and Apache Kafka actions, so they don't appear in any specific order.

The following example shows CloudTrail log entries that demonstrate the `DescribeCluster` and `DeleteCluster` Amazon MSK actions.

```
{
  "Records": [
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "ABCDEF0123456789ABCDE",
        "arn": "arn:aws:iam::012345678901:user/Joe",
        "accountId": "012345678901",
        "accessKeyId": "AIDACKCEVSQ6C2EXAMPLE",
        "userName": "Joe"
      },
      "eventTime": "2018-12-12T02:29:24Z",
      "eventSource": "kafka.amazonaws.com",
      "eventName": "DescribeCluster",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "aws-cli/1.14.67 Python/3.6.0 Windows/10 botocore/1.9.20",
      "requestParameters": {
        "clusterArn": "arn%3Aaws%3Akafka%3Aus-east-1%3A012345678901%3Acluster%2Fexamplecluster%2F01234567-abcd-0123-abcd-abcd0123efa-2"
      },
      "responseElements": null,
      "requestID": "bd83f636-fdb5-abcd-0123-157e2fbf2bde",
      "eventID": "60052aba-0123-4511-bcde-3e18dbd42aa4",
      "readOnly": true,
      "eventType": "AwsApiCall",
      "recipientAccountId": "012345678901"
    },
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "ABCDEF0123456789ABCDE",
        "arn": "arn:aws:iam::012345678901:user/Joe",
        "accountId": "012345678901",
```

```

        "accessKeyId": "AIDACKCEVSQ6C2EXAMPLE",
        "userName": "Joe"
    },
    "eventTime": "2018-12-12T02:29:40Z",
    "eventSource": "kafka.amazonaws.com",
    "eventName": "DeleteCluster",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.14.67 Python/3.6.0 Windows/10 botocore/1.9.20",
    "requestParameters": {
        "clusterArn": "arn%3Aaws%3Akafka%3Aus-east-1%3A012345678901%3Acluster%2Fexamplecluster%2F01234567-abcd-0123-abcd-abcd0123efa-2"
    },
    "responseElements": {
        "clusterArn": "arn:aws:kafka:us-east-1:012345678901:cluster/examplecluster/01234567-abcd-0123-abcd-abcd0123efa-2",
        "state": "DELETING"
    },
    "requestID": "c6bfb3f7-abcd-0123-afa5-293519897703",
    "eventID": "8a7f1fcf-0123-abcd-9bdb-1ebf0663a75c",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "012345678901"
}
]
}

```

The following example shows a CloudTrail log entry that demonstrates the `kafka-cluster:CreateTopic` action.

```

{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "ABCDEFGH1IJKLMN2P34Q5",
        "arn": "arn:aws:iam::111122223333:user/Admin",
        "accountId": "111122223333",
        "accessKeyId": "CDEFAB1C2UUUUU3AB4TT",
        "userName": "Admin"
    },
    "eventTime": "2021-03-01T12:51:19Z",
    "eventSource": "kafka-cluster.amazonaws.com",
    "eventName": "CreateTopic",

```

```
"awsRegion": "us-east-1",
"sourceIPAddress": "198.51.100.0/24",
"userAgent": "aws-msk-iam-auth/unknown-version/aws-internal/3 aws-sdk-java/1.11.970
Linux/4.14.214-160.339.amzn2.x86_64 OpenJDK_64-Bit_Server_VM/25.272-b10 java/1.8.0_272
scala/2.12.8 vendor/Red_Hat,_Inc.",
"requestParameters": {
  "kafkaAPI": "CreateTopics",
  "resourceARN": "arn:aws:kafka:us-east-1:111122223333:topic/IamAuthCluster/3ebafd8e-
dae9-440d-85db-4ef52679674d-1/Topic9"
},
"responseElements": null,
"requestID": "e7c5e49f-6aac-4c9a-a1d1-c2c46599f5e4",
"eventID": "be1f93fd-4f14-4634-ab02-b5a79cb833d2",
"readOnly": false,
"eventType": "AwsApiCall",
"managementEvent": true,
"eventCategory": "Management",
"recipientAccountId": "111122223333"
}
```

Metadata management

Amazon MSK supports Apache ZooKeeper or KRaft metadata management modes.

From Apache Kafka version 3.7.x on Amazon MSK, you can create clusters that use KRaft mode instead of ZooKeeper mode. KRaft-based clusters rely on controllers within Kafka to manage metadata.

Topics

- [ZooKeeper mode](#)
- [KRaft mode](#)

ZooKeeper mode

[Apache ZooKeeper](#) is "a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications," including Apache Kafka.

If your cluster is using ZooKeeper mode, you can use the steps below to get the Apache ZooKeeper connection string. However, we recommend that you use the `BootstrapServerString` to

connect to your cluster and perform admin operations as the `--zookeeper` flag has been deprecated in Kafka 2.5 and is removed from Kafka 3.0.

Getting the Apache ZooKeeper connection string using the AWS Management Console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. The table shows all the clusters for the current region under this account. Choose the name of a cluster to view its description.
3. On the **Cluster summary** page, choose **View client information**. This shows you the bootstrap brokers, as well as the Apache ZooKeeper connection string.

Getting the Apache ZooKeeper connection string using the AWS CLI

1. If you don't know the Amazon Resource Name (ARN) of your cluster, you can find it by listing all the clusters in your account. For more information, see [the section called "List clusters"](#).
2. To get the Apache ZooKeeper connection string, along with other information about your cluster, run the following command, replacing *ClusterArn* with the ARN of your cluster.

```
aws kafka describe-cluster --cluster-arn ClusterArn
```

The output of this `describe-cluster` command looks like the following JSON example.

```
{
  "ClusterInfo": {
    "BrokerNodeGroupInfo": {
      "BrokerAZDistribution": "DEFAULT",
      "ClientSubnets": [
        "subnet-0123456789abcdef0",
        "subnet-2468013579abcdef1",
        "subnet-1357902468abcdef2"
      ],
      "InstanceType": "kafka.m5.large",
      "StorageInfo": {
        "EbsStorageInfo": {
          "VolumeSize": 1000
        }
      }
    },
    "ClusterArn": "arn:aws:kafka:us-east-1:111122223333:cluster/testcluster/12345678-abcd-4567-2345-abcdef123456-2",
  }
}
```

```

    "ClusterName": "testcluster",
    "CreationTime": "2018-12-02T17:38:36.75Z",
    "CurrentBrokerSoftwareInfo": {
        "KafkaVersion": "2.2.1"
    },
    "CurrentVersion": "K13V1IB3VIYZZH",
    "EncryptionInfo": {
        "EncryptionAtRest": {
            "DataVolumeKMSKeyId": "arn:aws:kms:us-east-1:555555555555:key/12345678-abcd-2345-ef01-abcdef123456"
        }
    },
    "EnhancedMonitoring": "DEFAULT",
    "NumberOfBrokerNodes": 3,
    "State": "ACTIVE",
    "ZookeeperConnectString": "10.0.1.101:2018,10.0.2.101:2018,10.0.3.101:2018"
}

```

The previous JSON example shows the `ZookeeperConnectString` key in the output of the `describe-cluster` command. Copy the value corresponding to this key and save it for when you need to create a topic on your cluster.

Important

Your Amazon MSK cluster must be in the `ACTIVE` state for you to be able to obtain the Apache ZooKeeper connection string. When a cluster is still in the `CREATING` state, the output of the `describe-cluster` command doesn't include `ZookeeperConnectString`. If this is the case, wait a few minutes and then run the `describe-cluster` again after your cluster reaches the `ACTIVE` state.

Getting the Apache ZooKeeper connection string using the API

To get the Apache ZooKeeper connection string using the API, see [DescribeCluster](#).

KRaft mode

Amazon MSK introduced support for KRaft (Apache Kafka Raft) in Kafka version 3.7.x. The Apache Kafka community developed KRaft to replace [Apache ZooKeeper](#) for metadata management in Apache Kafka clusters. In KRaft mode, cluster metadata is propagated within a group of

Kafka controllers, which are part of the Kafka cluster, instead of across ZooKeeper nodes. KRaft controllers are included at no additional cost to you, and require no additional setup or management from you. See [KIP-500](#) for more information about KRaft.

Here are some points to note about KRaft mode on MSK:

- KRaft mode is only available for new clusters. You cannot switch metadata modes once the cluster is created.
- On the MSK console, you can create a KRaft-based cluster by choosing Kafka version 3.7.x and selecting the KRaft checkbox in the cluster creation window.
- To create a cluster in KRaft mode using the MSK API [CreateCluster](#) or [CreateClusterV2](#) operations, you should use `3.7.x.kraft` as the version. Use `3.7.x` as the version to create a cluster in ZooKeeper mode.
- The number of partitions per broker is the same on KRaft and ZooKeeper based clusters. However, KRaft allows you to host more partitions per cluster by provisioning [more brokers in a cluster](#).
- There are no API changes required to use KRaft mode on Amazon MSK. However, if your clients still use the `--zookeeper` connection string today, you should update your clients to use the `--bootstrap-server` connection string to connect to your cluster. The `--zookeeper` flag is deprecated in Apache Kafka version 2.5 and is removed starting with Kafka version 3.0. We therefore recommend you use recent Apache Kafka client versions and the `--bootstrap-server` connection string for all connections to your cluster.
- ZooKeeper mode continues to be available for all released versions where zookeeper is also supported by Apache Kafka. See [Supported Apache Kafka versions](#) for details on the end of support for Apache Kafka versions and future updates.
- You should check that any tools you use are capable of using Kafka Admin APIs without ZooKeeper connections. Refer to [Use LinkedIn's Cruise Control for Apache Kafka with Amazon MSK](#) for updated steps to connect your cluster to Cruise Control. Cruise Control also has instructions for [running Cruise Control without ZooKeeper](#).
- You do not need to access your cluster's KRaft controllers directly for any administrative actions. However, if you are using open monitoring to collect metrics, you also need the DNS endpoints of your controllers in order to collect some non-controller related metrics about your cluster. You can get these DNS endpoints from the MSK Console or using the [ListNodes](#) API operation. See [Monitor an MSK Provisioned cluster with Prometheus](#) for updated steps for setting up open-monitoring for KRaft-based clusters.

- There are no additional [CloudWatch metrics](#) you need to monitor for KRaft mode clusters over ZooKeeper mode clusters. MSK manages the KRaft controllers used in your clusters.
- You can continue managing ACLs using in KRaft mode clusters using the `--bootstrap-server` connection string. You should not use the `--zookeeper` connection string to manage ACLs. See [Apache Kafka ACLs](#).
- In KRaft mode, your cluster's metadata is stored on KRaft controllers within Kafka and not external ZooKeeper nodes. Therefore, you don't need to control access to controller nodes separately [as you do with ZooKeeper nodes](#).

Amazon MSK resources

The term *resources* has two meanings in Amazon MSK, depending on the context. In the context of APIs a resource is a structure on which you can invoke an operation. For a list of these resources and the operations that you can invoke on them, see [Resources](#) in the Amazon MSK API Reference. In the context of [the section called "IAM access control"](#), a resource is an entity to which you can allow or deny access, as defined in the [the section called "Authorization policy resources"](#) section.

Apache Kafka versions

When you create an Amazon MSK cluster, you specify which Apache Kafka version you want to have on it. You can also update the Apache Kafka version of an existing cluster. The topics in the chapter help you understand timelines for Kafka version support and suggestions for best practices.

Topics

- [Supported Apache Kafka versions](#)
- [Amazon MSK version support](#)

Supported Apache Kafka versions

Amazon Managed Streaming for Apache Kafka (Amazon MSK) supports the following Apache Kafka and Amazon MSK versions. The Apache Kafka community provides approximately 12 months of support for a version after its release date. For more details check the [Apache Kafka EOL \(end of life\) policy](#).

Supported Apache Kafka versions

Apache Kafka version	MSK release date	End of support date
1.1.1	--	2024-06-05
2.1.0	--	2024-06-05
2.2.1	2019-07-31	2024-06-08
2.3.1	2019-12-19	2024-06-08
2.4.1	2020-04-02	2024-06-08
2.4.1.1	2020-09-09	2024-06-08
2.5.1	2020-09-30	2024-06-08
2.6.0	2020-10-21	2024-09-11
2.6.1	2021-01-19	2024-09-11
2.6.2	2021-04-29	2024-09-11
2.6.3	2021-12-21	2024-09-11
2.7.0	2020-12-29	2024-09-11
2.7.1	2021-05-25	2024-09-11
2.7.2	2021-12-21	2024-09-11
2.8.0	2021-05-19	2024-09-11
2.8.1	2022-10-28	2024-09-11
2.8.2-tiered	2022-10-28	2025-01-14
3.1.1	2022-06-22	2024-09-11
3.2.0	2022-06-22	2024-09-11
3.3.1	2022-10-26	2024-09-11

Apache Kafka version	MSK release date	End of support date
3.3.2	2023-03-02	2024-09-11
3.4.0	2023-05-04	2025-08-04
3.5.1	2023-09-26	2025-10-23
3.6.0	2023-11-16	--
3.7.x	2024-05-29	--
3.8.x	2025-02-20	--
3.9.x	2025-04-21	--
4.0.x	2025-05-16	--

For more information about Amazon MSK version support policy, see [Amazon MSK version support policy](#).

Amazon MSK version 4.0.x

Amazon Managed Streaming for Apache Kafka (Amazon MSK) now supports Apache Kafka version 4.0. This version brings the latest advancements in cluster management and performance to MSK Provisioned. Kafka 4.0 introduces a new consumer rebalance protocol, now generally available, that helps ensure smoother and faster group rebalances. In addition, Kafka 4.0 requires brokers and tools to use Java 17, providing improved security and performance, includes various bug fixes and improvements, and deprecates metadata management via Apache ZooKeeper.

For more details and a complete list of improvements and bug fixes, see the [Apache Kafka release notes for version 4.0](#).

Amazon MSK version 3.9.x

Amazon Managed Streaming for Apache Kafka (Amazon MSK) now supports Apache Kafka version 3.9. This version allows you to retain tiered data when disabling tiered storage at the topic level. Consumer applications can continue to read historical data from the remote log start offset (Rx) while maintaining continuous log offsets across both local and remote storage.

For more details and a complete list of improvements and bug fixes, see the [Apache Kafka release notes for version 3.9.x](#).

Amazon MSK version 3.8.x

Amazon Managed Streaming for Apache Kafka (Amazon MSK) now supports Apache Kafka version 3.8. You can now create new clusters using version 3.8 with either KRAFT or ZooKeeper mode for metadata management or upgrade your existing ZooKeeper based clusters to use version 3.8. Apache Kafka version 3.8 includes several bug fixes and new features that improve performance. Key new features include support for compression level configuration. This allows you to further optimize your performance when using compression types such as lz4, zstd and gzip, by allowing you to change the default compression level.

For more details and a complete list of improvements and bug fixes, see the [Apache Kafka release notes for version 3.8.x](#).

Apache Kafka version 3.7.x (with production-ready tiered storage)

Apache Kafka version 3.7.x on MSK includes support for Apache Kafka version 3.7.0. You can create clusters or upgrade existing clusters to use the new 3.7.x version. With this change in version naming, you no longer have to adopt newer patch fix versions such as 3.7.1 when they are released by the Apache Kafka community. Amazon MSK will automatically update 3.7.x to support future patch versions once they become available. This allows you to benefit from the security and bug fixes available through patch fix versions without triggering a version upgrade. These patch fix versions released by Apache Kafka don't break version compatibility and you can benefit from the new patch fix versions without worrying about read or write errors for your client applications. Please make sure your infrastructure automation tools, such as CloudFormation, are updated to account for this change in version naming.

Amazon MSK now supports KRaft mode (Apache Kafka Raft) in Apache Kafka version 3.7.x. On Amazon MSK, like with ZooKeeper nodes, KRaft controllers are included at no additional cost to you, and require no additional setup or management from you. You can now create clusters in either KRaft mode or ZooKeeper mode on Apache Kafka version 3.7.x. In KRaft mode, you can add up to 60 brokers to host more partitions per-cluster, without requesting a limit increase, compared to the 30-broker quota on Zookeeper-based clusters. To learn more about KRaft on MSK, see [KRaft mode](#).

Apache Kafka version 3.7.x also includes several bug fixes and new features that improve performance. Key improvements include leader discovery optimizations for clients and log

segment flush optimization options. For a complete list of improvements and bug fixes, see the Apache Kafka release notes for [3.7.0](#).

Apache Kafka version 3.6.0 (with production-ready tiered storage)

For information about Apache Kafka version 3.6.0 (with production-ready tiered storage), see its [release notes](#) on the Apache Kafka downloads site.

Amazon MSK will continue to use and manage Zookeeper for quorum management in this release for stability.

Amazon MSK version 3.5.1

Amazon Managed Streaming for Apache Kafka (Amazon MSK) now supports Apache Kafka version 3.5.1 for new and existing clusters. Apache Kafka 3.5.1 includes several bug fixes and new features that improve performance. Key features include the introduction of new rack-aware partition assignment for consumers. Amazon MSK will continue to use and manage Zookeeper for quorum management in this release. For a complete list of improvements and bug fixes, see the Apache Kafka release notes for 3.5.1.

For information about Apache Kafka version 3.5.1, see its [release notes](#) on the Apache Kafka downloads site.

Amazon MSK version 3.4.0

Amazon Managed Streaming for Apache Kafka (Amazon MSK) now supports Apache Kafka version 3.4.0 for new and existing clusters. Apache Kafka 3.4.0 includes several bug fixes and new features that improve performance. Key features include a fix to improve stability to fetch from the closest replica. Amazon MSK will continue to use and manage Zookeeper for quorum management in this release. For a complete list of improvements and bug fixes, see the Apache Kafka release notes for 3.4.0.

For information about Apache Kafka version 3.4.0, see its [release notes](#) on the Apache Kafka downloads site.

Amazon MSK version 3.3.2

Amazon Managed Streaming for Apache Kafka (Amazon MSK) now supports Apache Kafka version 3.3.2 for new and existing clusters. Apache Kafka 3.3.2 includes several bug fixes and new features that improve performance. Key features include a fix to improve stability to fetch from the closest

replica. Amazon MSK will continue to use and manage Zookeeper for quorum management in this release. For a complete list of improvements and bug fixes, see the Apache Kafka release notes for 3.3.2.

For information about Apache Kafka version 3.3.2, see its [release notes](#) on the Apache Kafka downloads site.

Amazon MSK version 3.3.1

Amazon Managed Streaming for Apache Kafka (Amazon MSK) now supports Apache Kafka version 3.3.1 for new and existing clusters. Apache Kafka 3.3.1 includes several bug fixes and new features that improve performance. Some of the key features include enhancements to metrics and partitioner. Amazon MSK will continue to use and manage Zookeeper for quorum management in this release for stability. For a complete list of improvements and bug fixes, see the Apache Kafka release notes for 3.3.1.

For information about Apache Kafka version 3.3.1, see its [release notes](#) on the Apache Kafka downloads site.

Amazon MSK version 3.1.1

Amazon Managed Streaming for Apache Kafka (Amazon MSK) now supports Apache Kafka version 3.1.1 and 3.2.0 for new and existing clusters. Apache Kafka 3.1.1 and Apache Kafka 3.2.0 includes several bug fixes and new features that improve performance. Some of the key features include enhancements to metrics and the use of topic IDs. MSK will continue to use and manage Zookeeper for quorum management in this release for stability. For a complete list of improvements and bug fixes, see the Apache Kafka release notes for 3.1.1 and 3.2.0.

For information about Apache Kafka version 3.1.1 and 3.2.0, see its [3.2.0 release notes](#) and [3.1.1 release notes](#) on the Apache Kafka downloads site.

Amazon MSK tiered storage version 2.8.2.tiered

This release is an Amazon MSK-only version of Apache Kafka version 2.8.2, and is compatible with open source Apache Kafka clients.

The 2.8.2.tiered release contains tiered storage functionality that is compatible with APIs introduced in [KIP-405 for Apache Kafka](#). For more information about the Amazon MSK tiered storage feature, see [Tiered storage for Standard brokers](#).

Apache Kafka version 2.5.1

Apache Kafka version 2.5.1 includes several bug fixes and new features, including encryption in-transit for Apache ZooKeeper and administration clients. Amazon MSK provides TLS ZooKeeper endpoints, which you can query with the [DescribeCluster](#) operation.

The output of the [DescribeCluster](#) operation includes the `ZookeeperConnectStringTls` node, which lists the TLS zookeeper endpoints.

The following example shows the `ZookeeperConnectStringTls` node of the response for the `DescribeCluster` operation:

```
"ZookeeperConnectStringTls": "z-3.awskafkatutorialc.abcd123.c3.kafka.us-east-1.amazonaws.com:2182,z-2.awskafkatutorialc.abcd123.c3.kafka.us-east-1.amazonaws.com:2182,z-1.awskafkatutorialc.abcd123.c3.kafka.us-east-1.amazonaws.com:2182"
```

For information about using TLS encryption with zookeeper, see [Using TLS security with Apache ZooKeeper](#).

For more information about Apache Kafka version 2.5.1, see its [release notes](#) on the Apache Kafka downloads site.

Amazon MSK bug-fix version 2.4.1.1

This release is an Amazon MSK-only bug-fix version of Apache Kafka version 2.4.1. This bug-fix release contains a fix for [KAFKA-9752](#), a rare issue that causes consumer groups to continuously rebalance and remain in the `PreparingRebalance` state. This issue affects clusters running Apache Kafka versions 2.3.1 and 2.4.1. This release contains a community-produced fix that is available in Apache Kafka version 2.5.0.

Note

Amazon MSK clusters running version 2.4.1.1 are compatible with any Apache Kafka client that is compatible with Apache Kafka version 2.4.1.

We recommend that you use MSK bug-fix version 2.4.1.1 for new Amazon MSK clusters if you prefer to use Apache Kafka 2.4.1. You can update existing clusters running Apache Kafka version

2.4.1 to this version to incorporate this fix. For information about upgrading an existing cluster, see [Upgrade the Apache Kafka version](#).

To work around this issue without upgrading the cluster to version 2.4.1.1, see the [Consumer group stuck in PreparingRebalance state](#) section of the [Troubleshoot your Amazon MSK cluster](#) guide.

Apache Kafka version 2.4.1 (use 2.4.1.1 instead)

Note

You can no longer create an MSK cluster with Apache Kafka version 2.4.1. Instead, you can use [Amazon MSK bug-fix version 2.4.1.1](#) with clients compatible with Apache Kafka version 2.4.1. And if you already have an MSK cluster with Apache Kafka version 2.4.1, we recommend you update it to use Apache Kafka version 2.4.1.1 instead.

KIP-392 is one of the key Kafka Improvement Proposals that are included in the 2.4.1 release of Apache Kafka. This improvement allows consumers to fetch from the closest replica. To use this feature, set `client.rack` in the consumer properties to the ID of the consumer's Availability Zone. An example AZ ID is `use1-az1`. Amazon MSK sets `broker.rack` to the IDs of the Availability Zones of the brokers. You must also set the `replica.selector.class` configuration property to `org.apache.kafka.common.replica.RackAwareReplicaSelector`, which is an implementation of rack awareness provided by Apache Kafka.

When you use this version of Apache Kafka, the metrics in the `PER_TOPIC_PER_BROKER` monitoring level appear only after their values become nonzero for the first time. For more information about this, see [the section called "PER_TOPIC_PER_BROKER Level monitoring"](#).

For information about how to find Availability Zone IDs, see [AZ IDs for Your Resource](#) in the AWS Resource Access Manager user guide.

For information about setting configuration properties, see [the section called "Broker configuration"](#).

For more information about KIP-392, see [Allow Consumers to Fetch from Closest Replica](#) in the Confluence pages.

For more information about Apache Kafka version 2.4.1, see its [release notes](#) on the Apache Kafka downloads site.

Amazon MSK version support

This topic describes the [Amazon MSK version support policy](#) and the procedure for [Upgrade the Apache Kafka version](#). If you're upgrading your Kafka version, follow the best practices outlined in [Best practices for version upgrades](#).

Topics

- [Amazon MSK version support policy](#)
- [Upgrade the Apache Kafka version](#)
- [Best practices for version upgrades](#)

Amazon MSK version support policy

This section describes the support policy for Amazon MSK supported Kafka versions.

- All Kafka versions are supported until they reach their end of support date. For details on end of support dates, see [Supported Apache Kafka versions](#). Upgrade your MSK cluster to the recommended Kafka version or higher version before the end of support date. For details about upgrading your Apache Kafka version, see [Upgrade the Apache Kafka version](#). A cluster using a Kafka version after its end of support date is auto-upgraded to the recommended Kafka version. Automatic upgrades can happen at any time after the end of support date. You will not receive any notification before the upgrade.
- MSK will phase out support for newly created clusters that use Kafka versions with published end of support dates.

Upgrade the Apache Kafka version

You can upgrade an existing MSK cluster to a newer version of Apache Kafka.

Note

- You can't upgrade an existing MSK cluster from a ZooKeeper-based Apache Kafka version to a newer version that uses or requires KRaft mode. Instead, to upgrade your cluster, create a new MSK cluster with a KRaft-supported Kafka version and migrate your data and workloads from the old cluster.
- Amazon MSK only upgrades the server software. It doesn't upgrade your clients.

- You can't downgrade an existing MSK cluster to an older version of Apache Kafka.

When you upgrade the Apache Kafka version of an MSK cluster, also check your client-side software to make sure its version enables you to use the features of the cluster's new Apache Kafka version.

For information about how to make a cluster highly available during an upgrade, see [the section called “Build highly available clusters”](#).

Upgrade the Apache Kafka version using the AWS Management Console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. In the navigation bar, choose the Region where you created the MSK cluster.
3. Choose the MSK cluster which you want to upgrade.
4. On the **Properties** tab, choose **Upgrade** in the **Apache Kafka version** section.
5. In the **Apache Kafka version** section, do the following:
 - a. In the *Choose Apache Kafka version* dropdown list, choose the version to which you want to upgrade. For example, choose **3.9.x**.
 - b. (Optional) Choose **Version compatibility** to view the compatibility between your cluster's current version and the version to which you want to upgrade. Then, choose **Choose** to proceed or choose **Cancel**.
 - c. Choose the **Update cluster configuration** checkbox to automatically apply a new Kafka configuration revision that's compatible with the upgraded version. This ensures compatibility and enables new features or improvements of the upgraded version. However, skip it if you want to maintain your existing custom configurations.
 - d. Choose **Upgrade**.

Upgrade the Apache Kafka version using the AWS CLI

1. Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see [the section called “List clusters”](#).

```
aws kafka get-compatible-kafka-versions --cluster-arn ClusterArn
```

The output of this command includes a list of the Apache Kafka versions to which you can upgrade the cluster. It looks like the following example.

```
{
  "CompatibleKafkaVersions": [
    {
      "SourceVersion": "2.2.1",
      "TargetVersions": [
        "2.3.1",
        "2.4.1",
        "2.4.1.1",
        "2.5.1"
      ]
    }
  ]
}
```

2. Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see [the section called "List clusters"](#).

Replace *Current-Cluster-Version* with the current version of the cluster. For *TargetVersion* you can specify any of the target versions from the output of the previous command.

Important

Cluster versions aren't simple integers. To find the current version of the cluster, use the [DescribeCluster](#) operation or the [describe-cluster](#) AWS CLI command. An example version is KTVDPKIKXØDER.

```
aws kafka update-cluster-kafka-version --cluster-arn ClusterArn --current-version Current-Cluster-Version --target-kafka-version TargetVersion
```

The output of the previous command looks like the following JSON.

```
{
  "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/
  abcdefab-1234-abcd-5678-cdef0123ab01-2",
  "ClusterOperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
  operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
  abcd-4f7f-1234-9876543210ef"
}
```

3. To get the result of the `update-cluster-kafka-version` operation, run the following command, replacing *ClusterOperationArn* with the ARN that you obtained in the output of the `update-cluster-kafka-version` command.

```
aws kafka describe-cluster-operation --cluster-operation-arn ClusterOperationArn
```

The output of this `describe-cluster-operation` command looks like the following JSON example.

```
{
  "ClusterOperationInfo": {
    "ClientRequestId": "62cd41d2-1206-4ebf-85a8-dbb2ba0fe259",
    "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/
    exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2",
    "CreationTime": "2021-03-11T20:34:59.648000+00:00",
    "OperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-
    operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-
    abcd-4f7f-1234-9876543210ef",
    "OperationState": "UPDATE_IN_PROGRESS",
    "OperationSteps": [
      {
        "StepInfo": {
          "StepStatus": "IN_PROGRESS"
        },
        "StepName": "INITIALIZE_UPDATE"
      },
      {
        "StepInfo": {
          "StepStatus": "PENDING"
        }
      }
    ]
  }
}
```

```

        "StepName": "UPDATE_APACHE_KAFKA_BINARIES"
      },
      {
        "StepInfo": {
          "StepStatus": "PENDING"
        },
        "StepName": "FINALIZE_UPDATE"
      }
    ],
    "OperationType": "UPDATE_CLUSTER_KAFKA_VERSION",
    "SourceClusterInfo": {
      "KafkaVersion": "2.4.1"
    },
    "TargetClusterInfo": {
      "KafkaVersion": "2.6.1"
    }
  }
}

```

If `OperationState` has the value `UPDATE_IN_PROGRESS`, wait a while, then run the `describe-cluster-operation` command again. When the operation is complete, the value of `OperationState` becomes `UPDATE_COMPLETE`. Because the time required for Amazon MSK to complete the operation varies, you might need to check repeatedly until the operation is complete.

Upgrade the Apache Kafka version using the API

1. Invoke the [GetCompatibleKafkaVersions](#) operation to get a list of the Apache Kafka versions to which you can upgrade the cluster.
2. Invoke the [UpdateClusterKafkaVersion](#) operation to upgrade the cluster to one of the compatible Apache Kafka versions.

Best practices for version upgrades

To ensure client continuity during the rolling update that is performed as part of the Kafka version upgrade process, review the configuration of your clients and your Apache Kafka topics as follows:

- Set the topic replication factor (RF) to a minimum value of 2 for two-AZ clusters and a minimum value of 3 for three-AZ clusters. An RF value of 2 can lead to offline partitions during patching.

- Set minimum in-sync replicas (minISR) to a maximum value of 1 less than your Replication Factor (RF), which is $\text{minISR} = (\text{RF}) - 1$. This makes sure that the partition replica set can tolerate one replica being offline or under-replicated.
- Configure clients to use multiple broker connection strings. Having multiple brokers in a client's connection string allows for failover if a specific broker supporting client I/O begins to be patched. For information about how to get a connection string with multiple brokers, see [Getting the bootstrap brokers for an Amazon MSK cluster](#).
- We recommend that you upgrade connecting clients to the recommended version or above to benefit from the features available in the new version. Client upgrades are not subject to the end of life (EOL) dates of your MSK cluster's Kafka version, and do not need to be completed by the EOL date. Apache Kafka provides a [bi-directional client compatibility policy](#) that allows older clients to work with newer clusters and vice versa.
- Kafka clients using versions 3.x.x are likely to come with the following defaults: `acks=all` and `enable.idempotence=true`. `acks=all` is different from the previous default of `acks=1` and provides extra durability by ensuring that all in-sync replicas acknowledge the produce request. Similarly, the default for `enable.idempotence` was previously `false`. The change to `enable.idempotence=true` as the default lowers the likelihood of duplicate messages. These changes are considered best practice settings and may introduce a small amount of additional latency that's within normal performance parameters.
- Use the recommended Kafka version when creating new MSK clusters. Using the recommended Kafka version allows you to benefit from the latest Kafka and MSK features.

Troubleshoot your Amazon MSK cluster

The following information can help you troubleshoot problems that you might have with your Amazon MSK cluster. You can also post your issue to [AWS re:Post](#). For troubleshooting Amazon MSK Replicator, see [Troubleshoot MSK Replicator](#).

Topics

- [Volume replacement causes disk saturation due to replication overload](#)
- [Consumer group stuck in PreparingRebalance state](#)
- [Error delivering broker logs to Amazon CloudWatch Logs](#)
- [No default security group](#)
- [Cluster appears stuck in the CREATING state](#)
- [Cluster state goes from CREATING to FAILED](#)

- [Cluster state is ACTIVE but producers cannot send data or consumers cannot receive data](#)
- [AWS CLI doesn't recognize Amazon MSK](#)
- [Partitions go offline or replicas are out of sync](#)
- [Disk space is running low](#)
- [Memory running low](#)
- [Producer gets NotLeaderForPartitionException](#)
- [Under-replicated partitions \(URP\) greater than zero](#)
- [Cluster has topics called __amazon_msk_canary and __amazon_msk_canary_state](#)
- [Partition replication fails](#)
- [Unable to access cluster that has public access turned on](#)
- [Unable to access cluster from within AWS: Networking issues](#)
- [Failed authentication: Too many connects](#)
- [Failed authentication: Session too short](#)
- [MSK Serverless: Cluster creation fails](#)
- [Can't update KafkaVersionsList in MSK configuration](#)

Volume replacement causes disk saturation due to replication overload

During unplanned volume hardware failure, Amazon MSK may replace the volume with a new instance. Kafka repopulates the new volume by replicating partitions from other brokers in the cluster. Once partitions are replicated and caught up, they are eligible for leadership and in-sync replica (ISR) membership.

Problem

In a broker recovering from volume replacement, some partitions of varying sizes may come back online before others. This can be problematic as those partitions can be serving traffic from the same broker that is still catching up (replicating) other partitions. This replication traffic can sometimes saturate the underlying volume throughput limits, which is 250 MiB per second in the default case. When this saturation occurs, any partitions that are already caught up will be impacted, resulting in latency across the cluster for any brokers sharing ISR with those caught up partitions (not just leader partitions due to remote acks `acks=all`). This problem is more common with larger clusters that have larger numbers of partitions that vary in size.

Recommendation

- To improve replication I/O posture, ensure that [best practice thread settings](#) are in place.
- To reduce the likelihood of underlying volume saturation, enable provisioned storage with a higher throughput. A min throughput value of 500 MiB/s is recommended for high throughput replication cases, but the actual value needed will vary with throughput and use case. [Provision storage throughput for Standard brokers in a Amazon MSK cluster](#).
- To minimize replication pressure, lower `num.replica.fetchers` to the default value of 2.

Consumer group stuck in PreparingRebalance state

If one or more of your consumer groups is stuck in a perpetual rebalancing state, the cause might be Apache Kafka issue [KAFKA-9752](#), which affects Apache Kafka versions 2.3.1 and 2.4.1.

To resolve this issue, we recommend that you upgrade your cluster to [Amazon MSK bug-fix version 2.4.1.1](#), which contains a fix for this issue. For information about updating an existing cluster to Amazon MSK bug-fix version 2.4.1.1, see [Upgrade the Apache Kafka version](#).

The workarounds for solving this issue without upgrading the cluster to Amazon MSK bug-fix version 2.4.1.1 are to either set the Kafka clients to use [Static membership protocol](#), or to [Identify and reboot](#) the coordinating broker node of the stuck consumer group.

Implementing static membership protocol

To implement Static Membership Protocol in your clients, do the following:

1. Set the `group.instance.id` property of your [Kafka Consumers](#) configuration to a static string that identifies the consumer in the group.
2. Ensure that other instances of the configuration are updated to use the static string.
3. Deploy the changes to your Kafka Consumers.

Using Static Membership Protocol is more effective if the session timeout in the client configuration is set to a duration that allows the consumer to recover without prematurely triggering a consumer group rebalance. For example, if your consumer application can tolerate 5 minutes of unavailability, a reasonable value for the session timeout would be 4 minutes instead of the default value of 10 seconds.

Note

Using Static Membership Protocol only reduces the probability of encountering this issue. You may still encounter this issue even when using Static Membership Protocol.

Rebooting the coordinating broker node

To reboot the coordinating broker node, do the following:

1. Identify the group coordinator using the `kafka-consumer-groups.sh` command.
2. Restart the group coordinator of the stuck consumer group using the [RebootBroker](#) API action.

Error delivering broker logs to Amazon CloudWatch Logs

When you try to set up your cluster to send broker logs to Amazon CloudWatch Logs, you might get one of two exceptions.

If you get an `InvalidInput.LengthOfCloudWatchResourcePolicyLimitExceeded` exception, try again but use log groups that start with `/aws/vendedlogs/`. For more information, see [Enabling Logging from Certain Amazon Web Services](#).

If you get an `InvalidInput.NumberOfCloudWatchResourcePoliciesLimitExceeded` exception, choose an existing Amazon CloudWatch Logs policy in your account, and append the following JSON to it.

```
{"Sid": "AWSLogDeliveryWrite", "Effect": "Allow", "Principal":  
{"Service": "delivery.logs.amazonaws.com"}, "Action":  
["logs:CreateLogStream", "logs:PutLogEvents"], "Resource": ["*"]}
```

If you try to append the JSON above to an existing policy but get an error that says you've reached the maximum length for the policy you picked, try to append the JSON to another one of your Amazon CloudWatch Logs policies. After you append the JSON to an existing policy, try once again to set up broker-log delivery to Amazon CloudWatch Logs.

No default security group

If you try to create a cluster and get an error indicating that there's no default security group, it might be because you are using a VPC that was shared with you. Ask your administrator to grant you permission to describe the security groups on this VPC and try again. For an example of a policy that allows this action, see [Amazon EC2: Allows Managing EC2 Security Groups Associated With a Specific VPC, Programmatically and in the Console](#).

Cluster appears stuck in the CREATING state

Sometimes cluster creation can take up to 30 minutes. Wait for 30 minutes and check the state of the cluster again.

Cluster state goes from CREATING to FAILED

Try creating the cluster again.

Cluster state is ACTIVE but producers cannot send data or consumers cannot receive data

- If the cluster creation succeeds (the cluster state is ACTIVE), but you can't send or receive data, ensure that your producer and consumer applications have access to the cluster. For more information, see the guidance in [the section called "Create a client machine"](#).
- If your producers and consumers have access to the cluster but still experience problems producing and consuming data, the cause might be [KAFKA-7697](#), which affects Apache Kafka version 2.1.0 and can lead to a deadlock in one or more brokers. Consider migrating to Apache Kafka 2.2.1, which is not affected by this bug. For information about how to migrate, see [the section called "Migrate to Amazon MSK Cluster"](#).

AWS CLI doesn't recognize Amazon MSK

If you have the AWS CLI installed, but it doesn't recognize the Amazon MSK commands, upgrade your AWS CLI to the latest version. For detailed instructions on how to upgrade the AWS CLI, see [Installing the AWS Command Line Interface](#). For information about how to use the AWS CLI to run Amazon MSK commands, see [the section called "Key features and concepts"](#).

Partitions go offline or replicas are out of sync

These can be symptoms of low disk space. See [the section called “Disk space is running low”](#).

Disk space is running low

See the following best practices for managing disk space: [the section called “Monitor disk space”](#) and [the section called “Adjust data retention parameters”](#).

Memory running low

If you see the `MemoryUsed` metric running high or `MemoryFree` running low, that doesn't mean there's a problem. Apache Kafka is designed to use as much memory as possible, and it manages it optimally.

Producer gets `NotLeaderForPartitionException`

This is often a transient error. Set the producer's `retries` configuration parameter to a value that's higher than its current value.

Under-replicated partitions (URP) greater than zero

The `UnderReplicatedPartitions` metric is an important one to monitor. In a healthy MSK cluster, this metric has the value 0. If it's greater than zero, that might be for one of the following reasons.

- If `UnderReplicatedPartitions` is spiky, the issue might be that the cluster isn't provisioned at the right size to handle incoming and outgoing traffic. See [the section called “Best practices for Standard brokers”](#).
- If `UnderReplicatedPartitions` is consistently greater than 0 including during low-traffic periods, the issue might be that you've set restrictive ACLs that don't grant topic access to brokers. To replicate partitions, brokers must be authorized to both `READ` and `DESCRIBE` topics. `DESCRIBE` is granted by default with the `READ` authorization. For information about setting ACLs, see [Authorization and ACLs](#) in the Apache Kafka documentation.

Cluster has topics called `__amazon_msk_canary` and `__amazon_msk_canary_state`

You might see that your MSK cluster has a topic with the name `__amazon_msk_canary` and another with the name `__amazon_msk_canary_state`. These are internal topics that Amazon

MSK creates and uses for cluster health and diagnostic metrics. These topics are negligible in size and can't be deleted.

Partition replication fails

Ensure that you haven't set ACLs on `CLUSTER_ACTIONS`.

Unable to access cluster that has public access turned on

If your cluster has public access turned on, but you still cannot access it from the internet, follow these steps:

1. Ensure that the cluster's security group's inbound rules allow your IP address and the cluster's port. For a list of cluster port numbers, see [the section called "Port information"](#). Also ensure that the security group's outbound rules allow outbound communications. For more information about security groups and their inbound and outbound rules, see [Security groups for your VPC](#) in the Amazon VPC User Guide.
2. Make sure that your IP address and the cluster's port are allowed in the inbound rules of the cluster's VPC network ACL. Unlike security groups, network ACLs are stateless. This means that you must configure both inbound and outbound rules. In the outbound rules, allow all traffic (port range: 0-65535) to your IP address. For more information, see [Add and delete rules](#) in the Amazon VPC User Guide.
3. Make sure that you are using the public-access bootstrap-brokers string to access the cluster. An MSK cluster that has public access turned on has two different bootstrap-brokers strings, one for public access, and one for access from within AWS. For more information, see [the section called "Get the bootstrap brokers using the AWS Management Console"](#).

Unable to access cluster from within AWS: Networking issues

If you have an Apache Kafka application that is unable to communicate successfully with an MSK cluster, start by performing the following connectivity test.

1. Use any of the methods described in [the section called "Get the bootstrap brokers"](#) to get the addresses of the bootstrap brokers.
2. In the following command replace *bootstrap-broker* with one of the broker addresses that you obtained in the previous step. Replace *port-number* with 9094 if the cluster is set up to use TLS authentication. If the cluster doesn't use TLS authentication, replace *port-number* with 9092. Run the command from the client machine.

```
telnet bootstrap-broker port-number
```

Where *port-number* is:

- 9094 if the cluster is set up to use TLS authentication.
- 9092 If the cluster doesn't use TLS authentication.
- A different port-number is required if public access is enabled.

Run the command from the client machine.

3. Repeat the previous command for all the bootstrap brokers.

If the client machine is able to access the brokers, this means there are no connectivity issues. In this case, run the following command to check whether your Apache Kafka client is set up correctly. To get *bootstrap-brokers*, use any of the methods described in [the section called "Get the bootstrap brokers"](#). Replace *topic* with the name of your topic.

```
<path-to-your-kafka-installation>/bin/kafka-console-producer.sh --broker-  
list bootstrap-brokers --producer.config client.properties --topic topic
```

If the previous command succeeds, this means that your client is set up correctly. If you're still unable to produce and consume from an application, debug the problem at the application level.

If the client machine is unable to access the brokers, see the following subsections for guidance that is based on your client-machine setup.

Amazon EC2 client and MSK cluster in the same VPC

If the client machine is in the same VPC as the MSK cluster, make sure the cluster's security group has an inbound rule that accepts traffic from the client machine's security group. For information about setting up these rules, see [Security Group Rules](#). For an example of how to access a cluster from an Amazon EC2 instance that's in the same VPC as the cluster, see [the section called "Get started"](#).

Amazon EC2 client and MSK cluster in different VPCs

If the client machine and the cluster are in two different VPCs, ensure the following:

- The two VPCs are peered.
- The status of the peering connection is active.
- The route tables of the two VPCs are set up correctly.

For information about VPC peering, see [Working with VPC Peering Connections](#).

On-premises client

In the case of an on-premises client that is set up to connect to the MSK cluster using AWS VPN, ensure the following:

- The VPN connection status is UP. For information about how to check the VPN connection status, see [How do I check the current status of my VPN tunnel?](#)
- The route table of the cluster's VPC contains the route for an on-premises CIDR whose target has the format `Virtual private gateway(vgw-xxxxxxx)`.
- The MSK cluster's security group allows traffic on port 2181, port 9092 (if your cluster accepts plaintext traffic), and port 9094 (if your cluster accepts TLS-encrypted traffic).

For more AWS VPN troubleshooting guidance, see [Troubleshooting Client VPN](#).

AWS Direct Connect

If the client uses AWS Direct Connect, see [Troubleshooting AWS Direct Connect](#).

If the previous troubleshooting guidance doesn't resolve the issue, ensure that no firewall is blocking network traffic. For further debugging, use tools like `tcpdump` and `Wireshark` to analyze traffic and to make sure that it is reaching the MSK cluster.

Failed authentication: Too many connects

The `Failed authentication ... Too many connects` error indicates that a broker is protecting itself because one or more IAM clients are trying to connect to it at an aggressive rate. To help brokers accept a higher rate of new IAM connections, you can increase the [reconnect.backoff.ms](#) configuration parameter.

To learn more about the rate limits for new connections per broker, see the [Amazon MSK quota](#) page.

Failed authentication: Session too short

The Failed authentication ... Session too short error occurs when your client tries to connect to a cluster using IAM credentials that are about to expire. Make sure that you check how your IAM credentials are being refreshed. Most likely, the credentials are being replaced too close to session expiry which leads to issues on the server side, and authentication failures.

MSK Serverless: Cluster creation fails

If you try to create an MSK Serverless cluster and the workflow fails, you may not have permission to create a VPC endpoint. Verify that your administrator has granted you permission to create a VPC endpoint by allowing the `ec2:CreateVpcEndpoint` action.

For a complete list of permissions required to perform all Amazon MSK actions, see [AWS managed policy: AmazonMSKFullAccess](#).

Can't update KafkaVersionsList in MSK configuration

When you update the [KafkaVersionsList](#) property in the [AWS::MSK::Configuration](#) resource, the update fails with the following error.

```
Resource of type 'AWS::MSK::Configuration' with identifier '<identifierName>' already exists.
```

When you update the `KafkaVersionsList` property, AWS CloudFormation recreates a new configuration with the updated property before deleting the old configuration. The AWS CloudFormation stack update fails because the new configuration uses the same name as the existing configuration. Such an update requires a [resource replacement](#). To successfully update `KafkaVersionsList`, you must also update the [Name](#) property in the same operation.

In addition, if your configuration is attached with any clusters created using the AWS Management Console or AWS CLI, add the following to your configuration resource to prevent [failed resource deletion attempts](#).

```
UpdateReplacePolicy: Retain
```

After the update succeeds, go to the Amazon MSK console and delete the old configuration. For information about MSK configurations, see [Amazon MSK Provisioned configuration](#).

Best practices for Standard and Express brokers

This section describes best practices to follow for Standard brokers and Express brokers. For information about Amazon MSK Replicator best practices, see [Best practices for using MSK Replicator](#).

Topics

- [Best practices for Standard brokers](#)
- [Best practices for Express brokers](#)
- [Best practices for Apache Kafka clients](#)

Best practices for Standard brokers

This topic outlines some best practices to follow when using Amazon MSK. For information about Amazon MSK Replicator best practices, see [Best practices for using MSK Replicator](#).

Client-side considerations

Your application's availability and performance depends not only on server-side settings but on client settings as well.

- Configure your clients for high availability. In a distributed system like Apache Kafka, ensuring high availability is crucial for maintaining a reliable and fault-tolerant messaging infrastructure. Brokers will go offline for both planned and unplanned events, for example upgrades, patching, hardware failure, and network issues. A Kafka cluster is tolerant of an offline broker, therefore Kafka clients must also handle broker fail-over gracefully. See the full details on [Best practices for Apache Kafka clients](#).
- Ensure client connection strings include at least one broker from each availability zone. Having multiple brokers in a client's connection string allows for failover when a specific broker is offline for an update. For information about how to get a connection string with multiple brokers, see [Get the bootstrap brokers for an Amazon MSK cluster](#).
- Run performance tests to verify that your client configurations allow you to meet your performance objectives.

Server-side considerations

Right-size your cluster: Number of partitions per Standard broker

The following table shows the recommended number of partitions (including leader and follower replicas) per Standard broker. The recommended number of partitions are not enforced and are a best practice for scenarios where you are sending traffic across all provisioned topic partitions.

Broker size	Recommended number of partitions (including leader and follower replicas) per broker	Maximum number of partitions that support update operations
<code>kafka.t3.small</code>	300	300
<code>kafka.m5.large</code> or <code>kafka.m5.xlarge</code>	1000	1500
<code>kafka.m5.2xlarge</code>	2000	3000
<code>kafka.m5.4xlarge</code> , <code>kafka.m5.8xlarge</code> , <code>kafka.m5.12xlarge</code> , <code>kafka.m5.16xlarge</code> , or <code>kafka.m5.24xlarge</code>	4000	6000
<code>kafka.m7g.large</code> or <code>kafka.m7g.xlarge</code>	1000	1500
<code>kafka.m7g.2xlarge</code>	2000	3000
<code>kafka.m7g.4xlarge</code> , <code>kafka.m7g.8xlarge</code> , <code>kafka.m7g.12xlarge</code> , or <code>kafka.m7g.16xlarge</code>	4000	6000

If you have high partition, low throughput use cases where you have higher partition counts, but you are not sending traffic across all partitions, you can pack more partitions per broker, as long

as you have performed sufficient testing and performance testing to validate that your cluster remains healthy with the higher partition count. If the number of partitions per broker exceeds the maximum allowed value and your cluster becomes overloaded, you will be prevented from performing the following operations:

- Update the cluster configuration
- Update the cluster to a smaller broker size
- Associate an AWS Secrets Manager secret with a cluster that has SASL/SCRAM authentication

A high number of partitions can also result in missing Kafka metrics on CloudWatch and on Prometheus scraping.

For guidance on choosing the number of partitions, see [Apache Kafka Supports 200K Partitions Per Cluster](#). We also recommend that you perform your own testing to determine the right size for your brokers. For more information about the different broker sizes, see [the section called “Broker types”](#).

Right-size your cluster: Number of Standard brokers per cluster

To determine the right number of Standard brokers for your MSK Provisioned cluster and understand costs, see the [MSK Sizing and Pricing](#) spreadsheet. This spreadsheet provides an estimate for sizing an MSK Provisioned cluster and the associated costs of Amazon MSK compared to a similar, self-managed, EC2-based Apache Kafka cluster. For more information about the input parameters in the spreadsheet, hover over the parameter descriptions. Estimates provided by this sheet are conservative and provide a starting point for a new MSK Provisioned cluster. Cluster performance, size, and costs are dependent on your use case and we recommend that you verify them with actual testing.

To understand how the underlying infrastructure affects Apache Kafka performance, see [Best practices for right-sizing your Apache Kafka clusters to optimize performance and cost](#) in the AWS Big Data Blog. The blog post provides information about how to size your clusters to meet your throughput, availability, and latency requirements. It also provides answers to questions, such as when you should scale *up* versus scale *out*, and guidance about how to continuously verify the size of your production clusters. For information about tiered storage based clusters, see [Best practices for running production workloads using Amazon MSK tiered storage](#).

Optimize cluster throughput for m5.4xl, m7g.4xl or larger instances

When using m5.4xl, m7g.4xl, or larger instances, you can optimize the MSK Provisioned cluster throughput by tuning the `num.io.threads` and `num.network.threads` configurations.

`Num.io.threads` is the number of threads that a Standard broker uses for processing requests. Adding more threads, up to the number of CPU cores supported for the instance size, can help improve cluster throughput.

`Num.network.threads` is the number of threads the Standard broker uses for receiving all incoming requests and returning responses. Network threads place incoming requests on a request queue for processing by `io.threads`. Setting `num.network.threads` to half the number of CPU cores supported for the instance size allows for full usage of the new instance size.

Important

Do not increase `num.network.threads` without first increasing `num.io.threads` as this can lead to congestion related to queue saturation.

Recommended settings

Instance size	Recommended value for <code>num.io.threads</code>	Recommended value for <code>num.network.threads</code>
m5.4xl	16	8
m5.8xl	32	16
m5.12xl	48	24
m5.16xl	64	32
m5.24xl	96	48
m7g.4xlarge	16	8
m7g.8xlarge	32	16
m7g.12xlarge	48	24

Instance size	Recommended value for num.io.threads	Recommended value for num.network.threads
m7g.16xlarge	64	32

Use latest Kafka AdminClient to avoid topic ID mismatch issue

The ID of a topic is lost (Error: does not match the topic Id for partition) when you use a Kafka AdminClient version lower than 2.8.0 with the flag `--zookeeper` to increase or reassign topic partitions for an MSK Provisioned cluster using Kafka version 2.8.0 or higher. Note that the `--zookeeper` flag is deprecated in Kafka 2.5 and is removed starting with Kafka 3.0. See [Upgrading to 2.5.0 from any version 0.8.x through 2.4.x](#).

To prevent topic ID mismatch, use a Kafka client version 2.8.0 or higher for Kafka admin operations. Alternatively, clients 2.5 and higher can use the `--bootstrap-servers` flag instead of the `--zookeeper` flag.

Build highly available clusters

Use the following recommendations so that your MSK Provisioned clusters can be highly available during an update (such as when you're updating the broker size or Apache Kafka version, for example) or when Amazon MSK is replacing a broker.

- Set up a three-AZ cluster.
- Ensure that the replication factor (RF) is at least 3. Note that a RF of 1 can lead to offline partitions during a rolling update; and a RF of 2 may lead to data loss.
- Set minimum in-sync replicas (minISR) to at most RF - 1. A minISR that is equal to the RF can prevent producing to the cluster during a rolling update. A minISR of 2 allows three-way replicated topics to be available when one replica is offline.

Monitor CPU usage

Amazon MSK strongly recommends that you maintain the CPU utilization for your brokers (defined as `CPU User + CPU System`) under 60%. This ensures that your cluster retains sufficient CPU headroom to handle operational events, such as broker failures, patching, and rolling upgrades.

Apache Kafka can redistribute CPU load across brokers in the cluster when necessary. For example, when Amazon MSK detects and recovers from a broker fault, it performs automatic maintenance,

such as patching. Similarly, when a user requests a broker-size change or version upgrade, Amazon MSK initiates rolling workflows that take one broker offline at a time. When brokers with lead partitions go offline, Apache Kafka reassigns partition leadership to redistribute work to other brokers in the cluster. By following this best practice, you ensure sufficient CPU headroom to tolerate these operational events.

Note

While monitoring CPU utilization, be aware that total CPU usage includes more than CPU User and CPU System. Other categories, such as `iowait`, `irq`, `softirq`, and `steal`, also contribute to overall CPU activity. Consequently, CPU Idle *isn't always equal* to $100\% - \text{CPU User} - \text{CPU System}$.

You can use [Amazon CloudWatch metric math](#) to create a composite metric ($\text{CPU User} + \text{CPU System}$), and set an alarm to trigger when the average usage exceeds 60%. When triggered, consider scaling the cluster using one of the following options:

- Option 1 (recommended): [Update your broker size](#) to the next larger size. For example, if the current size is `kafka.m5.large`, update the cluster to use `kafka.m5.xlarge`. Keep in mind that when you update the broker size in the cluster, Amazon MSK takes brokers offline in a rolling fashion and temporarily reassigns partition leadership to other brokers. A size update typically takes 10-15 minutes per broker.
- Option 2: If there are topics with all messages ingested from producers that use round-robin writes (in other words, messages aren't keyed and ordering isn't important to consumers), [expand your cluster](#) by adding brokers. Also add partitions to existing topics with the highest throughput. Next, use `kafka-topics.sh --describe` to ensure that newly added partitions are assigned to the new brokers. The main benefit of this option compared to the previous one is that you can manage resources and costs more granularly. Additionally, you can use this option if CPU load significantly exceeds 60% because this form of scaling doesn't typically result in increased load on existing brokers.
- Option 3: Expand your MSK Provisioned cluster by adding brokers, then reassign existing partitions by using the partition reassignment tool named `kafka-reassign-partitions.sh`. However, if you use this option, the cluster will need to spend resources to replicate data from broker to broker after partitions are reassigned. Compared to the two previous options, this can significantly increase the load on the cluster at first. As a result, Amazon MSK doesn't recommend using this option when CPU utilization is above 70% because replication causes

additional CPU load and network traffic. Amazon MSK only recommends using this option if the two previous options aren't feasible.

Other recommendations:

- Monitor total CPU utilization per broker as a proxy for load distribution. If brokers have consistently uneven CPU utilization it might be a sign that load isn't evenly distributed within the cluster. We recommend using [Cruise Control](#) to continuously manage load distribution via partition assignment.
- Monitor produce and consume latency. Produce and consume latency can increase linearly with CPU utilization.
- **JMX scrape interval:** If you enable open monitoring with the [Prometheus feature](#), it is recommended that you use a 60 second or higher scrape interval (scrape_interval: 60s) for your Prometheus host configuration (prometheus.yml). Lowering the scrape interval can lead to high CPU usage on your cluster.

Monitor disk space

To avoid running out of disk space for messages, create a CloudWatch alarm that watches the `KafkaDataLogsDiskUsed` metric. When the value of this metric reaches or exceeds 85%, perform one or more of the following actions:

- Use [the section called “Automatic scaling for clusters”](#). You can also manually increase broker storage as described in [the section called “Manual scaling”](#).
- Reduce the message retention period or log size. For information on how to do that, see [the section called “Adjust data retention parameters”](#).
- Delete unused topics.

For information on how to set up and use alarms, see [Using Amazon CloudWatch Alarms](#). For a full list of Amazon MSK metrics, see [the section called “Monitor a cluster”](#).

Adjust data retention parameters

Consuming messages doesn't remove them from the log. To free up disk space regularly, you can explicitly specify a retention time period, which is how long messages stay in the log. You can also specify a retention log size. When either the retention time period or the retention log size are reached, Apache Kafka starts removing inactive segments from the log.

To specify a retention policy at the cluster level, set one or more of the following parameters: `log.retention.hours`, `log.retention.minutes`, `log.retention.ms`, or `log.retention.bytes`. For more information, see [the section called “Custom Amazon MSK configurations”](#).

You can also specify retention parameters at the topic level:

- To specify a retention time period per topic, use the following command.

```
kafka-configs.sh --bootstrap-server $bs --alter --entity-type topics --entity-name TopicName --add-config retention.ms=DesiredRetentionTimePeriod
```

- To specify a retention log size per topic, use the following command.

```
kafka-configs.sh --bootstrap-server $bs --alter --entity-type topics --entity-name TopicName --add-config retention.bytes=DesiredRetentionLogSize
```

The retention parameters that you specify at the topic level take precedence over cluster-level parameters.

Speeding up log recovery after unclean shutdown

After an unclean shutdown, a broker can take a while to restart as it does log recovery. By default, Kafka only uses a single thread per log directory to perform this recovery. For example, if you have thousands of partitions, log recovery can take hours to complete. To speed up log recovery, it's recommended to increase the number of threads using configuration property [num.recovery.threads.per.data.dir](#). You can set it to the number of CPU cores.

Monitor Apache Kafka memory

We recommend that you monitor the memory that Apache Kafka uses. Otherwise, the cluster may become unavailable.

To determine how much memory Apache Kafka uses, you can monitor the `HeapMemoryAfterGC` metric. `HeapMemoryAfterGC` is the percentage of total heap memory that is in use after garbage collection. We recommend that you create a CloudWatch alarm that takes action when `HeapMemoryAfterGC` increases above 60%.

The steps that you can take to decrease memory usage vary. They depend on the way that you configure Apache Kafka. For example, if you use transactional message delivery, you can decrease

the `transactional.id.expiration.ms` value in your Apache Kafka configuration from 604800000 ms to 86400000 ms (from 7 days to 1 day). This decreases the memory footprint of each transaction.

Don't add non-MSK brokers

For ZooKeeper-based MSK Provisioned clusters, if you use Apache ZooKeeper commands to add brokers, these brokers don't get added to your MSK Provisioned cluster, and your Apache ZooKeeper will contain incorrect information about the cluster. This might result in data loss. For supported MSK Provisioned cluster operations, see [the section called "Key features and concepts"](#).

Enable in-transit encryption

For information about encryption in transit and how to enable it, see [the section called "Amazon MSK encryption in transit"](#).

Reassign partitions

To move partitions to different brokers on the same MSK Provisioned cluster, you can use the partition reassignment tool named `kafka-reassign-partitions.sh`. We recommend that you don't reassign more than 10 partitions in a single `kafka-reassign-partitions` call for safe operations. For example, after you add new brokers to expand a cluster or to move partitions in order to removing brokers, you can rebalance that cluster by reassigning partitions to the new brokers. For information about how to add brokers to an MSK Provisioned cluster, see [the section called "Expand a cluster"](#). For information about how to remove brokers from an MSK Provisioned cluster, see [the section called "Remove a broker"](#). For information about the partition reassignment tool, see [Expanding your cluster](#) in the Apache Kafka documentation.

Best practices for Express brokers

This topic outlines some best practices to follow when using Express brokers. Express brokers come pre-configured for high availability and durability. Your data is distributed across three availability zones by default, replication is always set to 3 and the minimum in-sync replica is always set to 2. However, there are still a few factors to consider in order to optimize your cluster's reliability and performance.

Client-side considerations

Your application's availability and performance depends not only on server-side settings but on client settings as well.

- Configure your clients for high availability. In a distributed system like Apache Kafka, ensuring high availability is crucial for maintaining a reliable and fault-tolerant messaging infrastructure. Brokers will go offline for both planned and unplanned events, for example upgrades, patching, hardware failure, and network issues. A Kafka cluster is tolerant of an offline broker, therefore Kafka clients must also handle broker fail-over gracefully. See the full details in [best practice recommendations for Apache Kafka clients](#).
- Run performance tests to verify that your client configurations allow you to meet your performance objectives even when we restart brokers under peak load. You can reboot brokers in your cluster from the MSK console or using the MSK APIs.

Server-side considerations

Right-size your cluster: Number of brokers per cluster

Choosing the number of brokers for your Express-based cluster is easy. Each Express broker comes with a defined throughput capacity for ingress and egress. You should use this throughput capacity as the primary means for sizing your cluster (and then consider other factors such as partition and connection count, discussed below).

For example, if your streaming application needs 45 MBps of data ingress (write) and 90 MBps data egress (read) capacity, you can simply use 3 `express.m7g.large` brokers to meet your throughput needs. Each `express.m7g.large` broker will handle 15 MBps of ingress and 30 MBps egress. See the following table for our recommended throughput limits for each Express broker size. If your throughput exceeds the recommended limits, you may experience degraded performance and you should reduce your traffic or scale your cluster. If your throughput exceeds the recommended limits and reaches the per broker quota, MSK will throttle your client traffic to prevent further overload.

You can also use our [MSK Sizing and Pricing](#) spreadsheet to evaluate multiple scenarios and consider other factors, such as partition count.

Recommended maximum throughput per broker

Instance size	Ingress (MBps)	Egress (MBps)
<code>express.m7g.large</code>	15.6	31.2
<code>express.m7g.xlarge</code>	31.2	62.5
<code>express.m7g.2xlarge</code>	62.5	125.0

Instance size	Ingress (MBps)	Egress (MBps)
express.m7g.4xlarge	124.9	249.8
express.m7g.8xlarge	250.0	500.0
express.m7g.12xlarge	375.0	750.0
express.m7g.16xlarge	500.0	1000.0

Monitor CPU usage

We recommend that you maintain the total CPU utilization for your brokers (defined as CPU User + CPU System) under 60%. When you have at least 40% of your cluster's total CPU available, Apache Kafka can redistribute CPU load across brokers in the cluster when necessary. This may be required due to planned or unplanned events. An example of a planned event is a cluster version upgrade during which MSK updates brokers in a cluster by restarting them one at a time. An example of an unplanned event is a hardware failure in a broker or, in the worst case, an AZ failure where all brokers in an AZ are affected. When brokers with partition lead replicas go offline, Apache Kafka reassigns partition leadership to redistribute work to other brokers in the cluster. By following this best practice, you can ensure you have enough CPU headroom in your cluster to tolerate operational events like these.

You can use [Using math expressions with CloudWatch metrics](#) in the *Amazon CloudWatch User Guide* to create a composite metric that is CPU User + CPU System. Set an alarm that gets triggered when the composite metric reaches an average CPU utilization of 60%. When this alarm is triggered, scale the cluster using one of the following options:

- Option 1: [Update your broker size](#) to the next larger size. Keep in mind that when you update the broker size in the cluster, Amazon MSK takes brokers offline in a rolling fashion and temporarily reassigns partition leadership to other brokers.
- Option 2: [Expand your cluster by adding brokers](#), then reassigning existing partitions using the partition reassignment tool named `kafka-reassign-partitions.sh`.

Other recommendations

- Monitor total CPU utilization per broker as a proxy for load distribution. If brokers have consistently uneven CPU utilization, it might be a sign that load isn't evenly distributed within

the cluster. We recommend using [Cruise Control](#) to continuously manage load distribution via partition assignment.

- Monitor produce and consume latency. Produce and consume latency can increase linearly with CPU utilization.
- JMX scrape interval: If you enable open monitoring with the Prometheus feature, it is recommended that you use a 60 second or higher scrape interval (`scrape_interval: 60s`) for your Prometheus host configuration (`prometheus.yml`). Lowering the scrape interval can lead to high CPU usage on your cluster.

Right-size your cluster: Number of partitions per Express broker

The following table shows the recommended number of partitions (including leader and follower replicas) per Express broker. The recommended number of partitions are not enforced and are a best practice for scenarios where you are sending traffic across all provisioned topic partitions.

Broker size	Recommended number of partitions (including leader and follower replicas) per broker	Maximum number of partitions that support update operations
express.m7g.large	1000	1500
express.m7g.xlarge		
express.m7g.2xlarge	2000	3000
express.m7g.4xlarge	4000	6000
express.m7g.8xlarge		
express.m7g.12xlarge		
express.m7g.16xlarge		

If you have high partition, low throughput use cases where you have higher partition counts, but you are not sending traffic across all partitions, you can pack more partitions per broker, as long as you have performed sufficient testing and performance testing to validate that your cluster remains healthy with the higher partition count. If the number of partitions per broker exceeds

the maximum allowed value and your cluster becomes overloaded, you will be prevented from performing the following operations:

- Update the cluster configuration
- Update the cluster to a smaller broker size
- Associate an AWS Secrets Manager secret with a cluster that has SASL/SCRAM authentication

A cluster overloaded with a high number of partitions can also result in missing Kafka metrics on CloudWatch and on Prometheus scraping.

For guidance on choosing the number of partitions, see [Apache Kafka Supports 200K Partitions Per Cluster](#). We also recommend that you perform your own testing to determine the right size for your brokers. For more information about the different broker sizes, see [Amazon MSK broker sizes](#).

Monitor connection count

The client connections to your brokers consume system resources such as memory and CPU. Depending on your authentication mechanism, you should monitor to ensure you are within the applicable limits. To handle retries on failed connections, you can set the `reconnect.backoff.ms` configuration parameter on the client side. For example, if you want a client to retry connections after 1 second, set `reconnect.backoff.ms` to `1000`. For more information about configuring retries, see [Apache Kafka documentation](#).

Dimension	Quota
Maximum TCP connections per broker (IAM Access control)	3000
Maximum TCP connections per broker (IAM)	100 per second
Maximum TCP connections per broker (non-IAM)	MSK does not enforce connection limits for non-IAM auth. You should however monitor other metrics such as CPU and memory usage to ensure you do not overload your cluster because of excessive connections.

Reassign partitions

To move partitions to different brokers on the same MSK Provisioned cluster, you can use the partition reassignment tool named `kafka-reassign-partitions.sh`. We recommend that you don't reassign more than 20 partitions in a single `kafka-reassign-partitions` call for safe operations. For example, after you add new brokers to expand a cluster or to move partitions in order to removing brokers, you can rebalance that cluster by reassigning partitions to the new brokers. For information about how to add brokers to an MSK Provisioned cluster, see [the section called “Expand a cluster”](#). For information about how to remove brokers from an MSK Provisioned cluster, see [the section called “Remove a broker”](#). For information about the partition reassignment tool, see [Expanding your cluster](#) in the Apache Kafka documentation.

Best practices for Apache Kafka clients

When working with Apache Kafka and Amazon MSK, it's important to correctly configure both the client and server for optimal performance and reliability. This guide provides recommendations of best-practice client-side configuration for Amazon MSK.

For information about Amazon MSK Replicator best practices, see [Best practices for using MSK Replicator](#). For Standard and Express broker best practices, see [Best practices for Standard and Express brokers](#).

Topics

- [Apache Kafka client availability](#)
- [Apache Kafka client performance](#)
- [Kafka client monitoring](#)

Apache Kafka client availability

In a distributed system like Apache Kafka, ensuring high availability is crucial for maintaining a reliable and fault-tolerant messaging infrastructure. Brokers will go offline for both planned and unplanned events, such as upgrades, patching, hardware failure, and network issues. A Kafka cluster is tolerant of an offline broker, therefore Kafka clients must also handle broker fail-over gracefully. To ensure high availability of Kafka clients, we recommend these best practices.

Producer availability

- Set `retries` to instruct the producer to retry sending failed messages during broker fail over. We recommend a value of integer max or similar high value for most use cases. Failure to do so will break Kafka's high availability.
- Set `delivery.timeout.ms` to specify the upper bound for the total time between sending a message and receiving an acknowledgement from the broker. This should reflect the business requirements of how long a message is valid. Set the time limit high enough to allow for enough retries to complete the failover operation. We recommend a value of 60 seconds or higher for most use cases.
- Set `request.timeout.ms` to the maximum a single request should wait before a resend is attempted. We recommend a value of 10 seconds or higher for most use cases.
- Set `retry.backoff.ms` to configure the delay between retries to avoid retry storms and availability impact. We recommend a minimum value of 200ms for most use cases.
- Set `acks=all` to configure high durability; this should be in line with a server-side config of `RF=3` and `min.isr=2` to ensure all partitions in ISR acknowledge the write. During a single broker offline, this is the `min.isr`, that is 2.

Consumer availability

- Set `auto.offset.reset` to `latest` initially for new or recreated consumer groups. This avoids the risk of adding cluster load by consuming the entire topic.
- Set `auto.commit.interval.ms` when using `enable.auto.commit`. We recommend a minimum value of 5 seconds for most use cases to avoid the risk of additional load.
- Implement exception handling within the consumer's message processing code to handle transient errors, for example, circuit breaker or a sleep with exponential back-off. Failure to do so can result in application crashes, which can cause excessive rebalancing.
- Set `isolation.level` to control how to read transactional messages:

We recommend always setting `read_uncommitted` implicitly by default. This is missing from some client implementations.

We recommend a value of `read_uncommitted` when using tiered storage.

- Set `client.rack` to use a nearest replica read. We recommend setting to the `az id` to minimize network traffic costs and latency. See [Reduce network traffic costs of your Amazon MSK consumers with rack awareness](#).

Consumer rebalances

- Set `session.timeout.ms` to a value larger than the startup time for an application, including any startup jitter implemented. We recommend a value of 60 seconds for most use cases.
- Set `heartbeat.interval.ms` to fine-tune how the group coordinator views a consumer as healthy. We recommend a value of 10 seconds for most use cases.
- Set a shutdown hook in your application to cleanly close the consumer on SIGTERM, rather than relying on session timeouts to identify when a consumer leaves a group. Kstream applications can set `internal.leave.group.on.close` to a value of `true`.
- Set `group.instance.id` to a distinct value within the consumer group. Ideally a host name, task-id, or pod-id. We recommend always setting this for more deterministic behaviors and better client/server log correlation during troubleshooting.
- Set `group.initial.rebalance.delay.ms` to a value in line with an average deployment time. This stops continual rebalances during deployment.
- Set `partition.assignment.strategy` to use sticky assignors. We recommend either `StickyAssignor` or `CooperativeStickyAssignor`.

Apache Kafka client performance

To ensure high performance of Kafka clients, we recommend these best practices.

Producer performance

- Set `linger.ms` to control the amount of time a producer waits for a batch to fill. Smaller batches are computationally expensive for Kafka as they translate to more threads and I/O operations at once. We recommend the following values.

A minimum value of 5ms for all use cases inc low latency.

We recommend a higher value of 25ms, for most use cases.

We recommend against ever using a value of zero in low latency use cases. (A value of zero typically causes latency irrespective because of the IO overhead).

- Set `batch.size` to control the batch size sent to the cluster. We recommend increasing this to a value of 64KB or 128KB.
- Set `buffer.memory` when using larger batch sizes. We recommend a value of 64MB for most use cases.

- Set `send.buffer.bytes` to control the TCP buffer used to receive bytes. We recommend a value of -1 to let the OS manage this buffer when running a producer on a high latency network.
- Set `compression.type` to control the compression of batches. We recommend either lz4 or zstd running a producer on a high latency network.

Consumer performance

- Set `fetch.min.bytes` to control the minimum fetch size to be valid to reduce the number of fetches and cluster load.

We recommend a minimum value of 32 bytes for all use cases.

We recommend a higher value of 128 bytes for most use cases.

- Set `fetch.max.wait.ms` to determine how long your consumer will wait before `fetch.min.bytes` is ignored. We recommend a value of 1000ms for most use cases.
- We recommend the number of consumers be at least equal to the number of partitions for better parallelism and resiliency. In some situations, you may choose to have fewer consumers than the number of partitions for low throughput topics.
- Set `receive.buffer.bytes` to control the TCP buffer used to receive bytes. We recommend a value of -1 to let the OS manage this buffer when running a consumer on a high latency network.

Client connections

Connections lifecycle has a computational and memory cost on a Kafka cluster. Too many connections created at once causes load that may impact the availability of a Kafka cluster. This availability impact can often lead to applications creating even more connections, thus causing a cascading failure, resulting in a full outage. A high number of connections can be achieved when created at a reasonable rate.

We recommend the following mitigations to manage high connection creation rates:

- Ensure your application deployment mechanism does not restart all producers/consumers at once, but preferably in smaller batches.
- At the application layer the developer should ensure that a random jitter (random sleep) is performed before creating an admin client, producer client, or consumer client.

- At SIGTERM, when closing the connection, a random sleep should be executed to ensure not all Kafka clients are closed at the same time. The random sleep should be within the timeout before SIGKILL occurs.

Example Example A (Java)

```
sleepInSeconds(randomNumberBetweenOneAndX);  
this.kafkaProducer = new KafkaProducer<>(this.props);
```

Example Example B (Java)

```
Runtime.getRuntime().addShutdownHook(new Thread(() -> {  
    sleepInSeconds(randomNumberBetweenOneAndTwentyFive);  
    kafkaProducer.close(Duration.ofSeconds(5));  
}));
```

- At the application layer, the developer should ensure that clients are created only once per application in a singleton pattern. For example, when using lambda, the client should be created in global scope, and not in the method handler.
- We recommend connection count is monitored with a goal of being stable. Connection creation/close/shift is normal during deployments and broker failover.

Kafka client monitoring

Monitoring Kafka clients is crucial for maintaining the health and efficiency of your Kafka ecosystem. Whether you're a Kafka administrator, developer, or operations team member, enabling client-side metrics is critical for understanding business impact during planned and unplanned events.

We recommend monitoring the following client-side metrics using your preferred metric capture mechanism.

When raising support tickets with AWS, include any abnormal values observed during the incident. Also include a sample of the client application logs detailing errors (not warnings).

Producer metrics

- byte-rate
- record-send-rate

- records-per-request-avg
- acks-latency-avg
- request-latency-avg
- request-latency-max
- record-error-rate
- record-retry-rate
- error-rate

 **Note**

Transient errors with retries are not a cause for concern, as this is part of Kafka's protocol for handling transient issues such as leader fail-over or network retransmits. `record-send-rate` will confirm whether producers are still proceeding with retries.

Consumer metrics

- records-consumed-rate
- bytes-consumed-rate
- fetch-rate
- records-lag-max
- record-error-rate
- fetch-error-rate
- poll-rate
- rebalance-latency-avg
- commit-rate

 **Note**

High fetch rates and commit-rates will cause unnecessary load on the cluster. It is optimal to perform requests in larger batches.

Common metrics

- connection-close-rate
- connection-creation-rate
- connection-count

Note

High connection creation/termination will cause unnecessary load on the cluster.

What is MSK Serverless?

Note

MSK Serverless is available in the US East (Ohio), US East (N. Virginia), US West (Oregon), Canada (Central), Asia Pacific (Mumbai), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), Asia Pacific (Seoul), Europe (Frankfurt), Europe (Stockholm), Europe (Ireland), Europe (Paris), and Europe (London) Regions.

MSK Serverless is a cluster type for Amazon MSK that makes it possible for you to run Apache Kafka without having to manage and scale cluster capacity. It automatically provisions and scales capacity while managing the partitions in your topic, so you can stream data without thinking about right-sizing or scaling clusters. MSK Serverless offers a throughput-based pricing model, so you pay only for what you use. Consider using a serverless cluster if your applications need on-demand streaming capacity that scales up and down automatically.

MSK Serverless is fully compatible with Apache Kafka, so you can use any compatible client applications to produce and consume data. It also integrates with the following services:

- AWS PrivateLink to provide private connectivity
- AWS Identity and Access Management (IAM) for authentication and authorization using Java and non-Java languages. For instruction on configuring clients for IAM, see [Configure clients for IAM access control](#).
- AWS Glue Schema Registry for schema management
- Amazon Managed Service for Apache Flink for Apache Flink-based stream processing
- AWS Lambda for event processing

Note

MSK Serverless requires IAM access control for all clusters. Apache Kafka access control lists (ACLs) are not supported. For more information, see [the section called “IAM access control”](#). For information about the service quota that apply to MSK Serverless, see [the section called “Quota for serverless clusters”](#).

To help you get started with serverless clusters, and to learn more about configuration and monitoring options for serverless clusters, see the following.

Topics

- [Use MSK Serverless clusters](#)
- [Configuration properties for MSK Serverless clusters](#)
- [Monitor MSK Serverless clusters](#)

Use MSK Serverless clusters

This tutorial shows you an example of how you can create an MSK Serverless cluster, create a client machine that can access it, and use the client to create topics on the cluster and to write data to those topics. This exercise doesn't represent all the options that you can choose when you create a serverless cluster. In different parts of this exercise, we choose default options for simplicity. This doesn't mean that they're the only options that work for setting up a serverless cluster. You can also use the AWS CLI or the Amazon MSK API. For more information, see the [Amazon MSK API Reference 2.0](#).

Topics

- [Create an MSK Serverless cluster](#)
- [Create an IAM role for topics on MSK Serverless cluster](#)
- [Create a client machine to access MSK Serverless cluster](#)
- [Create an Apache Kafka topic](#)
- [Produce and consume data in MSK Serverless](#)
- [Delete resources that you created for MSK Serverless](#)

Create an MSK Serverless cluster

In this step, you perform two tasks. First, you create an MSK Serverless cluster with default settings. Second, you gather information about the cluster. This is information that you need in later steps when you create a client that can send data to the cluster.

To create a serverless cluster

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home>.
2. Choose **Create cluster**.
3. For **Creation method**, leave the **Quick create** option selected. The **Quick create** option lets you create a serverless cluster with default settings.
4. For **Cluster name**, enter a descriptive name, such as **msk-serverless-tutorial-cluster**.
5. For **General cluster properties**, choose **Serverless** as the **Cluster type**. Use the default values for the remaining **General cluster** properties.
6. Note the table under **All cluster settings**. This table lists the default values for important settings such as networking and availability, and indicates whether you can change each setting after you create the cluster. To change a setting before you create the cluster, you should choose the **Custom create** option under **Creation method**.

Note

You can connect clients from up to five different VPCs with MSK Serverless clusters. To help client applications switch over to another Availability Zone in the event of an outage, you must specify at least two subnets in each VPC.

7. Choose **Create cluster**.

To gather information about the cluster

1. In the **Cluster summary** section, choose **View client information**. This button remains grayed out until Amazon MSK finishes creating the cluster. You might need to wait a few minutes until the button becomes active so you can use it.
2. Copy the string under the label **Endpoint**. This is your bootstrap server string.
3. Choose the **Properties** tab.
4. Under the **Networking settings** section, copy the IDs of the subnets and the security group and save them because you need this information later to create a client machine.
5. Choose any of the subnets. This opens the Amazon VPC Console. Find the ID of the Amazon VPC that is associated with the subnet. Save this Amazon VPC ID for later use.

Next Step

[Create an IAM role for topics on MSK Serverless cluster](#)

Create an IAM role for topics on MSK Serverless cluster

In this step, you perform two tasks. The first task is to create an IAM policy that grants access to create topics on the cluster and to send data to those topics. The second task is to create an IAM role and associate this policy with it. In a later step, we create a client machine that assumes this role and uses it to create a topic on the cluster and to send data to that topic.

To create an IAM policy that makes it possible to create topics and write to them

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. On the navigation pane, choose **Policies**.
3. Choose **Create Policy**.
4. Choose the **JSON** tab, then replace the JSON in the editor window with the following JSON.

In the following example, replace the following:

- *region* with the code of the AWS Region where you created your cluster.
- *Account-ID* with your AWS account ID.
- *msk-serverless-tutorial-cluster/c07c74ea-5146-4a03-add1-9baa787a5b14-s3* and *msk-serverless-tutorial-cluster* with your serverless cluster ID and topic name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:Connect",
        "kafka-cluster:DescribeCluster"
      ],
      "Resource": [
        "arn:aws:kafka:region:Account-ID:cluster/msk-serverless-tutorial-cluster/c07c74ea-5146-4a03-add1-9baa787a5b14-s3"
      ]
    }
  ]
}
```

```
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "kafka-cluster:CreateTopic",  
        "kafka-cluster:WriteData",  
        "kafka-cluster:DescribeTopic"  
      ],  
      "Resource": [  
        "arn:aws:kafka:region:Account-ID:topic/msk-serverless-tutorial-cluster/*"  
      ]  
    }  
  ]  
}
```

For instructions about how to write secure policies, see [the section called “IAM access control”](#).

5. Choose **Next: Tags**.
6. Choose **Next: Review**.
7. For the policy name, enter a descriptive name, such as **msk-serverless-tutorial-policy**.
8. Choose **Create policy**.

To create an IAM role and attach the policy to it

1. On the navigation pane, choose **Roles**.
2. Choose **Create role**.
3. Under **Common use cases**, choose **EC2**, then choose **Next: Permissions**.
4. In the search box, enter the name of the policy that you previously created for this tutorial. Then select the box to the left of the policy.
5. Choose **Next: Tags**.
6. Choose **Next: Review**.
7. For the role name, enter a descriptive name, such as **msk-serverless-tutorial-role**.
8. Choose **Create role**.

Next Step

[Create a client machine to access MSK Serverless cluster](#)

Create a client machine to access MSK Serverless cluster

In the step, you perform two tasks. The first task is to create an Amazon EC2 instance to use as an Apache Kafka client machine. The second task is to install Java and Apache Kafka tools on the machine.

To create a client machine

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Launch instance**.
3. Enter a descriptive **Name** for your client machine, such as **msk-serverless-tutorial-client**.
4. Leave the **Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type** selected for **Amazon Machine Image (AMI) type**.
5. Leave the **t2.micro** instance type selected.
6. Under **Key pair (login)**, choose **Create a new key pair**. Enter **MSKServerlessKeyPair** for **Key pair name**. Then choose **Download Key Pair**. Alternatively, you can use an existing key pair.
7. For **Network settings**, choose **Edit**.
8. Under **VPC**, enter the ID of the virtual private cloud (VPC) for your serverless cluster . This is the VPC based on the Amazon VPC service whose ID you saved after you created the cluster.
9. For **Subnet**, choose the subnet whose ID you saved after you created the cluster.
10. For **Firewall (security groups)**, select the security group associated with the cluster. This value works if that security group has an inbound rule that allows traffic from the security group to itself. With such a rule, members of the same security group can communicate with each other. For more information, see [Security group rules](#) in the Amazon VPC Developer Guide.
11. Expand the **Advanced details** section and choose the IAM role that you created in [Create an IAM role for topics on MSK Serverless cluster](#).
12. Choose **Launch**.
13. In the left navigation pane, choose **Instances**. Then choose the check box in the row that represents your newly created Amazon EC2 instance. From this point forward, we call this instance the *client machine*.
14. Choose **Connect** and follow the instructions to connect to the client machine.

To set up Apache Kafka client tools on the client machine

1. To install Java, run the following command on the client machine:

```
sudo yum -y install java-11
```

2. To get the Apache Kafka tools that we need to create topics and send data, run the following commands:

```
wget https://archive.apache.org/dist/kafka/2.8.1/kafka_2.12-2.8.1.tgz
```

```
tar -xzf kafka_2.12-2.8.1.tgz
```

Note

After extracting the Kafka archive, make sure that the scripts in the `bin` directory have proper execute permissions. To do this, run the following command.

```
chmod +x kafka_2.12-2.8.1/bin/.sh
```

3. Go to the `kafka_2.12-2.8.1/libs` directory, then run the following command to download the Amazon MSK IAM JAR file. The Amazon MSK IAM JAR makes it possible for the client machine to access the cluster.

```
wget https://github.com/aws/aws-msk-iam-auth/releases/download/v2.3.0/aws-msk-iam-auth-2.3.0-all.jar
```

Using this command, you can also [download other or newer versions](#) of Amazon MSK IAM JAR file.

4. Go to the `kafka_2.12-2.8.1/bin` directory. Copy the following property settings and paste them into a new file. Name the file `client.properties` and save it.

```
security.protocol=SASL_SSL
sasl.mechanism=AWS_MSK_IAM
sasl.jaas.config=software.amazon.msk.auth.iam.IAMLoginModule required;
sasl.client.callback.handler.class=software.amazon.msk.auth.iam.IAMClientCallbackHandler
```

Next Step

[Create an Apache Kafka topic](#)

Create an Apache Kafka topic

In this step, you use the previously created client machine to create a topic on the serverless cluster.

To create a topic and write data to it

1. In the following export command, replace *my-endpoint* with the bootstrap-server string you that you saved after you created the cluster. Then, go to the `kafka_2.12-2.8.1/bin` directory on the client machine and run the export command.

```
export BS=my-endpoint
```

2. Run the following command to create a topic called `msk-serverless-tutorial`.

```
<path-to-your-kafka-installation>/bin/kafka-topics.sh --bootstrap-server $BS  
--command-config client.properties --create --topic msk-serverless-tutorial --  
partitions 6
```

Next Step

[Produce and consume data in MSK Serverless](#)

Produce and consume data in MSK Serverless

In this step, you produce and consume data using the topic that you created in the previous step.

To produce and consume messages

1. Run the following command to create a console producer.

```
<path-to-your-kafka-installation>/bin/kafka-console-producer.sh --broker-list $BS  
--producer.config client.properties --topic msk-serverless-tutorial
```

2. Enter any message that you want, and press **Enter**. Repeat this step two or three times. Every time you enter a line and press **Enter**, that line is sent to your cluster as a separate message.

3. Keep the connection to the client machine open, and then open a second, separate connection to that machine in a new window.
4. Use your second connection to the client machine to create a console consumer with the following command. Replace *my-endpoint* with the bootstrap server string that you saved after you created the cluster.

```
<path-to-your-kafka-installation>/bin/kafka-console-consumer.sh --bootstrap-server my-endpoint --consumer.config client.properties --topic msk-serverless-tutorial --from-beginning
```

You start seeing the messages you entered earlier when you used the console producer command.

5. Enter more messages in the producer window, and watch them appear in the consumer window.

If you encounter classpath issues while running these commands, make sure that you're running them from the correct directory. Also, make sure that the AWS MSK IAM JAR is in the `libs` directory. Alternatively, you can run Kafka commands using the full Java command with explicit classpath, as shown in the following example.

```
java -cp "kafka_2.12-2.8.1/libs/*:kafka_2.12-2.8.1/libs/aws-msk-iam-auth-2.3.0-all.jar" org.apache.kafka.tools.ConsoleProducer --broker-list $BS --producer.config client.properties --topic msk-serverless-tutorial
```

Next Step

[Delete resources that you created for MSK Serverless](#)

Delete resources that you created for MSK Serverless

In this step, you delete the resources that you created in this tutorial.

To delete the cluster

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/home>.
2. In the list of clusters, choose the cluster that you created for this tutorial.
3. For **Actions**, choose **Delete cluster**.

4. Enter `delete` in the field, then choose **Delete**.

To stop the client machine

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the list of Amazon EC2 instances, choose the client machine that you created for this tutorial.
3. Choose **Instance state**, then choose **Terminate instance**.
4. Choose **Terminate**.

To delete the IAM policy and role

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. On the navigation pane, choose **Roles**.
3. In the search box, enter the name of the IAM role that you created for this tutorial.
4. Choose the role. Then choose **Delete role**, and confirm the deletion.
5. On the navigation pane, choose **Policies**.
6. In the search box, enter the name of the policy that you created for this tutorial.
7. Choose the policy to open its summary page. On the policy's **Summary** page, choose **Delete policy**.
8. Choose **Delete**.

Configuration properties for MSK Serverless clusters

Amazon MSK sets broker configuration properties for serverless clusters. You can't change these broker configuration property settings. However, you can set or modify the following topic-level configuration properties. All other topic-level configuration properties are not configurable.

Configuration property	Default	Editable	Maximum allowed value
cleanup.policy	Delete	Yes, but only at topic creation time	

Configuration property	Default	Editable	Maximum allowed value
compression.type	Producer	Yes	
max.message.bytes	1048588	Yes	8388608 (8MiB)
message.timestamp.difference.max.ms	long.max	Yes	
message.timestamp.type	CreateTime	Yes	
retention.bytes	250 GiB	Yes	Unlimited; set it to -1 for unlimited retention
retention.ms	7 days	Yes	Unlimited; set it to -1 for unlimited retention

To set or modify these topic-level configuration properties, you can use Apache Kafka command line tools. See [3.2 Topic-level Configs](#) in the official Apache Kafka documentation for more information and examples of how to set them.

Note

You can't modify the `segment.bytes` configuration for topics in MSK Serverless. However, a Kafka Streams application might attempt to create an internal topic with a `segment.bytes` configuration value, which is different from what MSK Serverless will allow. For information about configuring Kafka Streams with MSK Serverless, see [Using Kafka Streams with MSK Express brokers and MSK Serverless](#).

When using the Apache Kafka command line tools with Amazon MSK Serverless, make sure you completed steps 1-4 in the *To set up Apache Kafka client tools on the client machine* section of the [Amazon MSK Serverless Getting Started documentation](#). Additionally, you must include the `--command-config client.properties` parameter in your commands.

For example, the following command can be used to modify the `retention.bytes` topic configuration property to set unlimited retention:

```
<path-to-your-kafka-client-installation>/bin/kafka-configs.sh --bootstrap-server <bootstrap_server_string> --command-config client.properties --entity-type topics --entity-name <topic_name> --alter --add-config retention.bytes=-1
```

In this example, replace `<bootstrap_server_string>` with the bootstrap server endpoint for your Amazon MSK Serverless cluster, and `<topic_name>` with the name of the topic you want to modify.

The `--command-config client.properties` parameter ensures that the Kafka command line tool uses the appropriate configuration settings to communicate with your Amazon MSK Serverless cluster.

Monitor MSK Serverless clusters

Amazon MSK integrates with Amazon CloudWatch so that you can collect, view, and analyze metrics for your MSK Serverless cluster. The metrics shown in the following table are available for all serverless clusters. As these metrics are published as individual data points for each partition in the topic, we recommend viewing them as a 'SUM' statistic to get the topic-level view.

Amazon MSK publishes `PerSec` metrics to CloudWatch at a frequency of once per minute. This means that the 'SUM' statistic for a one-minute period accurately represents per-second data for `PerSec` metrics. To collect per-second data for a period of longer than one minute, use the following CloudWatch math expression: `m1 * 60/PERIOD(m1)`.

Metrics available at the DEFAULT monitoring level

Name	When visible	Dimensions	Description
BytesInPerSec	After a producer writes to a topic	Cluster Name, Topic	The number of bytes per second received from clients. This metric is available for each topic.
BytesOutPerSec	After a consumer group	Cluster Name, Topic	The number of bytes per second sent to clients. This metric is available for each topic.

Name	When visible	Dimensions	Description
	consumes from a topic		
FetchMessageConversionsPerSec	After a consumer group consumes from a topic	Cluster Name, Topic	The number of fetch message conversions per second for the topic.
EstimatedMaxTimeLag	After a consumer group consumes from a topic	Cluster Name, Consumer Group, Topic	A time estimate of the MaxOffsetLag metric.
MaxOffsetLag	After a consumer group consumes from a topic	Cluster Name, Consumer Group, Topic	The maximum offset lag across all partitions in a topic.
MessagesInPerSec	After a producer writes to a topic	Cluster Name, Topic	The number of incoming messages per second for the topic.
ProduceMessageConversionsPerSec	After a producer writes to a topic	Cluster Name, Topic	The number of produce message conversions per second for the topic.
SumOffsetLag	After a consumer group consumes from a topic	Cluster Name, Consumer Group, Topic	The aggregated offset lag for all the partitions in a topic.

To view MSK Serverless metrics

1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, under **Metrics**, choose **All metrics**.
3. In the metrics search for the term **kafka**.
4. Choose **AWS/Kafka / Cluster Name, Topic** or **AWS/Kafka / Cluster Name, Consumer Group, Topic** to see different metrics.

Understand MSK Connect

MSK Connect is a feature of Amazon MSK that makes it easy for developers to stream data to and from their Apache Kafka clusters. MSK Connect uses Kafka Connect versions 2.7.1 or 3.7.x, which are open-source frameworks for connecting Apache Kafka clusters with external systems such as databases, search indexes, and file systems. With MSK Connect, you can deploy fully managed connectors built for Kafka Connect that move data into or pull data from popular data stores like Amazon S3 and Amazon OpenSearch Service. You can deploy connectors developed by 3rd parties like Debezium for streaming change logs from databases into an Apache Kafka cluster, or deploy an existing connector with no code changes. Connectors automatically scale to adjust for changes in load and you pay only for the resources that you use.

Use source connectors to import data from external systems into your topics. With sink connectors, you can export data from your topics to external systems.

MSK Connect supports connectors for any Apache Kafka cluster with connectivity to an Amazon VPC, whether it is an MSK cluster or an independently hosted Apache Kafka cluster.

MSK Connect continuously monitors connector health and delivery state, patches and manages the underlying hardware, and autoscales the connectors to match changes in throughput.

To get started using MSK Connect, see [the section called “Getting started”](#).

To learn about the AWS resources that you can create with MSK Connect, see [the section called “Understand connectors”](#), [the section called “Create custom plugins”](#), and [the section called “Understand MSK Connect workers”](#).

For information about the MSK Connect API, see the [Amazon MSK Connect API Reference](#).

Benefits of using Amazon MSK Connect

Apache Kafka is one of the most widely adopted open source streaming platforms for ingesting and processing real-time data streams. With Apache Kafka, you can decouple and independently scale your data-producing and data-consuming applications.

Kafka Connect is an important component of building and running streaming applications with Apache Kafka. Kafka Connect provides a standardized way of moving data between Kafka and

external systems. Kafka Connect is highly scalable and can handle large volumes of data. Kafka Connect provides a powerful set of API operations and tools for configuring, deploying, and monitoring connectors that move data between Kafka topics and external systems. You can use these tools to customize and extend the functionality of Kafka Connect to meet the specific needs of your streaming application.

You might encounter challenges when you're operating Apache Kafka Connect clusters on their own, or when you're trying to migrate open source Apache Kafka Connect applications to AWS. These challenges include time required to setup infrastructure and deploying applications, engineering obstacles when setting up self-managed Apache Kafka Connect clusters, and administrative operational overhead.

To address these challenges, we recommend using Amazon Managed Streaming for Apache Kafka Connect (Amazon MSK Connect) to migrate your open source Apache Kafka Connect applications to AWS. Amazon MSK Connect simplifies using Kafka Connect to stream data to and from between Apache Kafka clusters and external systems, such as databases, search indexes, and file systems.

Here are some of the benefits to migrating to Amazon MSK Connect:

- **Elimination of operational overhead** — Amazon MSK Connect takes away the operational burden associated with patching, provisioning, and scaling of Apache Kafka Connect clusters. Amazon MSK Connect continuously monitors the health of your Connect clusters and automates patching and version upgrades without causing any disruptions to your workloads.
- **Automatic restarting of Connect tasks** — Amazon MSK Connect can automatically recover failed tasks to reduce production disruptions. Task failures can be caused by temporary errors, such as breaching the TCP connection limit for Kafka, and task rebalancing when new workers join the consumer group for sink connectors.
- **Automatic horizontal and vertical scaling** — Amazon MSK Connect enables the connector application to automatically scale to support higher throughputs. Amazon MSK Connect manages scaling for you. You only need to specify the number of workers in the auto scaling group and the utilization thresholds. You can use the Amazon MSK Connect `UpdateConnector` API operation to vertically scale up or scale down the vCPUs between 1 and 8 vCPUs for supporting variable throughput.
- **Private network connectivity** — Amazon MSK Connect privately connects to source and sink systems by using AWS PrivateLink and private DNS names.

Getting started with MSK Connect

This is a step-by-step tutorial that uses the AWS Management Console to create an MSK cluster and a sink connector that sends data from the cluster to an S3 bucket.

Topics

- [Set up resources required for MSK Connect](#)
- [Create custom plugin](#)
- [Create client machine and Apache Kafka topic](#)
- [Create connector](#)
- [Send data to the MSK cluster](#)

Set up resources required for MSK Connect

In this step you create the following resources that you need for this getting-started scenario:

- An Amazon S3 bucket to serve as the destination that receives data from the connector.
- An MSK cluster to which you will send data. The connector will then read the data from this cluster and send it to the destination S3 bucket.
- An IAM policy that contains the permissions to write to the destination S3 bucket.
- An IAM role that allows the connector to write to the destination S3 bucket. You'll add the IAM policy that you create to this role.
- An Amazon VPC endpoint to make it possible to send data from the Amazon VPC that has the cluster and the connector to Amazon S3.

To create the S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose **Create bucket**.
3. For the name of the bucket, enter a descriptive name such as `amzn-s3-demo-bucket-mkc-tutorial`.
4. Scroll down and choose **Create bucket**.
5. In the list of buckets, choose the newly created bucket.

6. Choose **Create folder**.
7. Enter `tutorial` for the name of the folder, then scroll down and choose **Create folder**.

To create the cluster

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. In the left pane, under **MSK Clusters**, choose **Clusters**.
3. Choose **Create cluster**.
4. In **Creation method**, choose **Custom create**.
5. For the cluster name enter **mkc-tutorial-cluster**.
6. In **Cluster type**, choose **Provisioned**.
7. Choose **Next**.
8. Under **Networking**, choose an Amazon VPC. Then select the Availability Zones and subnets that you want to use. Remember the IDs of the Amazon VPC and subnets that you selected because you need them later in this tutorial.
9. Choose **Next**.
10. Under **Access control methods** ensure that only **Unauthenticated access** is selected.
11. Under **Encryption** ensure that only **Plaintext** is selected.
12. Continue through the wizard and then choose **Create cluster**. This takes you to the details page for the cluster. On that page, under **Security groups applied**, find the security group ID. Remember that ID because you need it later in this tutorial.

To create an IAM policy with permissions to write to the S3 bucket

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. On the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. In **Policy editor**, choose **JSON**, and then replace the JSON in the editor window with the following JSON.

In the following example, replace `<amzn-s3-demo-bucket-my-tutorial>` with the name of your S3 bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowListBucket",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::<amzn-s3-demo-bucket-my-tutorial>"
    },
    {
      "Sid": "AllowObjectActions",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts",
        "s3:ListBucketMultipartUploads"
      ],
      "Resource": "arn:aws:s3:::<amzn-s3-demo-bucket-my-tutorial>/*"
    }
  ]
}
```

For instructions about how to write secure policies, see [the section called “IAM access control”](#).

5. Choose **Next**.
6. On the **Review and create** page, do the following:
 - a. For **Policy name**, enter a descriptive name, such as **mkc-tutorial-policy**.
 - b. In **Permissions defined in this policy**, review and/or edit the permissions defined in your policy.
 - c. (Optional) To help identify, organize, or search for the policy, choose **Add new tag** to add tags as key-value pairs. For example, add a tag to your policy with the key-value pair of **Environment** and **Test**.

For more information about using tags, see [Tags for AWS Identity and Access Management resources](#) in the *IAM User Guide*.

7. Choose **Create policy**.

To create the IAM role that can write to the destination bucket

1. On the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
2. On the **Select trusted entity** page, do the following:
 - a. For **Trusted entity type**, choose **AWS service**.
 - b. For **Service or use case**, choose **S3**.
 - c. Under **Use case**, choose **S3**.
3. Choose **Next**.
4. On the **Add permissions** page, do the following:
 - a. In the search box under **Permissions policies**, enter the name of the policy that you previously created for this tutorial. For example, **mkc-tutorial-policy**. Then, choose the box to the left of the policy name.
 - b. (Optional) Set a [permissions boundary](#). This is an advanced feature that is available for service roles, but not service-linked roles. For information about setting a permissions boundary, see [Creating roles and attaching policies \(console\)](#) in the *IAM User Guide*.
5. Choose **Next**.
6. On the **Name, review, and create** page, do the following:
 - a. For **Role name**, enter a descriptive name, such as **mkc-tutorial-role**.

Important

When you name a role, note the following:

- Role names must be unique within your AWS account, and can't be made unique by case.

For example, don't create roles named both **PRODRole** and **prodrole**. When a role name is used in a policy or as part of an ARN, the role name is case

sensitive, however when a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive.

- You can't edit the name of the role after it's created because other entities might reference the role.

- (Optional) For **Description**, enter a description for the role.
- (Optional) To edit the use cases and permissions for the role, in **Step 1: Select trusted entities** or **Step 2: Add permissions** sections, choose **Edit**.
- (Optional) To help identify, organize, or search for the role, choose **Add new tag** to add tags as key-value pairs. For example, add a tag to your role with the key-value pair of **ProductManager** and **John**.

For more information about using tags, see [Tags for AWS Identity and Access Management resources](#) in the *IAM User Guide*.

7. Review the role, and then choose **Create role**.

To allow MSK Connect to assume the role

1. In the IAM console, in the left pane, under **Access management**, choose **Roles**.
2. Find the `mkc-tutorial-role` and choose it.
3. Under the role's **Summary**, choose the **Trust relationships** tab.
4. Choose **Edit trust relationship**.
5. Replace the existing trust policy with the following JSON.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "kafkaconnect.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

6. Choose **Update Trust Policy**.

To create an Amazon VPC endpoint from the cluster's VPC to Amazon S3

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the left pane, choose **Endpoints**.
3. Choose **Create endpoint**.
4. Under **Service Name** choose the **com.amazonaws.us-east-1.s3** service and the **Gateway** type.
5. Choose the cluster's VPC and then select the box to the left of the route table that is associated with the cluster's subnets.
6. Choose **Create endpoint**.

Next Step

[Create custom plugin](#)

Create custom plugin

A plugin contains the code that defines the logic of the connector. In this step you create a custom plugin that has the code for the Lenses Amazon S3 Sink Connector. In a later step, when you create the MSK connector, you specify that its code is in this custom plugin. You can use the same plugin to create multiple MSK connectors with different configurations.

To create the custom plugin

1. Download the [S3 connector](#).
2. Upload the ZIP file to an S3 bucket to which you have access. For information on how to upload files to Amazon S3, see [Uploading objects](#) in the Amazon S3 user guide.
3. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
4. In the left pane expand **MSK Connect**, then choose **Custom plugins**.
5. Choose **Create custom plugin**.
6. Choose **Browse S3**.
7. In the list of buckets find the bucket where you uploaded the ZIP file, and choose that bucket.
8. In the list of objects in the bucket, select the radio button to the left of the ZIP file, then choose the button labeled **Choose**.
9. Enter `mkc-tutorial-plugin` for the custom plugin name, then choose **Create custom plugin**.

It might take AWS a few minutes to finish creating the custom plugin. When the creation process is complete, you see the following message in a banner at the top of the browser window.

Custom plugin mkc-tutorial-plugin was successfully created

The custom plugin was created. You can now create a connector using this custom plugin.

Next Step

[Create client machine and Apache Kafka topic](#)

Create client machine and Apache Kafka topic

In this step you create an Amazon EC2 instance to use as an Apache Kafka client instance. You then use this instance to create a topic on the cluster.

To create a client machine

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Launch instances**.
3. Enter a **Name** for your client machine, such as **mkc-tutorial-client**.
4. Leave **Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type** selected for **Amazon Machine Image (AMI) type**.
5. Choose the **t2.xlarge** instance type.
6. Under **Key pair (login)**, choose **Create a new key pair**. Enter **mkc-tutorial-key-pair** for **Key pair name**, and then choose **Download Key Pair**. Alternatively, you can use an existing key pair.
7. Choose **Launch instance**.
8. Choose **View Instances**. Then, in the **Security Groups** column, choose the security group that is associated with your new instance. Copy the ID of the security group, and save it for later.

To allow the newly created client to send data to the cluster

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the left pane, under **SECURITY**, choose **Security Groups**. In the **Security group ID** column, find the security group of the cluster. You saved the ID of this security group when you created the cluster in [the section called "Set up resources required for MSK Connect"](#). Choose this

security group by selecting the box to the left of its row. Make sure no other security groups are simultaneously selected.

3. In the bottom half of the screen, choose the **Inbound rules** tab.
4. Choose **Edit inbound rules**.
5. In the bottom left of the screen, choose **Add rule**.
6. In the new rule, choose **All traffic** in the **Type** column. In the field to the right of the **Source** column, enter the ID of the security group of the client machine. This is the security group ID that you saved after you created the client machine.
7. Choose **Save rules**. Your MSK cluster will now accept all traffic from the client you created in the previous procedure.

To create a topic

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the table of instances choose `mkc-tutorial-client`.
3. Near the top of the screen, choose **Connect**, then follow the instructions to connect to the instance.
4. Install Java on the client instance by running the following command:

```
sudo yum install java-1.8.0
```

5. Run the following command to download Apache Kafka.

```
wget https://archive.apache.org/dist/kafka/2.2.1/kafka_2.12-2.2.1.tgz
```

Note

If you want to use a mirror site other than the one used in this command, you can choose a different one on the [Apache](#) website.

6. Run the following command in the directory where you downloaded the TAR file in the previous step.

```
tar -xzf kafka_2.12-2.2.1.tgz
```

7. Go to the **kafka_2.12-2.2.1** directory.

8. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
9. In the left pane choose **Clusters**, then choose the name `mkc-tutorial-cluster`.
10. Choose **View client information**.
11. Copy the **Plaintext** connection string.
12. Choose **Done**.
13. Run the following command on the client instance (`mkc-tutorial-client`), replacing *bootstrapServerString* with the value that you saved when you viewed the cluster's client information.

```
<path-to-your-kafka-installation>/bin/kafka-topics.sh --create --bootstrap-server bootstrapServerString --replication-factor 2 --partitions 1 --topic mkc-tutorial-topic
```

If the command succeeds, you see the following message: Created topic `mkc-tutorial-topic`.

Next Step

[Create connector](#)

Create connector

This procedure describes how to create a connector using the AWS Management Console.

To create the connector

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. In the left pane, expand **MSK Connect**, then choose **Connectors**.
3. Choose **Create connector**.
4. In the list of plugins, choose `mkc-tutorial-plugin`, then choose **Next**.
5. For the connector name enter `mkc-tutorial-connector`.
6. In the list of clusters, choose `mkc-tutorial-cluster`.
7. Copy the following configuration and paste it into the connector configuration field.

Make sure that you replace region with the code of the AWS Region where you're creating the connector. Also, replace the Amazon S3 bucket name *<amzn-s3-demo-bucket-my-tutorial>* with the name of your bucket in the following example.

```
connector.class=io.confluent.connect.s3.S3SinkConnector
s3.region=us-east-1
format.class=io.confluent.connect.s3.format.json.JsonFormat
flush.size=1
schema.compatibility=NONE
tasks.max=2
topics=mkc-tutorial-topic
partitioner.class=io.confluent.connect.storage.partitionner.DefaultPartitioner
storage.class=io.confluent.connect.s3.storage.S3Storage
s3.bucket.name=<amzn-s3-demo-bucket-my-tutorial>
topics.dir=tutorial
```

8. Under **Access permissions** choose `mkc-tutorial-role`.
9. Choose **Next**. On the **Security** page, choose **Next** again.
10. On the **Logs** page choose **Next**.
11. Under **Review and create** choose **Create connector**.

Next Step

[Send data to the MSK cluster](#)

Send data to the MSK cluster

In this step you send data to the Apache Kafka topic that you created earlier, and then look for that same data in the destination S3 bucket.

To send data to the MSK cluster

1. In the `bin` folder of the Apache Kafka installation on the client instance, create a text file named `client.properties` with the following contents.

```
security.protocol=SASL_SSL
sasl.mechanism=AWS_MSK_IAM
```

2. Run the following command to create a console producer. Replace *BootstrapBrokerString* with the value that you obtained when you ran the previous command.

```
<path-to-your-kafka-installation>/bin/kafka-console-producer.sh --broker-list BootstrapBrokerString --producer.config client.properties --topic mkc-tutorial-topic
```

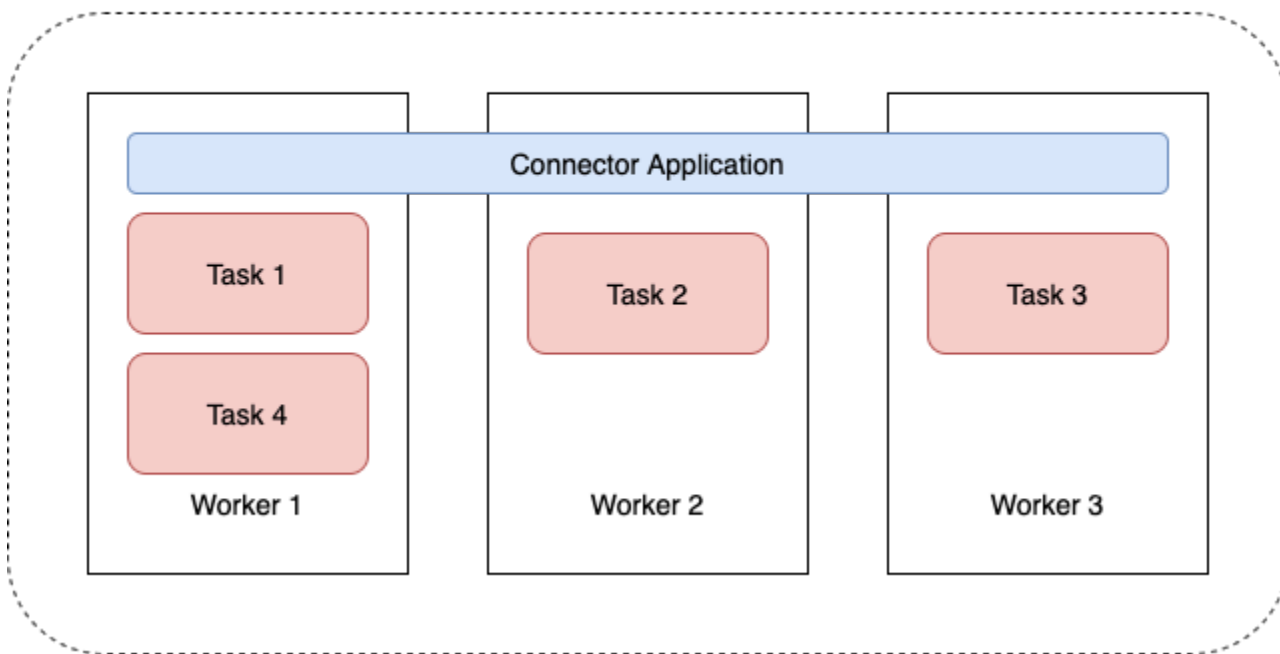
3. Enter any message that you want, and press **Enter**. Repeat this step two or three times. Every time you enter a line and press **Enter**, that line is sent to your Apache Kafka cluster as a separate message.
4. Look in the destination Amazon S3 bucket to find the messages that you sent in the previous step.

Understand connectors

A connector integrates external systems and Amazon services with Apache Kafka by continuously copying streaming data from a data source into your Apache Kafka cluster, or continuously copying data from your cluster into a data sink. A connector can also perform lightweight logic such as transformation, format conversion, or filtering data before delivering the data to a destination. Source connectors pull data from a data source and push this data into the cluster, while sink connectors pull data from the cluster and push this data into a data sink.

The following diagram shows the architecture of a connector. A worker is a Java virtual machine (JVM) process that runs the connector logic. Each worker creates a set of tasks that run in parallel threads and do the work of copying the data. Tasks don't store state, and can therefore be started, stopped, or restarted at any time in order to provide a resilient and scalable data pipeline.

Connector Architecture



Understand connector capacity

The total capacity of a connector depends on the number of workers that the connector has, as well as on the number of MSK Connect Units (MCUs) per worker. Each MCU represents 1 vCPU of compute and 4 GiB of memory. The MCU memory pertains to the total memory of a worker instance and not the heap memory in use.

MSK Connect workers consume IP addresses in the customer-provided subnets. Each worker uses one IP address from one of the customer-provided subnets. You should ensure that you have enough available IP addresses in the subnets provided to a `CreateConnector` request to account for their specified capacity, especially when autoscaling connectors where the number of workers can fluctuate.

To create a connector, you must choose between one of the following two capacity modes.

- *Provisioned* - Choose this mode if you know the capacity requirements for your connector. You specify two values:
 - The number of workers.
 - The number of MCUs per worker.

- *Autoscaled* - Choose this mode if the capacity requirements for your connector are variable or if you don't know them in advance. When you use autoscaled mode, Amazon MSK Connect overrides your connector's `tasks.max` property with a value that is proportional to the number of workers running in the connector and the number of MCUs per worker.

You specify three sets of values:

- The minimum and maximum number of workers.
- The scale-in and scale-out percentages for CPU utilization, which is determined by the `CpuUtilization` metric. When the `CpuUtilization` metric for the connector exceeds the scale-out percentage, MSK Connect increases the number of workers that are running in the connector. When the `CpuUtilization` metric goes below the scale-in percentage, MSK Connect decreases the number of workers. The number of workers always remains within the minimum and maximum numbers that you specify when you create the connector.
- The number of MCUs per worker.

For more information about workers, see [the section called “Understand MSK Connect workers”](#). To learn about MSK Connect metrics, see [the section called “Monitoring”](#).

Create a connector

This procedure describes how to create a connector using the AWS Management Console.

Creating a connector using the AWS Management Console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. In the left pane, under **MSK Connect**, choose **Connectors**.
3. Choose **Create connector**.
4. You can choose between using an existing custom plugin to create the connector, or creating a new custom plugin first. For information on custom plugins and how to create them, see [the section called “Create custom plugins”](#). In this procedure, let's assume you have a custom plugin that you want to use. In the list of custom plugins, find the one that you want to use, and select the box to its left, then choose **Next**.
5. Enter a name and, optionally, a description.
6. Choose the cluster that you want to connect to.
7. Specify the connector configuration. The configuration parameters that you need to specify depend on the type of connector that you want to create. However, some parameters are

common to all connectors, for example, the `connector.class` and `tasks.max` parameters. The following is an example configuration for the [Confluent Amazon S3 Sink Connector](#).

```
connector.class=io.confluent.connect.s3.S3SinkConnector
tasks.max=2
topics=my-example-topic
s3.region=us-east-1
s3.bucket.name=amzn-s3-demo-bucket
flush.size=1
storage.class=io.confluent.connect.s3.storage.S3Storage
format.class=io.confluent.connect.s3.format.json.JsonFormat
partitioner.class=io.confluent.connect.storage.partitioners.DefaultPartitioner
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
schema.compatibility=NONE
```

8. Next, you configure your connector capacity. You can choose between two capacity modes: provisioned and auto scaled. For information about these two options, see [the section called “Understand connector capacity”](#).
9. Choose either the default worker configuration or a custom worker configuration. For information about creating custom worker configurations, see [the section called “Understand MSK Connect workers”](#).
10. Next, you specify the service execution role. This must be an IAM role that MSK Connect can assume, and that grants the connector all the permissions that it needs to access the necessary AWS resources. Those permissions depend on the logic of the connector. For information about how to create this role, see [the section called “Understand service execution role”](#).
11. Choose **Next**, review the security information, then choose **Next** again.
12. Specify the logging options that you want, then choose **Next**. For information about logging, see [the section called “Logging”](#).
13. Choose **Create connector**.

To use the MSK Connect API to create a connector, see [CreateConnector](#).

You can use `UpdateConnector` API to modify the connector's configuration. For more information, see [the section called “Update a connector”](#).

Update a connector

This procedure describes how to update the configuration of an existing MSK Connect connector using the AWS Management Console.

Updating connector configuration using the AWS Management Console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. In the left pane, under **MSK Connect**, choose **Connectors**.
3. Select an existig connector.
4. Choose **Edit connector configuration**.
5. Update the connector configuration. You can't override `connector.class` using `UpdateConnector`. The following example shows an example configuration for the Confluent Amazon S3 Sink connector.

```
connector.class=io.confluent.connect.s3.S3SinkConnector
tasks.max=2
topics=my-example-topic
s3.region=us-east-1
s3.bucket.name=amzn-s3-demo-bucket
flush.size=1
storage.class=io.confluent.connect.s3.storage.S3Storage
format.class=io.confluent.connect.s3.format.json.JsonFormat
partitioner.class=io.confluent.connect.storage.partitionner.DefaultPartitioner
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
schema.compatibility=NONE
```

6. Choose **Submit**.
7. You can then monitor the current state of the operation in the **Operations** tab of the connector.

To use the MSK Connect API to update the configuration of a connector, see [UpdateConnector](#).

Connecting from connectors

The following best practices can improve the performance of your connectivity to Amazon MSK Connect.

Do not overlap IPs for Amazon VPC peering or Transit Gateway

If you are using Amazon VPC peering or Transit Gateway with Amazon MSK Connect, do not configure your connector for reaching the peered VPC resources with IPs in the CIDR ranges:

- "10.99.0.0/16"
- "192.168.0.0/16"
- "172.21.0.0/16"

Create custom plugins

A plugin is an AWS resource that contains the code that defines your connector logic. You upload a JAR file (or a ZIP file that contains one or more JAR files) to an S3 bucket, and specify the location of the bucket when you create the plugin. When you create a connector, you specify the plugin that you want MSK Connect to use for it. The relationship of plugins to connectors is one-to-many: You can create one or more connectors from the same plugin.

For information on how to develop the code for a connector, see the [Connector Development Guide](#) in the Apache Kafka documentation.

Creating a custom plugin using the AWS Management Console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. In the left pane, under **MSK Connect**, choose **Custom plugins**.
3. Choose **Create custom plugin**.
4. Choose **Browse S3**.
5. In the list of S3 buckets, choose the bucket that has the JAR or ZIP file for the plugin.
6. In the list of object, select the box to the left of the JAR or ZIP file for the plugin, then choose **Choose**.
7. Choose **Create custom plugin**.

To use the MSK Connect API to create a custom plugin, see [CreateCustomPlugin](#).

Understand MSK Connect workers

A worker is a Java virtual machine (JVM) process that runs the connector logic. Each worker creates a set of tasks that run in parallel threads and do the work of copying the data. Tasks don't store state, and can therefore be started, stopped, or restarted at any time in order to provide a resilient and scalable data pipeline. Changes to the number of workers, whether due to a scaling event or due to unexpected failures, are automatically detected by the remaining workers. They coordinate to rebalance tasks across the set of remaining workers. Connect workers use Apache Kafka's consumer groups to coordinate and rebalance.

If your connector's capacity requirements are variable or difficult to estimate, you can let MSK Connect scale the number of workers as needed between a lower limit and an upper limit that you specify. Alternatively, you can specify the exact number of workers that you want to run your connector logic. For more information, see [the section called “Understand connector capacity”](#).

MSK Connect workers consume IP addresses

MSK Connect workers consume IP addresses in the customer-provided subnets. Each worker uses one IP address from one of the customer-provided subnets. You should ensure that you have enough available IP addresses in the subnets provided to a CreateConnector request to account for their specified capacity, especially when autoscaling connectors where the number of workers can fluctuate.

Default worker configuration

MSK Connect provides the following default worker configuration:

```
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
```

Supported worker configuration properties

MSK Connect provides a default worker configuration. You also have the option to create a custom worker configuration to use with your connectors. The following list includes information about the worker configuration properties that Amazon MSK Connect does or does not support.

- The `key.converter` and `value.converter` properties are required.
- MSK Connect supports the following `producer.` configuration properties.

```
producer.acks
producer.batch.size
producer.buffer.memory
producer.compression.type
producer.enable.idempotence
producer.key.serializer
producer.linger.ms
producer.max.request.size
producer.metadata.max.age.ms
producer.metadata.max.idle.ms
producer.partitioner.class
producer.reconnect.backoff.max.ms
producer.reconnect.backoff.ms
producer.request.timeout.ms
producer.retry.backoff.ms
producer.value.serializer
```

- MSK Connect supports the following `consumer.` configuration properties.

```
consumer.allow.auto.create.topics
consumer.auto.offset.reset
consumer.check.crcs
consumer.fetch.max.bytes
consumer.fetch.max.wait.ms
consumer.fetch.min.bytes
consumer.heartbeat.interval.ms
consumer.key.deserializer
consumer.max.partition.fetch.bytes
consumer.max.poll.interval.ms
consumer.max.poll.records
consumer.metadata.max.age.ms
consumer.partition.assignment.strategy
consumer.reconnect.backoff.max.ms
consumer.reconnect.backoff.ms
consumer.request.timeout.ms
consumer.retry.backoff.ms
consumer.session.timeout.ms
consumer.value.deserializer
```

- All other configuration properties that don't start with the `producer.` or `consumer.` prefixes are supported *except* for the following properties.

```
access.control.  
admin.  
admin.listeners.https.  
client.  
connect.  
inter.worker.  
internal.  
listeners.https.  
metrics.  
metrics.context.  
rest.  
sasl.  
security.  
socket.  
ssl.  
topic.tracking.  
worker.  
bootstrap.servers  
config.storage.topic  
connections.max.idle.ms  
connector.client.config.override.policy  
group.id  
listeners  
metric.reporters  
plugin.path  
receive.buffer.bytes  
response.http.headers.config  
scheduled.rebalance.max.delay.ms  
send.buffer.bytes  
status.storage.topic
```

For more information about worker configuration properties and what they represent, see [Kafka Connect Configs](#) in the Apache Kafka documentation.

Create a custom worker configuration

This procedure describes how to create a custom worker configuration using the AWS Management Console.

Creating a custom worker configuration using the AWS Management Console

1. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
2. In the left pane, under **MSK Connect**, choose **Worker configurations**.
3. Choose **Create worker configuration**.
4. Enter a name and an optional description, then add the properties and values that you want to set them to.
5. Choose **Create worker configuration**.

To use the MSK Connect API to create a worker configuration, see [CreateWorkerConfiguration](#).

Manage source connector offsets using `offset.storage.topic`

This section provides information to help you manage source connector offsets using the *offset storage topic*. The offset storage topic is an internal topic that Kafka Connect uses to store connector and task configuration offsets.

Considerations

Consider the following when you manage source connector offsets.

- To specify an offset storage topic, provide the name of the Kafka topic where connector offsets are stored as the value for `offset.storage.topic` in your worker configuration.
- Use caution when you make changes to a connector configuration. Changing configuration values may result in unintended connector behavior if a source connector uses values from the configuration to key offset records. We recommend that you refer to your plugin's documentation for guidance.
- **Customize default number of partitions** – In addition to customizing the worker configuration by adding `offset.storage.topic`, you can customize the number of partitions for the offset and status storage topics. Default partitions for internal topics are as follows.
 - `config.storage.topic`: 1, not configurable, must be single partition topic
 - `offset.storage.topic`: 25, configurable by providing `offset.storage.partitions`
 - `status.storage.topic`: 5, configurable by providing `status.storage.partitions`
- **Manually deleting topics** – Amazon MSK Connect creates new Kafka connect internal topics (topic name starts with `__amazon_msk_connect`) on every deployment of connectors.

Old topics that are attached to deleted connectors are not automatically removed because internal topics, such as `offset.storage.topic`, can be reused among connectors. However, you can manually delete unused internal topics created by MSK Connect. The internal topics are named following the format `__amazon_msk_connect_<offsets|status|configs>_connector_name_connector_id`.

The regular expression `__amazon_msk_connect_<offsets|status|configs>_connector_name_connector_id` can be used to delete the internal topics. You should not delete an internal topic that is currently in use by a running connector.

- **Using the same name for the internal topics created by MSK Connect** – If you want to reuse the offset storage topic to consume offsets from a previously created connector, you must give the new connector the same name as the old connector. The `offset.storage.topic` property can be set using the worker configuration to assign the same name to the `offset.storage.topic` and reused between different connectors. This configuration is described in [Managing connector offsets](#). MSK Connect does not allow different connectors to share `config.storage.topic` and `status.storage.topic`. Those topics are created each time you create a new connector in MSKC. They are automatically named following the format `__amazon_msk_connect_<status|configs>_connector_name_connector_id`, and so are different across the different connectors that you create.

Use the default offset storage topic

By default, Amazon MSK Connect generates a new offset storage topic on your Kafka cluster for each connector that you create. MSK constructs the default topic name using parts of the connector ARN. For example, `__amazon_msk_connect_offsets_my-mskc-connector_12345678-09e7-4abc-8be8-c657f7e4ff32-2`.

Use custom offset storage topic

To provide offset continuity between source connectors, you can use an offset storage topic of your choice instead of the default topic. Specifying an offset storage topic helps you accomplish tasks like creating a source connector that resumes reading from the last offset of a previous connector.

To specify an offset storage topic, you supply a value for the `offset.storage.topic` property in your worker configuration before you create a connector. If you want to reuse the offset storage topic to consume offsets from a previously created connector, you must give the new connector the same name as the old connector. If you create a custom offset storage topic, you must set [cleanup.policy](#) to compact in your topic configuration.

Note

If you specify an offset storage topic when you create a *sink* connector, MSK Connect creates the topic if it does not already exist. However, the topic will not be used to store connector offsets.

Sink connector offsets are instead managed using the Kafka consumer group protocol. Each sink connector creates a group named `connect-{CONNECTOR_NAME}`. As long as the consumer group exists, any successive sink connectors that you create with the same `CONNECTOR_NAME` value will continue from the last committed offset.

Example : Specifying an offset storage topic to recreate a source connector with an updated configuration

Suppose you have a change data capture (CDC) connector and you want to modify the connector configuration without losing your place in the CDC stream. You can't update the existing connector configuration, but you can delete the connector and create a new one with the same name.

To tell the new connector where to start reading in the CDC stream, you can specify the old connector's offset storage topic in your worker configuration. The following steps demonstrate how to accomplish this task.

1. On your client machine, run the following command to find the name of your connector's offset storage topic. Replace `<bootstrapBrokerString>` with your cluster's bootstrap broker string. For instructions on getting your bootstrap broker string, see [Get the bootstrap brokers for an Amazon MSK cluster](#).

```
<path-to-your-kafka-installation>/bin/kafka-topics.sh --list --bootstrap-server <bootstrapBrokerString>
```

The following output shows a list of all cluster topics, including any default internal connector topics. In this example, the existing CDC connector uses the [default offset storage topic](#) created by MSK Connect. This is why the offset storage topic is called `__amazon_msk_connect_offsets_my-mskc-connector_12345678-09e7-4abc-8be8-c657f7e4ff32-2`.


```
__consumer_offsets
__amazon_msk_canary
```

```
__amazon_msk_connect_configs_my-mskc-connector_12345678-09e7-4abc-8be8-  
c657f7e4ff32-2  
__amazon_msk_connect_offsets_my-mskc-connector_12345678-09e7-4abc-8be8-  
c657f7e4ff32-2  
__amazon_msk_connect_status_my-mskc-connector_12345678-09e7-4abc-8be8-  
c657f7e4ff32-2  
my-msk-topic-1  
my-msk-topic-2
```

2. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/>.
3. Choose your connector from the **Connectors** list. Copy and save the contents of the **Connector configuration** field so that you can modify it and use it to create the new connector.
4. Choose **Delete** to delete the connector. Then enter the connector name in the text input field to confirm deletion.
5. Create a custom worker configuration with values that fit your scenario. For instructions, see [Create a custom worker configuration](#).

In your worker configuration, you must specify the name of the offset storage topic that you previously retrieved as the value for `offset.storage.topic` like in the following configuration.

```
config.providers.secretManager.param.aws.region=eu-west-3  
key.converter=<org.apache.kafka.connect.storage.StringConverter>  
value.converter=<org.apache.kafka.connect.storage.StringConverter>  
config.providers.secretManager.class=com.github.jcustenborder.kafka.config.aws.SecretsManager  
config.providers=secretManager  
offset.storage.topic=__amazon_msk_connect_offsets_my-mskc-  
connector_12345678-09e7-4abc-8be8-c657f7e4ff32-2
```

6.  **Important**
You must give your new connector the same name as the old connector.

Create a new connector using the worker configuration that you set up in the previous step. For instructions, see [Create a connector](#).

Tutorial: Externalizing sensitive information using config providers

This example shows how to externalize sensitive information for Amazon MSK Connect using an open source configuration provider. A configuration providers lets you specify variables instead of plaintext in a connector or worker configuration, and workers running in your connector resolve these variables at runtime. This prevents credentials and other secrets from being stored in plaintext. The configuration provider in the example supports retrieving configuration parameters from AWS Secrets Manager, Amazon S3 and Systems Manager (SSM). In [Step 2](#), you can see how to set up storage and retrieval of sensitive information for the service that you want to configure.

Considerations

Consider the following while using the MSK config provider with Amazon MSK Connect:

- Assign appropriate permissions when using the config providers to the IAM Service Execution Role.
- Define the config providers in worker configurations and their implementation in the connector configuration.
- Sensitive configuration values can appear in connector logs if a plugin does not define those values as secret. Kafka Connect treats undefined configuration values the same as any other plaintext value. To learn more, see [Preventing secrets from appearing in connector logs](#).
- By default, MSK Connect frequently restarts a connector when the connector uses a configuration provider. To turn off this restart behavior, you can set the `config.action.reload` value to `none` in your connector configuration.

Create a custom plugin and upload to S3

To create a custom-plugin, create a zip file that contains the connector and the msk-config-provider by running the following commands on your local machine.

To create a custom plugin using a terminal window and Debezium as the connector

Use the AWS CLI to run commands as a superuser with credentials that allow you to access your AWS S3 bucket. For information on installing and setting up the AWS CLI, see [Getting started with the AWS CLI](#) in the *AWS Command Line Interface User Guide*. For information on using the AWS CLI

with Amazon S3, see [Using Amazon S3 with the AWS CLI](#) in the *AWS Command Line Interface User Guide*.

1. In a terminal window, create a folder named custom-plugin in your workspace using the following command.

```
mkdir custom-plugin && cd custom-plugin
```

2. Download the latest stable release of the **MySQL Connector Plug-in** from the [Debezium site](#) using the following command.

```
wget https://repo1.maven.org/maven2/io/debezium/debezium-connectormysql/2.2.0.Final/debezium-connector-mysql-2.2.0.Final-plugin.tar.gz
```

Extract the downloaded gzip file in the custom-plugin folder using the following command.

```
tar xzf debezium-connector-mysql-2.2.0.Final-plugin.tar.gz
```

3. Download the [MSK config provider zip file](#) using the following command.

```
wget https://github.com/aws-samples/msk-config-providers/releases/download/r0.1.0/msk-config-providers-0.1.0-with-dependencies.zip
```

Extract the downloaded zip file in the custom-plugin folder using the following command.

```
unzip msk-config-providers-0.1.0-with-dependencies.zip
```

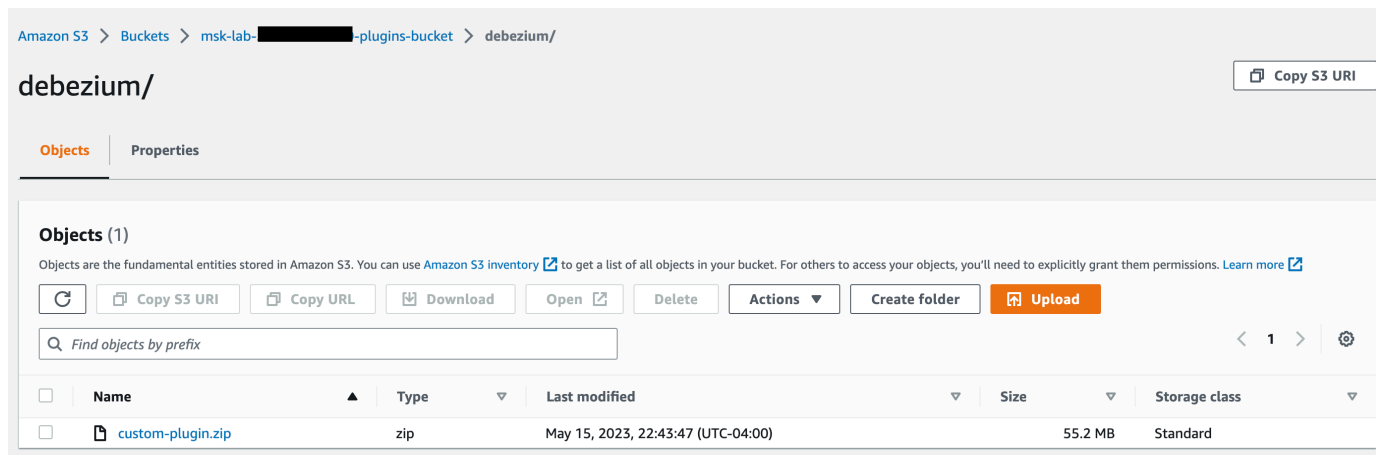
4. Zip the contents of the MSK config provider from the above step and the custom connector into a single file named custom-plugin.zip.

```
zip -r ../custom-plugin.zip *
```

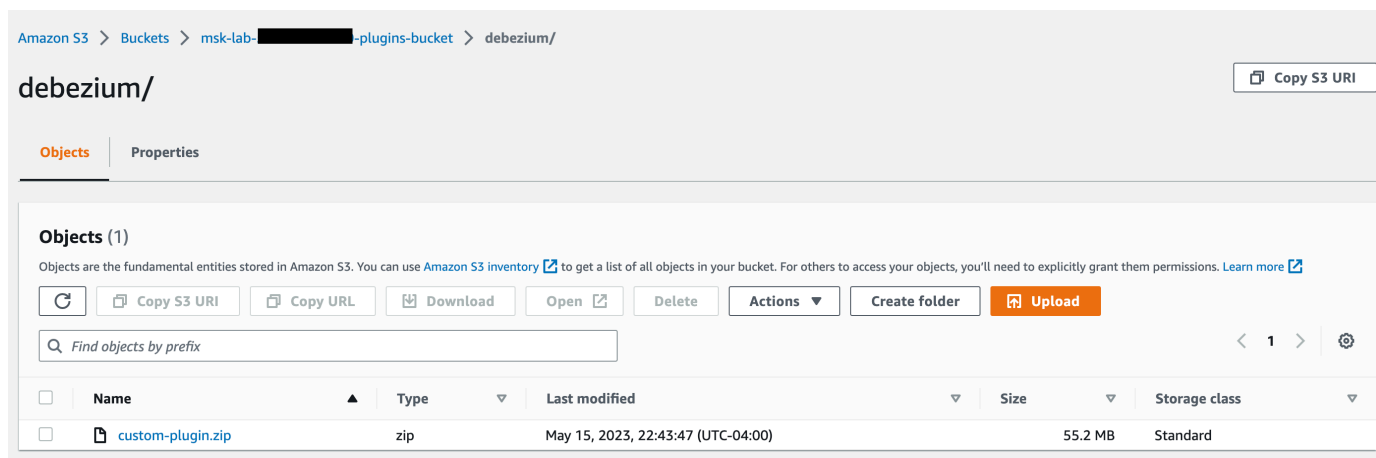
5. Upload the file to S3 to be referenced later.

```
aws s3 cp ../custom-plugin.zip s3:<S3_URI_BUCKET_LOCATION>
```

6. On the Amazon MSK console, under the **MSK Connect** section, choose **Custom Plugin**, then choose **Create custom plugin** and browse the **s3:<**S3_URI_BUCKET_LOCATION**>** S3 bucket to select the custom plugin ZIP file you just uploaded.



7. Enter **debezium-custom-plugin** for the plugin name. Optionally, enter a description and choose **Create Custom Plugin**.



Configure parameters and permissions for different providers

You can configure parameter values in these three services:

- Secrets Manager
- Systems Manager Parameter Store
- S3 - Simple Storage Service

Select one of the tabs below for instructions on setting up parameters and relevant permissions for that service.

Configure in Secrets Manager

To configure parameter values in Secrets Manager

1. Open the [Secrets Manager console](#).
2. Create a new secret to store your credentials or secrets. For instructions, see [Create an AWS Secrets Manager secret](#) in the AWS Secrets Manager User Guide.
3. Copy your secret's ARN.
4. Add the Secrets Manager permissions from the following example policy to your [Service execution role](#). Replace *<arn:aws:secretsmanager:us-east-1:123456789000:secret:MySecret-1234>* with the ARN of your secret.
5. Add worker configuration and connector instructions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetResourcePolicy",
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret",
        "secretsmanager:ListSecretVersionIds"
      ],
      "Resource": [
        "<arn:aws:secretsmanager:us-east-1:123456789000:secret:MySecret-1234>"
      ]
    }
  ]
}
```

6. For using the Secrets Manager configuration provider, copy the following lines of code to the worker configuration textbox in Step 3:

```
# define name of config provider:

config.providers = secretsmanager

# provide implementation classes for secrets manager:
```

```
config.providers.secretsmanager.class =
    com.amazonaws.kafka.config.providers.SecretsManagerConfigProvider

# configure a config provider (if it needs additional initialization), for
# example you can provide a region where the secrets or parameters are located:

config.providers.secretsmanager.param.region = us-east-1
```

7. For the secrets manager configuration provider, copy the following lines of code in the connector configuration in Step 4.

```
#Example implementation for secrets manager variable
database.user=${secretsmanager:MSKAuroraDBCredentials:username}
database.password=${secretsmanager:MSKAuroraDBCredentials:password}
```

You may also use the above step with more configuration providers.

Configure in Systems Manager Parameter Store

To configure parameter values in Systems Manager Parameter Store

1. Open the [Systems Manager console](#).
2. In the navigation pane, choose **Parameter Store**.
3. Create a new parameter to store in the Systems Manager. For instructions, see [Create a Systems Manager parameter \(console\)](#) in the AWS Systems Manager User Guide.
4. Copy your parameter's ARN.
5. Add the Systems Manager permissions from the following example policy to your [Service execution role](#). Replace `<arn:aws:ssm:us-east-1:123456789000:parameter/MyParameterName>` with the ARN of your parameter.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameterHistory",
        "ssm:GetParametersByPath",
        "ssm:GetParameters",
```

```

        "ssm:GetParameter"
    ],
    "Resource": "arn:aws:ssm:us-east-1:123456789000:parameter/
MyParameterName"
    }
]
}

```

- For using the parameter store configuration provider, copy the following lines of code to the worker configuration textbox in Step 3:

```

# define name of config provider:

config.providers = ssm

# provide implementation classes for parameter store:

config.providers.ssm.class =
    com.amazonaws.kafka.config.providers.SsmParamStoreConfigProvider

# configure a config provider (if it needs additional initialization), for
# example you can provide a region where the secrets or parameters are located:

config.providers.ssm.param.region = us-east-1

```

- For the parameter store configuration provider copy the following lines of code in the connector configuration in Step 5.

```

#Example implementation for parameter store variable
schema.history.internal.kafka.bootstrap.servers=
${ssm:MSKBootstrapServerAddress}

```

You may also bundle the above two steps with more configuration providers.

Configure in Amazon S3

To configure objects/files in Amazon S3

- Open the [Amazon S3 console](#).
- Upload your object to a bucket in S3. For instructions, see [Uploading objects](#).
- Copy your object's ARN.

4. Add the Amazon S3 Object Read permissions from the following example policy to your [Service execution role](#). Replace `<arn:aws:s3:::MY_S3_BUCKET/path/to/custom-plugin.zip>` with the ARN of your object.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "<arn:aws:s3:::MY_S3_BUCKET/path/to/custom-
plugin.zip>"
    }
  ]
}
```

5. For using the Amazon S3 configuration provider, copy the following lines of code to the worker configuration text-box in Step 3:

```
# define name of config provider:

config.providers = s3import
# provide implementation classes for S3:

config.providers.s3import.class =
  com.amazonaws.kafka.config.providers.S3ImportConfigProvider
```

6. For the Amazon S3 configuration provider, copy the following lines of code to the connector configuration in Step 4.

```
#Example implementation for S3 object

database.ssl.truststore.location = ${s3import:us-west-2:my_cert_bucket/path/to/
truststore_unique_filename.jks}
```

You may also bundle the above two steps with more configuration providers.

Create a custom worker configuration with information about your configuration provider

1. Select **Worker configurations** under the **Amazon MSK Connect** section.
2. Select **Create worker configuration**.
3. Enter `SourceDebeziumCustomConfig` in the Worker Configuration Name textbox. The Description is optional.
4. Copy the relevant configuration code based on the providers desired, and paste it in the **Worker configuration** textbox.
5. This is an example of the worker configuration for all the three providers:

```
key.converter=org.apache.kafka.connect.storage.StringConverter
key.converter.schemas.enable=false
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=false
offset.storage.topic=offsets_my_debezium_source_connector

# define names of config providers:

config.providers=secretsmanager,ssm,s3import

# provide implementation classes for each provider:

config.providers.secretsmanager.class    =
  com.amazonaws.kafka.config.providers.SecretsManagerConfigProvider
config.providers.ssm.class                =
  com.amazonaws.kafka.config.providers.SsmParamStoreConfigProvider
config.providers.s3import.class           =
  com.amazonaws.kafka.config.providers.S3ImportConfigProvider

# configure a config provider (if it needs additional initialization), for example
# you can provide a region where the secrets or parameters are located:

config.providers.secretsmanager.param.region = us-east-1
config.providers.ssm.param.region = us-east-1
```

6. Click on **Create worker configuration**.

Create the connector

1. Create a new connector using the instructions in [Create a new connector](#).
2. Choose the custom-plugin.zip file that you uploaded to your S3 bucket in [???](#) as the source for the custom plugin.
3. Copy the relevant configuration code based on the providers desired, and paste them in the Connector configuration field.
4. This is an example for the connector configuration for all the three providers:

```
#Example implementation for parameter store variable
schema.history.internal.kafka.bootstrap.servers=${ssm:MSKBootstrapServerAddress}

#Example implementation for secrets manager variable
database.user=${secretsmanager:MSKAuroraDBCredentials:username}
database.password=${secretsmanager:MSKAuroraDBCredentials:password}

#Example implementation for Amazon S3 file/object
database.ssl.truststore.location = ${s3import:us-west-2:my_cert_bucket/path/to/trustore_unique_filename.jks}
```

5. Select **Use a custom configuration** and choose **SourceDebeziumCustomConfig** from the **Worker Configuration** dropdown.
6. Follow the remaining steps from instructions in [Create connector](#).

IAM roles and policies for MSK Connect

This section helps you set up the appropriate IAM policies and roles to securely deploy and manage Amazon MSK Connect within your AWS environment. The following sections explain the service execution role that must be used with MSK Connect, including the required trust policy and additional permissions needed when connecting to an IAM-authenticated MSK cluster. The page also provides examples of comprehensive IAM policies to grant full access to MSK Connect functionality, as well as details on AWS managed policies available for the service.

Topics

- [Understand service execution role](#)
- [Example of IAM policy for MSK Connect](#)
- [Prevent cross-service confused deputy problem](#)

- [AWS managed policies for MSK Connect](#)
- [Use service-linked roles for MSK Connect](#)

Understand service execution role

Note

Amazon MSK Connect does not support using the [Service-linked role](#) as the service execution role. You must create a separate service execution role. For instructions on how to create a custom IAM role, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

When you create a connector with MSK Connect, you're required to specify an AWS Identity and Access Management (IAM) role to use with it. Your service execution role must have the following trust policy so that MSK Connect can assume it. For information about the condition context keys in this policy, see [the section called "Prevent cross-service confused deputy problem"](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "kafkaconnect.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:kafkaconnect:us-east-1:123456789012:connector/
myConnector/abc12345-abcd-4444-a8b9-123456f513ed-2"
        }
      }
    }
  ]
}
```

If the Amazon MSK cluster that you want to use with your connector is a cluster that uses IAM authentication, then you must add the following permissions policy to the connector's service execution role. For information about how to find your cluster's UUID and how to construct topic ARNs, see [the section called "Authorization policy resources"](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:Connect",
        "kafka-cluster:DescribeCluster"
      ],
      "Resource": [
        "arn:aws:kafka:us-east-1:000000000001:cluster/
        testClusterName/300d0000-0000-0005-000f-00000000000b-1"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:ReadData",
        "kafka-cluster:DescribeTopic"
      ],
      "Resource": [
        "arn:aws:kafka:us-east-1:123456789012:topic/
        myCluster/300a0000-0000-0003-000a-00000000000b-6/__amazon_msk_connect_read"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:WriteData",
        "kafka-cluster:DescribeTopic"
      ],
      "Resource": [
        "arn:aws:kafka:us-east-1:123456789012:topic/
        testCluster/300f0000-0000-0008-000d-00000000000m-7/__amazon_msk_connect_write"
      ]
    },
    {
      "Effect": "Allow",
```

```

        "Action": [
            "kafka-cluster:CreateTopic",
            "kafka-cluster:WriteData",
            "kafka-cluster:ReadData",
            "kafka-cluster:DescribeTopic"
        ],
        "Resource": [
            "arn:aws:kafka:us-  
east-1:123456789012:topic/testCluster/300f0000-0000-0008-000d-000000000000m-7/  
__amazon_msk_connect_*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "kafka-cluster:AlterGroup",
            "kafka-cluster:DescribeGroup"
        ],
        "Resource": [
            "arn:aws:kafka:us-  
east-1:123456789012:group/testCluster/300d0000-0000-0005-000f-000000000000b-1/  
__amazon_msk_connect_*",
            "arn:aws:kafka:us-  
east-1:123456789012:group/testCluster/300d0000-0000-0005-000f-000000000000b-1/connect-*"
        ]
    }
]
}

```

Depending on the kind of connector, you might also need to attach to the service execution role a permissions policy that allows it to access AWS resources. For example, if your connector needs to send data to an S3 bucket, then the service execution role must have a permissions policy that grants permission to write to that bucket. For testing purposes, you can use one of the pre-built IAM policies that give full access, like `arn:aws:iam::aws:policy/AmazonS3FullAccess`. However, for security purposes, we recommend that you use the most restrictive policy that allows your connector to read from the AWS source or write to the AWS sink.

Example of IAM policy for MSK Connect

To give a non-admin user full access to all MSK Connect functionality, attach a policy like the following one to the user's IAM role.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MSKConnectFullAccess",
      "Effect": "Allow",
      "Action": [
        "kafkaconnect:CreateConnector",
        "kafkaconnect:DeleteConnector",
        "kafkaconnect:DescribeConnector",
        "kafkaconnect:GetConnector",
        "kafkaconnect:ListConnectors",
        "kafkaconnect:UpdateConnector",
        "kafkaconnect:CreateCustomPlugin",
        "kafkaconnect:DeleteCustomPlugin",
        "kafkaconnect:DescribeCustomPlugin",
        "kafkaconnect:GetCustomPlugin",
        "kafkaconnect:ListCustomPlugins",
        "kafkaconnect:CreateWorkerConfiguration",
        "kafkaconnect:DeleteWorkerConfiguration",
        "kafkaconnect:DescribeWorkerConfiguration",
        "kafkaconnect:GetWorkerConfiguration",
        "kafkaconnect:ListWorkerConfigurations"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRole",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::123456789012:role/MSKConnectServiceRole",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "kafkaconnect.amazonaws.com"
        }
      }
    }
  ],
  {
    "Sid": "EC2NetworkAccess",
    "Effect": "Allow",
    "Action": [
      "ec2:CreateNetworkInterface",
      "ec2:DescribeNetworkInterfaces",

```

```

        "ec2:DeleteNetworkInterface",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
    ],
    "Resource": "*"
},
{
    "Sid": "MSKClusterAccess",
    "Effect": "Allow",
    "Action": [
        "kafka:DescribeCluster",
        "kafka:DescribeClusterV2",
        "kafka:GetBootstrapBrokers"
    ],
    "Resource": "arn:aws:kafka:us-east-1:123456789012:cluster/myCluster/"
},
{
    "Sid": "MSKLogGroupAccess",
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups"
    ],
    "Resource": [
        "arn:aws:logs:us-east-1:123456789012:log-group:/aws/msk-connect/*"
    ]
},
{
    "Sid": "S3PluginAccess",
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:ListBucket",
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket1-custom-plugins",
        "arn:aws:s3:::amzn-s3-demo-bucket1-custom-plugins/*"
    ]
}

```

```
]
}
```

Prevent cross-service confused deputy problem

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the [aws:SourceArn](#) and [aws:SourceAccount](#) global condition context keys in resource policies to limit the permissions that MSK Connect gives another service to the resource. If the `aws:SourceArn` value does not contain the account ID (for example, an Amazon S3 bucket ARN doesn't contain the account ID), you must use both global condition context keys to limit permissions. If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement. 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.

In the case of MSK Connect, the value of `aws:SourceArn` must be an MSK connector.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:kafkaconnect:us-east-1:123456789012:connector/*` represents all connectors that belong to the account with ID 123456789012 in the US East (N. Virginia) Region.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in MSK Connect to prevent the confused deputy problem. Replace *Account-ID* and *MSK-Connector-ARN* with your information.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": " kafkaconnect.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "Account-ID"
      },
      "ArnLike": {
        "aws:SourceArn": "MSK-Connector-ARN"
      }
    }
  }
]
```

AWS managed policies for MSK Connect

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining [customer managed policies](#) that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see [AWS managed policies](#) in the *IAM User Guide*.

AWS managed policy: AmazonMSKConnectReadOnlyAccess

This policy grants the user the permissions that are needed to list and describe MSK Connect resources.

You can attach the AmazonMSKConnectReadOnlyAccess policy to your IAM identities.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kafkaconnect:ListConnectors",
        "kafkaconnect:ListCustomPlugins",
        "kafkaconnect:ListWorkerConfigurations"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kafkaconnect:DescribeConnector"
      ],
      "Resource": [
        "arn:aws:kafkaconnect:*:*:connector/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kafkaconnect:DescribeCustomPlugin"
      ],
      "Resource": [
        "arn:aws:kafkaconnect:*:*:custom-plugin/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kafkaconnect:DescribeWorkerConfiguration"
      ],
      "Resource": [
```

```

        "arn:aws:kafkaconnect:*:*:worker-configuration/*"
    ]
}
]
}

```

AWS managed policy: **KafkaConnectServiceRolePolicy**

This policy grants the MSK Connect service the permissions that are needed to create and manage network interfaces that have the tag `AmazonMSKConnectManaged:true`. These network interfaces give MSK Connect network access to resources in your Amazon VPC, such as an Apache Kafka cluster or a source or a sink.

You can't attach `KafkaConnectServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows MSK Connect to perform actions on your behalf.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface"
      ],
      "Resource": "arn:aws:ec2:*:*:network-interface/*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/AmazonMSKConnectManaged": "true"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": "AmazonMSKConnectManaged"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/*"
      ]
    }
  ]
}

```

```

},
{
  "Effect": "Allow",
  "Action": [
    "ec2:CreateTags"
  ],
  "Resource": "arn:aws:ec2:*:*:network-interface/*",
  "Condition": {
    "StringEquals": {
      "ec2:CreateAction": "CreateNetworkInterface"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:DescribeNetworkInterfaces",
    "ec2:CreateNetworkInterfacePermission",
    "ec2:AttachNetworkInterface",
    "ec2:DetachNetworkInterface",
    "ec2>DeleteNetworkInterface"
  ],
  "Resource": "arn:aws:ec2:*:*:network-interface/*",
  "Condition": {
    "StringEquals": {
      "ec2:ResourceTag/AmazonMSKConnectManaged": "true"
    }
  }
}
]
}

```

MSK Connect updates to AWS managed policies

View details about updates to AWS managed policies for MSK Connect since this service began tracking these changes.

Change	Description	Date
MSK Connect updated read-only policy	MSK Connect updated the AmazonMSKConnectRe	October 13, 2021

Change	Description	Date
	adOnlyAccess policy to remove the restrictions on listing operations.	
MSK Connect started tracking changes	MSK Connect started tracking changes for its AWS managed policies.	September 14, 2021

Use service-linked roles for MSK Connect

Amazon MSK Connect uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to MSK Connect. Service-linked roles are predefined by MSK Connect and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up MSK Connect easier because you don't have to manually add the necessary permissions. MSK Connect defines the permissions of its service-linked roles, and unless defined otherwise, only MSK Connect can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for MSK Connect

MSK Connect uses the service-linked role named **AWSServiceRoleForKafkaConnect** – Allows Amazon MSK Connect to access Amazon resources on your behalf.

The **AWSServiceRoleForKafkaConnect** service-linked role trusts the `kafkaconnect.amazonaws.com` service to assume the role.

For information about the permissions policy that the role uses, see [the section called “KafkaConnectServiceRolePolicy”](#).

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Creating a service-linked role for MSK Connect

You don't need to manually create a service-linked role. When you create a connector in the AWS Management Console, the AWS CLI, or the AWS API, MSK Connect creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a connector, MSK Connect creates the service-linked role for you again.

Editing a service-linked role for MSK Connect

MSK Connect does not allow you to edit the `AWSServiceRoleForKafkaConnect` service-linked role. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a service-linked role for MSK Connect

You can use the IAM console, the AWS CLI or the AWS API to manually delete the service-linked role. To do this, you must first manually delete all of your MSK Connect connectors, and then you can manually delete the role. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

Supported Regions for MSK Connect service-linked roles

MSK Connect supports using service-linked roles in all of the regions where the service is available. For more information, see [AWS Regions and Endpoints](#).

Enable internet access for Amazon MSK Connect

If your connector for Amazon MSK Connect needs access to the internet, we recommend that you use the following Amazon Virtual Private Cloud (VPC) settings to enable that access.

- Configure your connector with private subnets.

- Create a public [NAT gateway](#) or [NAT instance](#) for your VPC in a public subnet. For more information, see the [Connect subnets to the internet or other VPCs using NAT devices](#) page in the *Amazon Virtual Private Cloud User Guide*.
- Allow outbound traffic from your private subnets to your NAT gateway or instance.

Set up a NAT gateway for Amazon MSK Connect

The following steps show you how to set up a NAT gateway to enable internet access for a connector. You must complete these steps before you create a connector in a private subnet.

Complete prerequisites for setting up a NAT gateway

Make sure you have the following items.

- The ID of the Amazon Virtual Private Cloud (VPC) associated with your cluster. For example, `vpc-123456ab`.
- The IDs of the private subnets in your VPC. For example, `subnet-a1b2c3de`, `subnet-f4g5h6ij`, etc. You must configure your connector with private subnets.

Steps to enable internet access for your connector

To enable internet access for your connector

1. Open the Amazon Virtual Private Cloud console at <https://console.aws.amazon.com/vpc/>.
2. Create a public subnet for your NAT gateway with a descriptive name, and note the subnet ID. For detailed instructions, see [Create a subnet in your VPC](#).
3. Create an internet gateway so that your VPC can communicate with the internet, and note the gateway ID. Attach the internet gateway to your VPC. For instructions, see [Create and attach an internet gateway](#).
4. Provision a public NAT gateway so that hosts in your private subnets can reach your public subnet. When you create the NAT gateway, select the public subnet that you created earlier. For instructions, see [Create a NAT gateway](#).
5. Configure your route tables. You must have two route tables in total to complete this setup. You should already have a main route table that was automatically created at the same time as your VPC. In this step you create an additional route table for your public subnet.

- a. Use the following settings to modify your VPC's main route table so that your private subnets route traffic to your NAT gateway. For instructions, see [Work with route tables](#) in the *Amazon Virtual Private Cloud User Guide*.

Private MSKC route table

Property	Value
Name tag	We recommend that you give this route table a descriptive name tag to help you identify it. For example, Private MSKC .
Associated subnets	Your private subnets
A route to enable internet access for MSK Connect	<ul style="list-style-type: none">• Destination: 0.0.0.0/0• Target: Your NAT gateway ID. For example, <i>nat-12a345bc6789efg1h</i>.
A local route for internal traffic	<ul style="list-style-type: none">• Destination: 10.0.0.0/16. This value may differ depending on your VPC's CIDR block.• Target: Local

- b. Follow the instructions in [Create a custom route table](#) to create a route table for your public subnet. When you create the table, enter a descriptive name in the **Name tag** field to help you identify which subnet the table is associated with. For example, **Public MSKC**.
- c. Configure your **Public MSKC** route table using the following settings.

Property	Value
Name tag	Public MSKC or a different descriptive name that you choose
Associated subnets	Your public subnet with NAT gateway

Property	Value
A route to enable internet access for MSK Connect	<ul style="list-style-type: none">• Destination: 0.0.0.0/0• Target: Your internet gateway ID. For example, <i>igw-1a234bc5</i>.
A local route for internal traffic	<ul style="list-style-type: none">• Destination: 10.0.0.0/16. This value may differ depending on your VPC's CIDR block.• Target: Local

Understand private DNS hostnames

With Private DNS hostname support in MSK Connect, you can configure connectors to reference public or private domain names. Support depends on the DNS servers specified in the VPC *DHCP option set*.

A DHCP option set is a group of network configurations that EC2 instances use in a VPC to communicate over the VPC network. Each VPC has a default DHCP option set, but you can create a custom DHCP option set if you want instances in a VPC to use a different DNS server for domain name resolution, instead of the Amazon-provided DNS server. See [DHCP option sets in Amazon VPC](#).

Before the Private DNS resolution capability/feature was included with MSK Connect, connectors used the service VPC DNS resolvers for DNS queries from a customer connector. Connectors did not use the DNS servers defined in the customer VPC DHCP option sets for DNS resolution.

Connectors could only reference hostnames in customer connector configurations or plugins that were publicly resolvable. They couldn't resolve private hostnames defined in a privately-hosted zone or use DNS servers in another customer network.

Without Private DNS, customers who chose to make their databases, data warehouses, and systems like the Secrets Manager in their own VPC inaccessible to the internet, couldn't work with MSK connectors. Customers often use private DNS hostnames to comply with corporate security posture.

Configure a VPC DHCP option set for your connector

Connectors automatically use the DNS servers defined in their VPC DHCP option set when the connector is created. Before you create a connector, make sure that you configure the VPC DHCP option set for your connector's DNS hostname resolution requirements.

Connectors that you created before the Private DNS hostname feature was available in MSK Connect continue to use the previous DNS resolution configuration with no modification required.

If you need only publicly resolvable DNS hostname resolution in your connector, for easier setup we recommend using the default VPC of your account when you create the connector. See [Amazon DNS Server](#) in the *Amazon VPC User Guide* for more information on the Amazon-provided DNS server or Amazon Route 53 Resolver.

If you need to resolve private DNS hostnames, make sure the VPC that is passed during connector creation has its DHCP options set correctly configured. For more information, see [Work with DHCP option sets](#) in the *Amazon VPC User Guide*.

When you configure a DHCP option set for private DNS hostname resolution, ensure that the connector can reach the custom DNS servers that you configure in the DHCP option set. Otherwise, your connector creation will fail.

After you customize the VPC DHCP option set, connectors subsequently created in that VPC use the DNS servers that you specified in the option set. If you change the option set after you create a connector, the connector adopts the settings in the new option set within a couple of minutes.

Configure DNS attributes for your VPC

Make sure you have the VPC DNS attributes correctly configured as described in [DNS attributes in your VPC](#) and [DNS hostnames](#) in the *Amazon VPC User Guide*.

See [Resolving DNS queries between VPCs and your network](#) in the *Amazon Route 53 Developer Guide* for information on using inbound and outbound resolver endpoints to connect other networks to your VPC to work with your connector.

Handle connector creation failures

This section describes possible connector creation failures associated with DNS resolution and suggested actions to resolve the issues.

Failure	Suggested action
<p>Connector creation fails if a DNS resolution query fails, or if DNS servers are unreachable from the connector.</p>	<p>You can see connector creation failures due to unsuccessful DNS resolution queries in your CloudWatch logs, if you've configured these logs for your connector.</p> <p>Check the DNS server configurations and ensure network connectivity to the DNS servers from the connector.</p>
<p>If you change the DNS servers configuration in your VPC DHCP option set while a connector is running, DNS resolution queries from the connector can fail. If the DNS resolution fails, some of the connector tasks can enter a failed state.</p>	<p>You can see connector creation failures due to unsuccessful DNS resolution queries in your CloudWatch logs, if you've configured these logs for your connector.</p> <p>The failed tasks should automatically restart to bring the connector back up. If that does not happen, you can contact support to restart the failed tasks for their connector or you can recreate the connector.</p>

Security for MSK Connect

You can use an Interface VPC Endpoint, powered by AWS PrivateLink, to prevent traffic between your Amazon VPC and Amazon MSK-Connect compatible APIs from leaving the Amazon network. Interface VPC endpoints don't require an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. For more information, see [Use Amazon MSK APIs with Interface VPC Endpoints](#).

Logging for MSK Connect

MSK Connect can write log events that you can use to debug your connector. When you create a connector, you can specify zero or more of the following log destinations:

- **Amazon CloudWatch Logs:** You specify the log group to which you want MSK Connect to send your connector's log events. For information on how to create a log group, see [Create a log group](#) in the *CloudWatch Logs User Guide*.
- **Amazon S3:** You specify the S3 bucket to which you want MSK Connect to send your connector's log events. For information on how to create an S3 bucket, see [Creating a bucket](#) in the *Amazon S3 User Guide*.
- **Amazon Data Firehose:** You specify the delivery stream to which you want MSK Connect to send your connector's log events. For information on how to create a delivery stream, see [Creating an Amazon Data Firehose delivery stream](#) in the *Firehose User Guide*.

To learn more about setting up logging, see [Enabling logging from certain AWS services](#) in the *Amazon CloudWatch Logs User Guide*.

MSK Connect emits the following types of log events:

Level	Description
INFO	Runtime events of interest at startup and shutdown.
WARN	Runtime situations that aren't errors but are undesirable or unexpected.
FATAL	Severe errors that cause premature termination.
ERROR	Unexpected conditions and runtime errors that aren't fatal.

The following is an example of a log event sent to CloudWatch Logs:

```
[Worker-0bb8afa0b01391c41] [2021-09-06 16:02:54,151] WARN [Producer
  clientId=producer-1] Connection to node 1 (b-1.my-test-cluster.twwhtj.c2.kafka.us-
  east-1.amazonaws.com/INTERNAL_IP) could not be established. Broker may not be
  available. (org.apache.kafka.clients.NetworkClient:782)
```

Preventing secrets from appearing in connector logs

Note

Sensitive configuration values can appear in connector logs if a plugin does not define those values as secret. Kafka Connect treats undefined configuration values the same as any other plaintext value.

If your plugin defines a property as secret, Kafka Connect redacts the property's value from connector logs. For example, the following connector logs demonstrate that if a plugin defines `aws.secret.key` as a `PASSWORD` type, then its value is replaced with **[hidden]**.

```
2022-01-11T15:18:55.000+00:00 [Worker-05e6586a48b5f331b] [2022-01-11
15:18:55,150] INFO SecretsManagerConfigProviderConfig values:
2022-01-11T15:18:55.000+00:00 [Worker-05e6586a48b5f331b] aws.access.key =
my_access_key
2022-01-11T15:18:55.000+00:00 [Worker-05e6586a48b5f331b] aws.region = us-east-1
2022-01-11T15:18:55.000+00:00 [Worker-05e6586a48b5f331b] aws.secret.key
= [hidden]
2022-01-11T15:18:55.000+00:00 [Worker-05e6586a48b5f331b] secret.prefix =
2022-01-11T15:18:55.000+00:00 [Worker-05e6586a48b5f331b] secret.ttl.ms = 300000
2022-01-11T15:18:55.000+00:00 [Worker-05e6586a48b5f331b]
(com.github.jcustenborder.kafka.config.aws.SecretsManagerConfigProviderConfig:361)
```

To prevent secrets from appearing in connector log files, a plugin developer must use the Kafka Connect enum constant [ConfigDef.Type.PASSWORD](#) to define sensitive properties. When a property is type `ConfigDef.Type.PASSWORD`, Kafka Connect excludes its value from connector logs even if the value is sent as plaintext.

Monitoring MSK Connect

Monitoring is an important part of maintaining the reliability, availability, and performance of MSK Connect and your other AWS solutions. Amazon CloudWatch monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your connector, so that you can increase its capacity if needed. For more information, see the [Amazon CloudWatch User Guide](#).

You can use the following API operations:

- `DescribeConnectorOperation`: Monitor the status of connector update operations.
- `ListConnectorOperations`: Track previous updates run on your connector.

The following table shows the metrics that MSK Connect sends to CloudWatch under the `ConnectorName` dimension. MSK Connect delivers these metrics by default and at no additional cost. CloudWatch keeps these metrics for 15 months, so that you can access historical information and gain a better perspective on how your connectors are performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

MSK Connect metrics

Metric name	Description
<code>BytesInPerSec</code>	The total number of bytes received by the connector.
<code>BytesOutPerSec</code>	The total number of bytes delivered by the connector.
<code>CpuUtilization</code>	The percentage of CPU consumption by system and user.
<code>ErroredTaskCount</code>	The number of tasks that have errored out.
<code>MemoryUtilization</code>	The percentage of the total memory on a worker instance, not just the Java virtual machine (JVM) heap memory currently in use. JVM doesn't typically release memory back to the operational system. So, JVM heap size (<code>MemoryUtilization</code>) usually starts with a minimum heap size that incrementally increases to a stable maximum of about 80-90%. JVM heap usage might increase or decrease as the connector's actual memory usage changes.

Metric name	Description
RebalanceCompletedTotal	The total number of rebalances completed by this connector.
RebalanceTimeAvg	The average time in milliseconds spent by the connector on rebalancing.
RebalanceTimeMax	The maximum time in milliseconds spent by the connector on rebalancing.
RebalanceTimeSinceLast	The time in milliseconds since this connector completed the most recent rebalance.
RunningTaskCount	The running number of tasks in the connector.
SinkRecordReadRate	The average per-second number of records read from the Apache Kafka or Amazon MSK cluster.
SinkRecordSendRate	The average per-second number of records that are output from the transformations and sent to the destination. This number doesn't include filtered records.
SourceRecordPollRate	The average per-second number of records produced or polled.
SourceRecordWriteRate	The average per-second number of records output from the transformations and written to the Apache Kafka or Amazon MSK cluster.
TaskStartupAttemptsTotal	The total number of task startups that the connector has attempted. You can use this metric to identify anomalies in task startup attempts.

Metric name	Description
TaskStartupSuccessPercentage	The average percentage of successful task starts for the connector. You can use this metric to identify anomalies in task startup attempts.
WorkerCount	The number of workers that are running in the connector.

Examples to set up Amazon MSK Connect resources

This section includes examples to help you set up Amazon MSK Connect resources such as common third-party connectors and configuration providers.

Topics

- [Set up Amazon S3 sink connector](#)
- [Set up EventBridge Kafka sink connector for MSK Connect](#)
- [Use Debezium source connector with configuration provider](#)

Set up Amazon S3 sink connector

This example shows how to use the Confluent [Amazon S3 sink connector](#) and the AWS CLI to create an Amazon S3 sink connector in MSK Connect.

1. Copy the following JSON and paste it in a new file. Replace the placeholder strings with values that correspond to your Amazon MSK cluster's bootstrap servers connection string and the cluster's subnet and security group IDs. For information about how to set up a service execution role, see [the section called "IAM roles and policies"](#).

```
{
  "connectorConfiguration": {
    "connector.class": "io.confluent.connect.s3.S3SinkConnector",
    "s3.region": "us-east-1",
    "format.class": "io.confluent.connect.s3.format.json.JsonFormat",
    "flush.size": "1",
    "schema.compatibility": "NONE",
```

```

    "topics": "my-test-topic",
    "tasks.max": "2",
    "partitioner.class":
"io.confluent.connect.storage.partitioners.DefaultPartitioner",
    "storage.class": "io.confluent.connect.s3.storage.S3Storage",
    "s3.bucket.name": "amzn-s3-demo-bucket"
  },
  "connectorName": "example-S3-sink-connector",
  "kafkaCluster": {
    "apacheKafkaCluster": {
      "bootstrapServers": "<cluster-bootstrap-servers-string>",
      "vpc": {
        "subnets": [
          "<cluster-subnet-1>",
          "<cluster-subnet-2>",
          "<cluster-subnet-3>"
        ],
        "securityGroups": ["<cluster-security-group-id>"]
      }
    }
  },
  "capacity": {
    "provisionedCapacity": {
      "mcuCount": 2,
      "workerCount": 4
    }
  },
  "kafkaConnectVersion": "2.7.1",
  "serviceExecutionRoleArn": "<arn-of-a-role-that-msk-connect-can-assume>",
  "plugins": [
    {
      "customPlugin": {
        "customPluginArn": "<arn-of-custom-plugin-that-contains-connector-
code>",
        "revision": 1
      }
    }
  ],
  "kafkaClusterEncryptionInTransit": {"encryptionType": "PLAINTEXT"},
  "kafkaClusterClientAuthentication": {"authenticationType": "NONE"}
}

```

2. Run the following AWS CLI command in the folder where you saved the JSON file in the previous step.

```
aws kafkaconnect create-connector --cli-input-json file://connector-info.json
```

The following is an example of the output that you get when you run the command successfully.

```
{
  "ConnectorArn": "arn:aws:kafkaconnect:us-east-1:123450006789:connector/example-S3-sink-connector/abc12345-abcd-4444-a8b9-123456f513ed-2",
  "ConnectorState": "CREATING",
  "ConnectorName": "example-S3-sink-connector"
}
```

Set up EventBridge Kafka sink connector for MSK Connect

This topic shows you how to set up the [EventBridge Kafka sink connector](#) for MSK Connect. This connector lets you send events from your MSK cluster to EventBridge [event buses](#). This topic describes the process for creating the required resources and configuring the connector to enable seamless data flow between Kafka and EventBridge.

Topics

- [Prerequisites](#)
- [Set up the resources required for MSK Connect](#)
- [Create the connector](#)
- [Send messages to Kafka](#)

Prerequisites

Before deploying the connector, make sure that you have the following resources:

- **Amazon MSK cluster:** An active MSK cluster to produce and consume Kafka messages.
- **Amazon EventBridge event bus:** An EventBridge event bus to receive events from the Kafka topics.
- **IAM roles:** Create IAM roles with the necessary permissions for MSK Connect and the EventBridge connector.

- [Access to the public internet](#) from MSK Connect or a [VPC interface endpoint](#) for EventBridge created in the VPC and subnet of your MSK cluster. This helps you avoid traversing the public internet and without requiring NAT gateways.
- A [client machine](#), such as an Amazon EC2 instance or [AWS CloudShell](#), to create topics and send records to Kafka.

Set up the resources required for MSK Connect

You create an IAM role for the connector, and then you create the connector. You also create an EventBridge rule to filter Kafka events sent to the EventBridge event bus.

Topics

- [IAM role for the connector](#)
- [An EventBridge rule for incoming events](#)

IAM role for the connector

The IAM role that you associate with the connector must have the [PutEvents](#) permission to allow sending events to EventBridge. The following IAM policy example grants you the permission to send events to an event bus named `example-event-bus`. Make sure that you replace the resource ARN in the following example with the ARN of your event bus.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutEvents"
      ],
      "Resource": "arn:aws:events:us-east-1:123456789012:event-bus/example-event-bus"
    }
  ]
}
```

In addition, you must make sure that your IAM role for the connector contains the following trust policy.

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "kafkaconnect.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
}

```

An EventBridge rule for incoming events

You create [rules](#) that match incoming events with event data criteria, known as [event pattern](#). With an event pattern, you can define the criteria to filter incoming events, and determine which events should trigger a particular rule and subsequently be routed to a designated [target](#). The following example of an event pattern matches Kafka events sent to the EventBridge event bus.

```

{
  "detail": {
    "topic": ["msk-eventbridge-tutorial"]
  }
}

```

The following is an example of an event sent from Kafka to EventBridge using the Kafka sink connector.

```

{
  "version": "0",
  "id": "dbc1c73a-c51d-0c0e-ca61-ab9278974c57",
  "account": "123456789012",
  "time": "2025-03-26T10:15:00Z",
  "region": "us-east-1",
  "detail-type": "msk-eventbridge-tutorial",
  "source": "kafka-connect.msk-eventbridge-tutorial",
  "resources": [],
  "detail": {
    "topic": "msk-eventbridge-tutorial",
    "partition": 0,
    "offset": 0,
    "timestamp": 1742984100000,

```

```
"timestampType": "CreateTime",
"headers": [],
"key": "order-1",
"value": {
  "orderItems": [
    "item-1",
    "item-2"
  ],
  "orderCreatedTime": "Wed Mar 26 10:15:00 UTC 2025"
}
}
```

In the EventBridge console, [create a rule](#) on the event bus using this example pattern and specify a target, such as a CloudWatch Logs group. The EventBridge console will automatically configure the necessary access policy for the CloudWatch Logs group.

Create the connector

In the following section, you create and deploy the [EventBridge Kafka sink connector](#) using the AWS Management Console.

Topics

- [Step 1: Download the connector](#)
- [Step 2: Create an Amazon S3 bucket](#)
- [Step 3: Create a plugin in MSK Connect](#)
- [Step 4: Create the connector](#)

Step 1: Download the connector

Download the latest EventBridge connector sink JAR from the [GitHub releases page](#) for the EventBridge Kafka connector. For example, to download the version v1.4.1, choose the JAR file link, `kafka-eventbridge-sink-with-dependencies.jar`, to download the connector. Then, save the file to a preferred location on your machine.

Step 2: Create an Amazon S3 bucket

1. To store the JAR file in Amazon S3 for use with MSK Connect, open the AWS Management Console, and then choose Amazon S3.

2. In the Amazon S3 console, choose **Create bucket**, and enter a unique bucket name. For example, **amzn-s3-demo-bucket1-eb-connector**.
3. Choose an appropriate Region for your Amazon S3 bucket. Make sure that it matches the Region where your MSK cluster is deployed.
4. For **Bucket settings**, keep the default selections or adjust as needed.
5. Choose **Create bucket**
6. Upload the JAR file to the Amazon S3 bucket.

Step 3: Create a plugin in MSK Connect

1. Open the AWS Management Console, and then navigate to **MSK Connect**.
2. In the left navigation pane, choose **Custom plugins**.
3. Choose **Create plugin**, and then enter a **Plugin name**. For example, **eventbridge-sink-plugin**.
4. For **Custom plugin location**, paste the **S3 object URL**.
5. Add an optional description for the plugin.
6. Choose **Create plugin**.

After the plugin is created, you can use it to configure and deploy the EventBridge Kafka connector in MSK Connect.

Step 4: Create the connector

Before creating the connector, we recommend to create the required Kafka topic to avoid connector errors. To create the topic, use your client machine.

1. In the left pane of the MSK console, choose **Connectors**, and then choose **Create connector**.
2. In the list of plugins, choose **eventbridge-sink-plugin**, then choose **Next**.
3. For the connector name, enter **EventBridgeSink**.
4. In the list of clusters, choose your MSK cluster.
5. Copy the following configuration for the connector and paste it into the **Connector configuration** field

Replace the placeholders in the following configuration, as required.

- Remove `aws.eventbridge.endpoint.uri` if your MSK cluster has public internet access.
- If you use PrivateLink to securely connect from MSK to EventBridge, replace the DNS part after `https://` with the correct private DNS name of the (optional) VPC interface endpoint for EventBridge that you created earlier.
- Replace the EventBridge event bus ARN in the following configuration with the ARN of your event bus.
- Update any Region-specific values.

```
{
  "connector.class":
    "software.amazon.event.kafkaconnector.EventBridgeSinkConnector",
  "aws.eventbridge.connector.id": "msk-eventbridge-tutorial",
  "topics": "msk-eventbridge-tutorial",
  "tasks.max": "1",
  "aws.eventbridge.endpoint.uri": "https://events.us-east-1.amazonaws.com",
  "aws.eventbridge.eventbus.arn": "arn:aws:events:us-east-1:123456789012:event-bus/
example-event-bus",
  "value.converter.schemas.enable": "false",
  "value.converter": "org.apache.kafka.connect.json.JsonConverter",
  "aws.eventbridge.region": "us-east-1",
  "auto.offset.reset": "earliest",
  "key.converter": "org.apache.kafka.connect.storage.StringConverter"
}
```

For more information about connector configuration, see [eventbridge-kafka-connector](#).

If needed, change the settings for workers and autoscaling. We also recommend to use the latest available (recommended) Apache Kafka Connect version from the dropdown. Under **Access permissions**, use the role created earlier. We also recommend to enable logging to CloudWatch for observability and troubleshooting. Adjust the other optional settings, such as tags, based on your needs. Then, deploy the connector and wait for the status to enter Running state.

Send messages to Kafka

You can configure message encodings, such as Apache Avro and JSON, by specifying different converters using the `value.converter` and, optionally, `key.converter` settings available in Kafka Connect.

The [connector example](#) in this topic is configured to work with JSON-encoded messages, as indicated by the use of `org.apache.kafka.connect.json.JsonConverter` for `value.converter`. When the connector is in Running state, send records to the `msk-eventbridge-tutorial` Kafka topic from your client machine.

Use Debezium source connector with configuration provider

This example shows how to use the Debezium MySQL connector plugin with a MySQL-compatible [Amazon Aurora](#) database as the source. In this example, we also set up the open-source [AWS Secrets Manager Config Provider](#) to externalize database credentials in AWS Secrets Manager. To learn more about configuration providers, see [Tutorial: Externalizing sensitive information using config providers](#).

Important

The Debezium MySQL connector plugin [supports only one task](#) and does not work with autoscaled capacity mode for Amazon MSK Connect. You should instead use provisioned capacity mode and set `workerCount` equal to one in your connector configuration. To learn more about the capacity modes for MSK Connect, see [Understand connector capacity](#).

Complete prerequisites to use Debezium source connector

Your connector must be able to access the internet so that it can interact with services such as AWS Secrets Manager that are outside of your Amazon Virtual Private Cloud. The steps in this section help you complete the following tasks to enable internet access.

- Set up a public subnet that hosts a NAT gateway and routes traffic to an internet gateway in your VPC.
- Create a default route that directs your private subnet traffic to your NAT gateway.

For more information, see [Enable internet access for Amazon MSK Connect](#).

Prerequisites

Before you can enable internet access, you need the following items:

- The ID of the Amazon Virtual Private Cloud (VPC) associated with your cluster. For example, *vpc-123456ab*.
- The IDs of the private subnets in your VPC. For example, *subnet-a1b2c3de*, *subnet-f4g5h6ij*, etc. You must configure your connector with private subnets.

To enable internet access for your connector

1. Open the Amazon Virtual Private Cloud console at <https://console.aws.amazon.com/vpc/>.
2. Create a public subnet for your NAT gateway with a descriptive name, and note the subnet ID. For detailed instructions, see [Create a subnet in your VPC](#).
3. Create an internet gateway so that your VPC can communicate with the internet, and note the gateway ID. Attach the internet gateway to your VPC. For instructions, see [Create and attach an internet gateway](#).
4. Provision a public NAT gateway so that hosts in your private subnets can reach your public subnet. When you create the NAT gateway, select the public subnet that you created earlier. For instructions, see [Create a NAT gateway](#).
5. Configure your route tables. You must have two route tables in total to complete this setup. You should already have a main route table that was automatically created at the same time as your VPC. In this step you create an additional route table for your public subnet.
 - a. Use the following settings to modify your VPC's main route table so that your private subnets route traffic to your NAT gateway. For instructions, see [Work with route tables](#) in the *Amazon Virtual Private Cloud User Guide*.

Private MSKC route table

Property	Value
Name tag	We recommend that you give this route table a descriptive name tag to help you identify it. For example, Private MSKC .
Associated subnets	Your private subnets

Property	Value
A route to enable internet access for MSK Connect	<ul style="list-style-type: none"> • Destination: 0.0.0.0/0 • Target: Your NAT gateway ID. For example, <i>nat-12a345bc6789efg1h</i>.
A local route for internal traffic	<ul style="list-style-type: none"> • Destination: 10.0.0.0/16. This value may differ depending on your VPC's CIDR block. • Target: Local

- b. Follow the instructions in [Create a custom route table](#) to create a route table for your public subnet. When you create the table, enter a descriptive name in the **Name tag** field to help you identify which subnet the table is associated with. For example, **Public MSKC**.
- c. Configure your **Public MSKC** route table using the following settings.

Property	Value
Name tag	Public MSKC or a different descriptive name that you choose
Associated subnets	Your public subnet with NAT gateway
A route to enable internet access for MSK Connect	<ul style="list-style-type: none"> • Destination: 0.0.0.0/0 • Target: Your internet gateway ID. For example, <i>igw-1a234bc5</i>.
A local route for internal traffic	<ul style="list-style-type: none"> • Destination: 10.0.0.0/16. This value may differ depending on your VPC's CIDR block. • Target: Local

Now that you have enabled internet access for Amazon MSK Connect you are ready to create a connector.

Create a Debezium source connector

This procedure describes how to create a Debezium source connector.

1. Create a custom plugin

- a. Download the MySQL connector plugin for the latest stable release from the [Debezium](#) site. Make a note of the Debezium release version you download (version 2.x, or the older series 1.x). Later in this procedure, you'll create a connector based on your Debezium version.
- b. Download and extract the [AWS Secrets Manager Config Provider](#).
- c. Place the following archives into the same directory:
 - The `debezium-connector-mysql` folder
 - The `jcusten-border-kafka-config-provider-aws-0.1.1` folder
- d. Compress the directory that you created in the previous step into a ZIP file and then upload the ZIP file to an S3 bucket. For instructions, see [Uploading objects](#) in the *Amazon S3 User Guide*.
- e. Copy the following JSON and paste it in a file. For example, `debezium-source-custom-plugin.json`. Replace `<example-custom-plugin-name>` with the name that you want the plugin to have, `<amzn-s3-demo-bucket-arn>` with the ARN of the Amazon S3 bucket where you uploaded the ZIP file, and `<file-key-of-ZIP-object>` with the file key of the ZIP object that you uploaded to S3.

```
{
  "name": "<example-custom-plugin-name>",
  "contentType": "ZIP",
  "location": {
    "s3Location": {
      "bucketArn": "<amzn-s3-demo-bucket-arn>",
      "fileKey": "<file-key-of-ZIP-object>"
    }
  }
}
```

- f. Run the following AWS CLI command from the folder where you saved the JSON file to create a plugin.

```
aws kafkaconnect create-custom-plugin --cli-input-json file://<debezium-source-  
custom-plugin.json>
```

You should see output similar to the following example.

```
{  
  "CustomPluginArn": "arn:aws:kafkaconnect:us-east-1:012345678901:custom-  
plugin/example-custom-plugin-name/abcd1234-a0b0-1234-c1-12345678abcd-1",  
  "CustomPluginState": "CREATING",  
  "Name": "example-custom-plugin-name",  
  "Revision": 1  
}
```

- g. Run the following command to check the plugin state. The state should change from CREATING to ACTIVE. Replace the ARN placeholder with the ARN that you got in the output of the previous command.

```
aws kafkaconnect describe-custom-plugin --custom-plugin-arn "<arn-of-your-  
custom-plugin>"
```

2. Configure AWS Secrets Manager and create a secret for your database credentials

- a. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
- b. Create a new secret to store your database sign-in credentials. For instructions, see [Create a secret](#) in the *AWS Secrets Manager User Guide*.
- c. Copy your secret's ARN.
- d. Add the Secrets Manager permissions from the following example policy to your [Understand service execution role](#). Replace `<arn:aws:secretsmanager:us-east-1:123456789000:secret:MySecret-1234>` with the ARN of your secret.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "secretsmanager:GetResourcePolicy",  
        "secretsmanager:GetSecretValue",  
        "secretsmanager:DescribeSecret",
```

```

        "secretsmanager:ListSecretVersionIds"
    ],
    "Resource": [
        "<arn:aws:secretsmanager:us-east-1:123456789000:secret:MySecret-1234>"
    ]
  }
]
}

```

For instructions on how to add IAM permissions, see [Adding and removing IAM identity permissions](#) in the *IAM User Guide*.

3. Create a custom worker configuration with information about your configuration provider

- a. Copy the following worker configuration properties into a file, replacing the placeholder strings with values that correspond to your scenario. To learn more about the configuration properties for the AWS Secrets Manager Config Provider, see [SecretsManagerConfigProvider](#) in the plugin's documentation.

```

key.converter=<org.apache.kafka.connect.storage.StringConverter>
value.converter=<org.apache.kafka.connect.storage.StringConverter>
config.providers.secretManager.class=com.github.jcustenborder.kafka.config.aws.SecretsM
config.providers=secretManager
config.providers.secretManager.param.aws.region=<us-east-1>

```

- b. Run the following AWS CLI command to create your custom worker configuration.

Replace the following values:

- *<my-worker-config-name>* - a descriptive name for your custom worker configuration
- *<encoded-properties-file-content-string>* - a base64-encoded version of the plaintext properties that you copied in the previous step

```

aws kafkaconnect create-worker-configuration --name <my-worker-config-name> --
properties-file-content <encoded-properties-file-content-string>

```

4. Create a connector

- a. Copy the following JSON that corresponds to your Debezium version (2.x or 1.x) and paste it in a new file. Replace the *<placeholder>* strings with values that correspond to your

scenario. For information about how to set up a service execution role, see [the section called "IAM roles and policies"](#).

Note that the configuration uses variables like

`${secretManager:MySecret-1234:dbusername}` instead of plaintext to specify database credentials. Replace *MySecret-1234* with the name of your secret and then include the name of the key that you want to retrieve. You must also replace *<arn-of-config-provider-worker-configuration>* with the ARN of your custom worker configuration.

Debezium 2.x

For Debezium 2.x versions, copy the following JSON and paste it in a new file. Replace the *<placeholder>* strings with values that correspond to your scenario.

```
{
  "connectorConfiguration": {
    "connector.class": "io.debezium.connector.mysql.MySqlConnector",
    "tasks.max": "1",
    "database.hostname": "<aurora-database-writer-instance-endpoint>",
    "database.port": "3306",
    "database.user": "<${secretManager:MySecret-1234:dbusername}>",
    "database.password": "<${secretManager:MySecret-1234:dbpassword}>",
    "database.server.id": "123456",
    "database.include.list": "<list-of-databases-hosted-by-specified-server>",
    "topic.prefix": "<logical-name-of-database-server>",
    "schema.history.internal.kafka.topic": "<kafka-topic-used-by-debezium-to-track-schema-changes>",
    "schema.history.internal.kafka.bootstrap.servers": "<cluster-bootstrap-servers-string>",
    "schema.history.internal.consumer.security.protocol": "SASL_SSL",
    "schema.history.internal.consumer.sasl.mechanism": "AWS_MSK_IAM",
    "schema.history.internal.consumer.sasl.jaas.config":
    "software.amazon.msk.auth.iam.IAMLoginModule required;",
    "schema.history.internal.consumer.sasl.client.callback.handler.class":
    "software.amazon.msk.auth.iam.IAMClientCallbackHandler",
    "schema.history.internal.producer.security.protocol": "SASL_SSL",
    "schema.history.internal.producer.sasl.mechanism": "AWS_MSK_IAM",
    "schema.history.internal.producer.sasl.jaas.config":
    "software.amazon.msk.auth.iam.IAMLoginModule required;",
    "schema.history.internal.producer.sasl.client.callback.handler.class":
    "software.amazon.msk.auth.iam.IAMClientCallbackHandler",
```

```

    "include.schema.changes": "true"
  },
  "connectorName": "example-Debezium-source-connector",
  "kafkaCluster": {
    "apacheKafkaCluster": {
      "bootstrapServers": "<cluster-bootstrap-servers-string>",
      "vpc": {
        "subnets": [
          "<cluster-subnet-1>",
          "<cluster-subnet-2>",
          "<cluster-subnet-3>"
        ],
        "securityGroups": ["<id-of-cluster-security-group>"]
      }
    }
  },
  "capacity": {
    "provisionedCapacity": {
      "mcuCount": 2,
      "workerCount": 1
    }
  },
  "kafkaConnectVersion": "2.7.1",
  "serviceExecutionRoleArn": "<arn-of-service-execution-role-that-msk-
connect-can-assume>",
  "plugins": [{
    "customPlugin": {
      "customPluginArn": "<arn-of-msk-connect-plugin-that-contains-connector-
code>",
      "revision": 1
    }
  }],
  "kafkaClusterEncryptionInTransit": {
    "encryptionType": "TLS"
  },
  "kafkaClusterClientAuthentication": {
    "authenticationType": "IAM"
  },
  "workerConfiguration": {
    "workerConfigurationArn": "<arn-of-config-provider-worker-configuration>",
    "revision": 1
  }
}

```

Debezium 1.x

For Debezium 1.x versions, copy the following JSON and paste it in a new file. Replace the *<placeholder>* strings with values that correspond to your scenario.

```
{
  "connectorConfiguration": {
    "connector.class": "io.debezium.connector.mysql.MySqlConnector",
    "tasks.max": "1",
    "database.hostname": "<aurora-database-writer-instance-endpoint>",
    "database.port": "3306",
    "database.user": "<${secretManager:MySecret-1234:dbusername}>",
    "database.password": "<${secretManager:MySecret-1234:dbpassword}>",
    "database.server.id": "123456",
    "database.server.name": "<logical-name-of-database-server>",
    "database.include.list": "<list-of-databases-hosted-by-specified-server>",
    "database.history.kafka.topic": "<kafka-topic-used-by-debezium-to-track-schema-changes>",
    "database.history.kafka.bootstrap.servers": "<cluster-bootstrap-servers-string>",
    "database.history.consumer.security.protocol": "SASL_SSL",
    "database.history.consumer.sasl.mechanism": "AWS_MSK_IAM",
    "database.history.consumer.sasl.jaas.config":
    "software.amazon.msk.auth.iam.IAMLoginModule required;",
    "database.history.consumer.sasl.client.callback.handler.class":
    "software.amazon.msk.auth.iam.IAMClientCallbackHandler",
    "database.history.producer.security.protocol": "SASL_SSL",
    "database.history.producer.sasl.mechanism": "AWS_MSK_IAM",
    "database.history.producer.sasl.jaas.config":
    "software.amazon.msk.auth.iam.IAMLoginModule required;",
    "database.history.producer.sasl.client.callback.handler.class":
    "software.amazon.msk.auth.iam.IAMClientCallbackHandler",
    "include.schema.changes": "true"
  },
  "connectorName": "example-Debezium-source-connector",
  "kafkaCluster": {
    "apacheKafkaCluster": {
      "bootstrapServers": "<cluster-bootstrap-servers-string>",
      "vpc": {
        "subnets": [
          "<cluster-subnet-1>",
          "<cluster-subnet-2>",
          "<cluster-subnet-3>"
        ]
      }
    }
  }
}
```

```

    ],
    "securityGroups": ["<id-of-cluster-security-group>"]
  }
},
"capacity": {
  "provisionedCapacity": {
    "mcuCount": 2,
    "workerCount": 1
  }
},
"kafkaConnectVersion": "2.7.1",
"serviceExecutionRoleArn": "<arn-of-service-execution-role-that-msk-
connect-can-assume>",
"plugins": [{
  "customPlugin": {
    "customPluginArn": "<arn-of-msk-connect-plugin-that-contains-connector-
code>",
    "revision": 1
  }
}],
"kafkaClusterEncryptionInTransit": {
  "encryptionType": "TLS"
},
"kafkaClusterClientAuthentication": {
  "authenticationType": "IAM"
},
"workerConfiguration": {
  "workerConfigurationArn": "<arn-of-config-provider-worker-configuration>",
  "revision": 1
}
}

```

- b. Run the following AWS CLI command in the folder where you saved the JSON file in the previous step.

```
aws kafkaconnect create-connector --cli-input-json file://connector-info.json
```

The following is an example of the output that you get when you run the command successfully.

```
{
```

```
"ConnectorArn": "arn:aws:kafkaconnect:us-east-1:123450006789:connector/
example-Debezium-source-connector/abc12345-abcd-4444-a8b9-123456f513ed-2",
"ConnectorState": "CREATING",
"ConnectorName": "example-Debezium-source-connector"
}
```

Update a Debezium connector configuration

To update the configuration of the Debezium connector, follow these steps:

1. Copy the following JSON and paste it to a new file. Replace the <placeholder> strings with values that correspond to your scenario.

```
{
  "connectorArn": <connector_arn>,
  "connectorConfiguration": <new_configuration_in_json>,
  "currentVersion": <current_version>
}
```

2. Run the following AWS CLI command in the folder where you saved the JSON file in the previous step.

```
aws kafkaconnect update-connector --cli-input-json file://connector-info.json
```

The following is an example of the output when you run the command successfully.

```
{
  "connectorArn": "arn:aws:kafkaconnect:us-east-1:123450006789:connector/example-
Debezium-source-connector/abc12345-abcd-4444-a8b9-123456f513ed-2",
  "connectorOperationArn": "arn:aws:kafkaconnect:us-
east-1:123450006789:connector-operation/example-Debezium-source-connector/abc12345-
abcd-4444-a8b9-123456f513ed-2/41b6ad56-3184-479b-850a-a8bedd5a02f3",
  "connectorState": "UPDATING"
}
```

3. You can now run the following command to monitor the current state of the operation:

```
aws kafkaconnect describe-connector-operation --connector-operation-arn
<operation_arn>
```

For a Debezium connector example with detailed steps, see [Introducing Amazon MSK Connect - Stream Data to and from Your Apache Kafka Clusters Using Managed Connectors](#).

Migrate to Amazon MSK Connect

This section describes how to migrate your Apache Kafka connector application to Amazon Managed Streaming for Apache Kafka Connect (Amazon MSK Connect). To know more about the benefits of migrating to Amazon MSK Connect, see [???](#).

This section also describes the state management topics used by Kafka Connect and Amazon MSK Connect and covers procedures for migrating source and sink connectors.

Understand internal topics used by Kafka Connect

An Apache Kafka Connect application that's running in distributed mode stores its state by using internal topics in the Kafka cluster and group membership. The following are the configuration values that correspond to the internal topics that are used for Kafka Connect applications:

- Configuration topic, specified through `config.storage.topic`

In the configuration topic, Kafka Connect stores the configuration of all the connectors and tasks that have been started by users. Each time users update the configuration of a connector or when a connector requests a reconfiguration (for example, the connector detects that it can start more tasks), a record is emitted to this topic. This topic is compaction enabled, so it always keeps the last state for each entity.

- Offsets topic, specified through `offset.storage.topic`

In the offsets topic, Kafka Connect stores the offsets of the source connectors. Like the configuration topic, the offsets topic is compaction enabled. This topic is used to write the source positions only for source connectors that produce data to Kafka from external systems. Sink connectors, which read data from Kafka and send to external systems, store their consumer offsets by using regular Kafka consumer groups.

- Status topic, specified through `status.storage.topic`

In the status topic, Kafka Connect stores the current state of connectors and tasks. This topic is used as the central place for the data that is queried by users of the REST API. This topic allows users to query any worker and still get the status of all running plugins. Like the configuration and offsets topics, the status topic is also compaction enabled.

In addition to these topics, Kafka Connect makes extensive use of Kafka's group membership API. The groups are named after the connector name. For example, for a connector named `file-sink`, the group is named `connect-file-sink`. Each consumer in the group provides records to a single task. These groups and their offsets can be retrieved by using regular consumer groups tools, such as `Kafka-consumer-group.sh`. For each sink connector, the Connect runtime runs a regular consumer group that extracts records from Kafka.

State management of Amazon MSK Connect applications

By default, Amazon MSK Connect creates three separate topics in the Kafka cluster for each Amazon MSK Connector to store the connector's configuration, offset, and status. The default topic names are structured as follows:

- `__msk_connect_configs_connector-name_connector-id`
- `__msk_connect_status_connector-name_connector-id`
- `__msk_connect_offsets_connector-name_connector-id`

Note

To provide the offset continuity between source connectors, you can use an offset storage topic of your choice, instead of the default topic. Specifying an offset storage topic helps you accomplish tasks like creating a source connector that resumes reading from the last offset of a previous connector. To specify an offset storage topic, supply a value for the [offset.storage.topic](#) property in the Amazon MSK Connect worker configuration before creating the connector.

Migrate source connectors to Amazon MSK Connect

Source connectors are Apache Kafka Connect applications that import records from external systems into Kafka. This section describes the process for migrating Apache Kafka Connect source connector applications that are running on-premises or self-managed Kafka Connect clusters that are running on AWS to Amazon MSK Connect.

The Kafka Connect source connector application stores offsets in a topic that's named with the value that's set for the config property `offset.storage.topic`. The following are the sample offset messages for a JDBC connector that's running two tasks that import data from two different

tables named `movies` and `shows`. The most recent row imported from the table `movies` has a primary ID of 18343. The most recent row imported from the `shows` table has a primary ID of 732.

```
[{"jdbcsource",{"protocol":"1","table":"sample.movies"}} {"incrementing":18343}
[{"jdbcsource",{"protocol":"1","table":"sample.shows"}} {"incrementing":732}
```

To migrate source connectors to Amazon MSK Connect, do the following:

1. Create an Amazon MSK Connect [custom plugin](#) by pulling connector libraries from your on-premises or self-managed Kafka Connect cluster.
2. Create Amazon MSK Connect [worker properties](#) and set the properties `key.converter`, `value.converter`, and `offset.storage.topic` to the same values that are set for the Kafka connector that's running in your existing Kafka Connect cluster.
3. Pause the connector application on the existing cluster by making a PUT `/connectors/connector-name/pause` request on the existing Kafka Connect cluster.
4. Make sure that all of the connector application's tasks are completely stopped. You can stop the tasks either by making a GET `/connectors/connector-name/status` request on the existing Kafka Connect cluster or by consuming the messages from the topic name that's set for the property `status.storage.topic`.
5. Get the connector configuration from the existing cluster. You can get the connector configuration either by making a GET `/connectors/connector-name/config/` request on the existing cluster or by consuming the messages from the topic name that's set for the property `config.storage.topic`.
6. Create a new [Amazon MSK Connector](#) with the same name as an existing cluster. Create this connector by using the connector custom plugin that you created in step 1, the worker properties that you created in step 2, and the connector configuration that you extracted in step 5.
7. When the Amazon MSK Connector status is active, view the logs to verify that the connector has started importing data from the source system.
8. Delete the connector in the existing cluster by making a DELETE `/connectors/connector-name` request.

Migrate sink connectors to Amazon MSK Connect

Sink connectors are Apache Kafka Connect applications that export data from Kafka to external systems. This section describes the process for migrating Apache Kafka Connect sink connector applications that are running on-premises or self-managed Kafka Connect clusters that are running on AWS to Amazon MSK Connect.

Kafka Connect sink connectors use the Kafka group membership API and store offsets in the same `__consumer__offset` topics as a typical consumer application. This behavior simplifies migration of the sink connector from a self-managed cluster to Amazon MSK Connect.

To migrate sink connectors to Amazon MSK Connect, do the following:

1. Create an Amazon MSK Connect [custom plugin](#) by pulling connector libraries from your on-premises or self-managed Kafka Connect cluster.
2. Create Amazon MSK Connect [worker properties](#) and set the properties `key.converter` and `value.converter` to the same values that are set for the Kafka connector that's running in your existing Kafka Connect cluster.
3. Pause the connector application on your existing cluster by making a PUT `/connectors/connector-name/pause` request on the existing Kafka Connect cluster.
4. Make sure that all of the connector application's tasks are completely stopped. You can stop the tasks either by making a GET `/connectors/connector-name/status` request on the existing Kafka Connect cluster, or by consuming the messages from the topic name that's set for the property `status.storage.topic`.
5. Get the connector configuration from the existing cluster. You can get the connector configuration either by making a GET `/connectors/connector-name/config` request on the existing cluster, or by consuming the messages from the topic name that's set for the property `config.storage.topic`.
6. Create a new [Amazon MSK Connector](#) with same name as the existing cluster. Create this connector by using the connector custom plugin that you created in step 1, the worker properties that you created in step 2, and the connector configuration that you extracted in step 5.
7. When the Amazon MSK Connector status is active, view the logs to verify that the connector has started importing data from the source system.
8. Delete the connector in the existing cluster by making a DELETE `/connectors/connector-name` request.

Troubleshoot issues in Amazon MSK Connect

The following information can help you troubleshoot problems that you might have while using MSK Connect. You can also post your issue to the [AWS re:Post](#).

Connector is unable to access resources hosted on the public internet

See [Enabling internet access for Amazon MSK Connect](#).

Connector's number of running tasks is not equal to the number of tasks specified in tasks.max

Here are some reasons a connector may use fewer tasks than the specified tasks.max configuration:

- Some connector implementations limit the number of tasks they can be used. For example, the Debezium connector for MySQL is limited to using a single task.
- When using autoscaled capacity mode, Amazon MSK Connect overrides a connector's tasks.max property with a value that is proportional to the number of workers running in the connector and the number of MCUs per worker.
- For sink connectors, the level of parallelism (number of tasks) cannot be more than the number of topic partitions. While you can set the tasks.max larger than that, a single partition is never processed by more than a single task at a time.
- In Kafka Connect 2.7.x, the default consumer partition assignor is RangeAssignor. The behavior of this assignor is to give the first partition of every topic to a single consumer, the second partition of every topic to a single consumer, etc. This means that the maximum number of active tasks for a sink connector using RangeAssignor is equal to the maximum number of partitions in any single topic being consumed. If this doesn't work for your use case, you should [create a Worker Configuration](#) in which the consumer.partition.assignment.strategy property is set to a more suitable consumer partition assignor. See [Kafka 2.7 Interface ConsumerPartitionAssignor: All Known Implementing Classes](#).

What is Amazon MSK Replicator?

Amazon MSK Replicator is an Amazon MSK feature that enables you to reliably replicate data across Amazon MSK clusters in different or the same AWS region(s). With MSK Replicator, you can easily build regionally resilient streaming applications for increased availability and business continuity. MSK Replicator provides automatic asynchronous replication across MSK clusters, eliminating the need to write custom code, manage infrastructure, or setup cross-region networking.

MSK Replicator automatically scales the underlying resources so that you can replicate data on-demand without having to monitor or scale capacity. MSK Replicator also replicates the necessary Kafka metadata including topic configurations, Access Control Lists (ACLs), and consumer group offsets. If an unexpected event occurs in a region, you can failover to the other AWS region and seamlessly resume processing.

MSK Replicator supports both cross-region replication (CRR) and same-region replication (SRR). In cross-region replication, the source and target MSK clusters are in different AWS Regions. In same-region replication, both the source and target MSK clusters are in the same AWS Region. You need to create source and target MSK clusters before using them with MSK Replicator.

Note

MSK Replicator supports the following AWS Regions: US East (us-east-1, N. Virginia); US East (us-east-2, Ohio); US West (us-west-2, Oregon); Europe (eu-west-1, Ireland); Europe (eu-central-1, Frankfurt); Asia Pacific (ap-southeast-1, Singapore); Asia Pacific (ap-southeast-2, Sydney), Europe (eu-north-1, Stockholm), Asia Pacific (ap-south-1, Mumbai), Europe (eu-west-3, Paris), South America (sa-east-1, São Paulo), Asia Pacific (ap-northeast-2, Seoul), Europe (eu-west-2, London), Asia Pacific (ap-northeast-1, Tokyo), US West (us-west-1, N. California), Canada (ca-central-1, Central).

Here are some common uses for Amazon MSK Replicator.

- **Build multi-region streaming applications:** Build highly available and fault-tolerant streaming applications for increased resiliency without setting up custom solutions.
- **Lower latency data access:** Provide lower latency data access to consumers in different geographic regions.

- **Distribute data to your partners:** Copy data from one Apache Kafka cluster to many Apache Kafka clusters, so that different teams/partners have their own copies of data.
- **Aggregate data for analytics:** Copy data from multiple Apache Kafka clusters into one cluster for easily generating insights on aggregated real-time data.
- **Write locally, access your data globally:** Set up multi-active replication to automatically propagate writes performed in one AWS Region to other Regions for providing data at lower latency and cost.

How Amazon MSK Replicator works

To get started with MSK Replicator, you need create a new Replicator in your target cluster's AWS Region. MSK Replicator automatically copies all data from the cluster in the primary AWS Region called *source* to the cluster in the destination region called *target*. Source and target clusters can be in the same or different AWS Regions. You will need to create the target cluster if it does not already exist.

When you create a Replicator, MSK Replicator deploys all required resources in the target cluster's AWS Region to optimize for data replication latency. Replication latency varies based on many factors, including the network distance between the AWS Regions of your MSK clusters, the throughput capacity of your source and target clusters, and the number of partitions on your source and target clusters. MSK Replicator automatically scales the underlying resources so that you can replicate data on-demand without having to monitor or scale capacity.

Data replication

By default, MSK Replicator copies all data asynchronously from the latest offset in the source cluster topic partitions to the target cluster. If the "Detect and copy new topics" setting is turned on, MSK Replicator automatically detects and copies new topics or topic partitions to the target cluster. However, it may take up to 30 seconds for the Replicator to detect and create the new topics or topic partitions on the target cluster. Any messages produced to the source topic before the topic has been created on the target cluster will not be replicated. Alternatively, you can [configure your Replicator during creation](#) to start replication from the earliest offset in the source cluster topic partitions if you want to replicate existing messages on your topics to the target cluster.

MSK Replicator does not store your data. Data is consumed from your source cluster, buffered in-memory and written to the target cluster. The buffer is cleared automatically when the data

is either successfully written or fails after retries. All the communication and data between MSK Replicator and your clusters are always encrypted in-transit. All MSK Replicator API calls like `DescribeClusterV2`, `CreateTopic`, `DescribeTopicDynamicConfiguration` are captured in AWS CloudTrail. Your MSK broker logs will also reflect the same.

MSK Replicator creates topics in the target cluster with a Replicator Factor of 3. If you need to, you can modify the replication factor directly on the target cluster.

Metadata replication

MSK Replicator also supports copying the metadata from the source cluster to the target cluster. The metadata includes topic configuration, Access Control Lists (ACLs), and consumer groups offsets. Like data replication, metadata replication also happens asynchronously. For better performance, MSK Replicator prioritizes data replication over metadata replication.

The following table is a list of Access Control Lists (ACLs) that MSK Replicator copies.

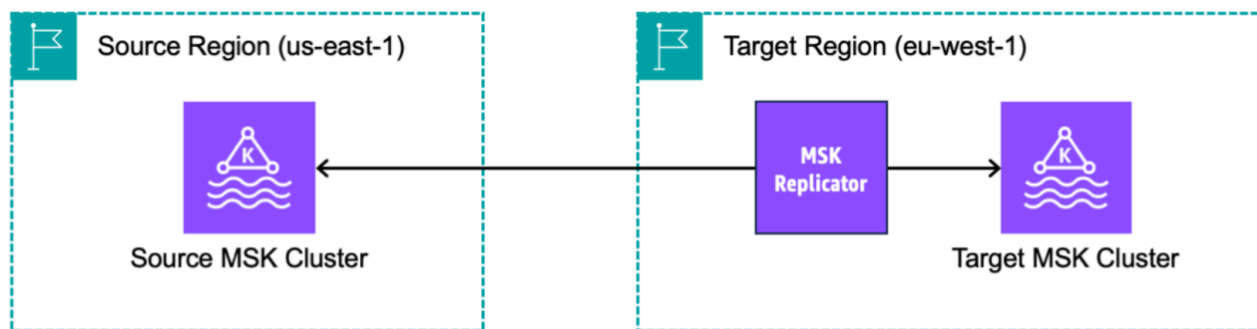
Operation	Research	APIs allowed
Alter	Topic	CreatePartitions
AlterConfigs	Topic	AlterConfigs
Create	Topic	CreateTopics, Metadata
Delete	Topic	DeleteRecords, DeleteTopics
Describe	Topic	ListOffsets, Metadata, OffsetFetch, OffsetForLeaderEpoch
DescribeConfigs	Topic	DescribeConfigs
Read	Topic	Fetch, OffsetCommit, TxnOffsetCommit
Write (deny only)	Topic	Produce, AddPartitionsToTxn

MSK Replicator copies LITERAL pattern type ACLs only for resource type Topic. PREFIXED pattern type ACLs and other resource type ACLs are not copied. MSK Replicator also does not delete ACLs

on the target cluster. If you delete an ACL on the source cluster, you should also delete on the target cluster at the same time. For more details on Kafka ACLs resource, pattern and operations, see https://kafka.apache.org/documentation/#security_authz_cli.

MSK Replicator replicates only Kafka ACLs, which IAM access control does not use. If your clients are using IAM access control to read/write to your MSK clusters, you need to configure the relevant IAM policies on your target cluster as well for seamless failover. This is also true for both Prefixed as well as Identical topic name replication configurations.

As part of consumer groups offsets syncing, MSK Replicator optimizes for your consumers on the source cluster which are reading from a position closer to the tip of the stream (end of the topic partition). If your consumer groups are lagging on the source cluster, you may see higher lag for those consumer groups on the target as compared to the source. This means after failover to the target cluster, your consumers will reprocess more duplicate messages. To reduce this lag, your consumers on the source cluster would need to catch up and start consuming from the tip of the stream (end of the topic partition). As your consumers catch up, MSK Replicator will automatically reduce the lag.



Topic name configuration

MSK Replicator has two topic name configuration modes: *Prefixed* (default) or *Identical* topic name replication.

Prefixed topic name replication

By default, MSK Replicator creates new topics in the target cluster with an auto-generated prefix added to the source cluster topic name, such as `<sourceKafkaClusterAlias>.topic`. This is to

distinguish the replicated topics from others in the target cluster and to avoid circular replication of data between the clusters.

For example, MSK Replicator replicates data in a topic named "topic" from the source cluster to a new topic in the target cluster called `<sourceKafkaClusterAlias>.topic`. You can find the prefix that will be added to the topic names in the target cluster under the **sourceKafkaClusterAlias** field using `DescribeReplicator` API or the **Replicator** details page on the MSK console. The prefix in the target cluster is `<sourceKafkaClusterAlias>`.

To make sure your consumers can reliably restart processing from the standby cluster, you need to configure your consumers to read data from the topics using a wildcard operator `.*`. For example, your consumers would need to consume using `.*topic1` in both AWS Regions. This example would also include a topic such as `footopic1`, so adjust the wildcard operator according to your needs.

You should use the MSK Replicator which adds a prefix when you want to keep replicator data in a separate topic in the target cluster, such as for active-active cluster setups.

Identical topic name replication

As an alternative to the default setting, Amazon MSK Replicator allows you to create a Replicator with topic replication set to Identical topic name replication (**Keep the same topics name** in console). You can create a new Replicator in the AWS Region which has your target MSK cluster. Identically-named replicated topics let you avoid reconfiguring clients to read from replicated topics.

Identical topic name replication (**Keep the same topics name** in console) has the following advantages:

- Allows you to retain identical topic names during the replication process, while also automatically avoiding the risk of infinite replication loops.
- Makes setting up and operating multi-cluster streaming architectures simpler, since you can avoid reconfiguring clients to read from the replicated topics.
- For active-passive cluster architectures, Identical topic name replication functionality also streamlines the failover process, allowing applications to seamlessly failover to a standby cluster without requiring any topic name changes or client reconfigurations.
- Can be used to more easily consolidate data from multiple MSK clusters into a single cluster for data aggregation or centralized analytics. This requires you to create separate Replicators for each source cluster and the same target cluster.

- Can streamline data migration from one MSK cluster to another by replicating data to identically named topics in the target cluster.

Amazon MSK Replicator uses Kafka headers to automatically avoid data being replicated back to the topic it originated from, eliminating the risk of infinite cycles during replication. A header is a key-value pair that can be included with the key, value, and timestamp in each Kafka message. MSK Replicator embeds identifiers for source cluster and topic into the header of each record being replicated. MSK Replicator uses the header information to avoid infinite replication loops. You should verify that your clients are able to read replicated data as expected.

Tutorial: Set up source and target clusters for Amazon MSK Replicator

This tutorial shows you how to set up a source cluster and a target cluster in the same AWS Region or in different AWS Regions. You then use those clusters to create an Amazon MSK Replicator.

Prepare the Amazon MSK source cluster

If you already have an MSK source cluster created for the MSK Replicator, make sure that it meets the requirements described in this section. Otherwise, follow these steps to create an MSK provisioned or serverless source cluster.

The process for creating a cross-region and same-region MSK Replicator source cluster are similar. Differences are called out in the following procedures.

1. Create an MSK provisioned or serverless cluster with [IAM access control turned on](#) in the source region. Your source cluster must have a minimum of three brokers.
2. For a cross-region MSK Replicator, if the source is a provisioned cluster, configure it with multi-VPC private connectivity turned on for IAM access control schemes. Note that the unauthenticated auth type is not supported when multi-VPC is turned on. You do not need to turn on multi-VPC private connectivity for other authentication schemes (mTLS or SASL/SCRAM). You can simultaneously use mTLS or SASL/SCRAM auth schemes for your other clients connecting to your MSK cluster. You can configure multi-VPC private connectivity in the console cluster details **Network settings** or with the UpdateConnectivity API. See [Cluster owner turns on multi-VPC](#). If your source cluster is an MSK Serverless cluster, you do not need to turn on multi-VPC private connectivity.

For a same-region MSK Replicator, the MSK source cluster does not require multi-VPC private connectivity and the cluster can still be accessed by other clients using the unauthenticated auth type.

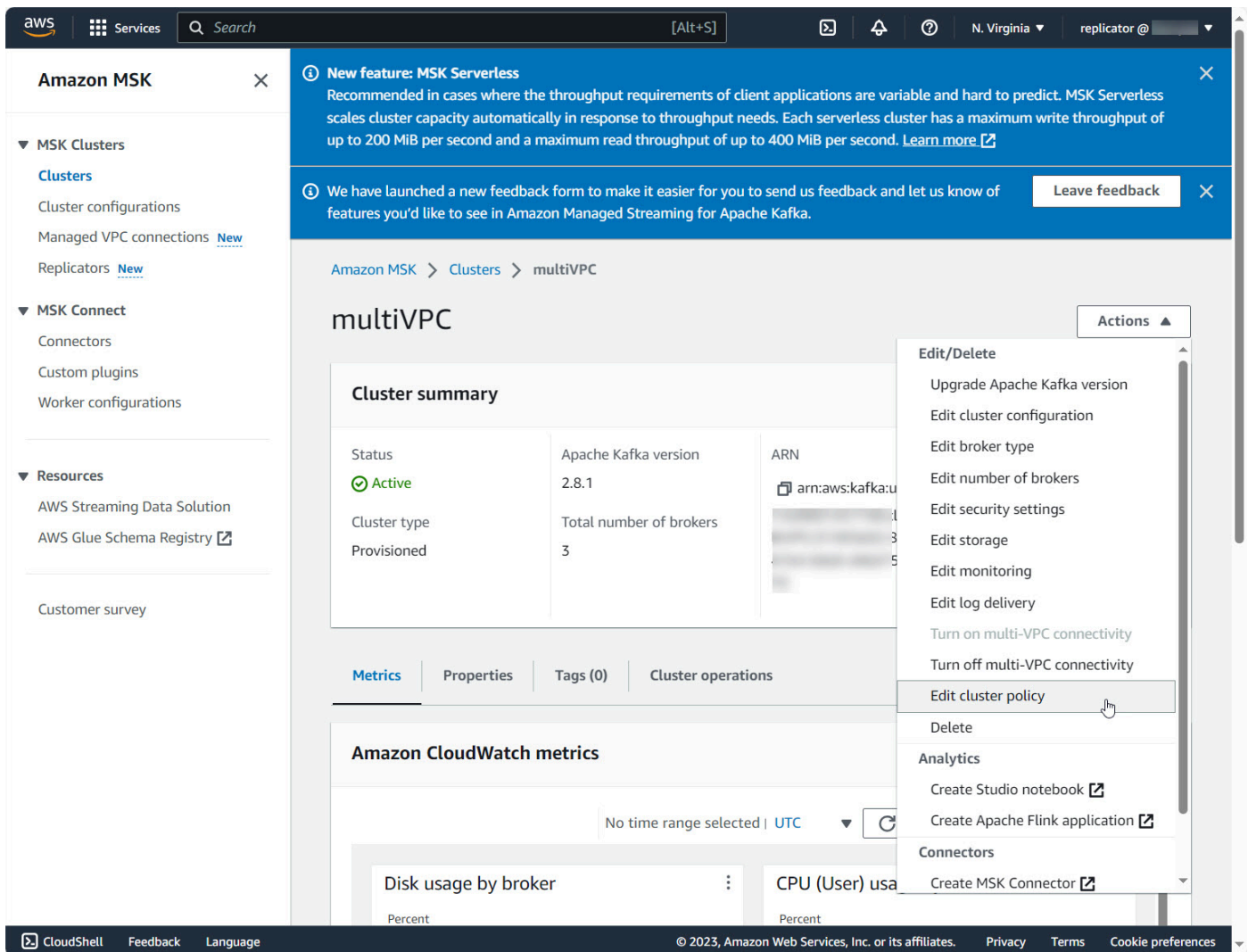
3. For cross-region MSK Replicators, you must attach a resource-based permissions policy to the source cluster. This allows MSK to connect to this cluster for replicating data. You can do this using the CLI or AWS Console procedures below. See also, [Amazon MSK resource-based policies](#). You do not need to perform this step for same-region MSK Replicators.

Console: create resource policy

Update the source cluster policy with the following JSON. Replace the placeholder with the ARN of your source cluster.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "kafka.amazonaws.com"
        ]
      },
      "Action": [
        "kafka:CreateVpcConnection",
        "kafka:GetBootstrapBrokers",
        "kafka:DescribeClusterV2"
      ],
      "Resource": "<sourceClusterARN>"
    }
  ]
}
```

Use the **Edit cluster policy** option under the **Actions** menu on the cluster details page.



CLI: create resource policy

Note: If you use the AWS console to create a source cluster and choose the option to create a new IAM role, AWS attaches the required trust policy to the role. If you want MSK to use an existing IAM role or if you create a role on your own, attach the following trust policies to that role so that MSK Replicator can assume it. For information about how to modify the trust relationship of a role, see [Modifying a Role](#).

1. Get the current version of the MSK cluster policy using this command. Replace placeholders with the actual cluster ARN.

```
aws kafka get-cluster-policy --cluster-arn <Cluster ARN>
{
  "CurrentVersion": "K1PA6795UKM GR7",
  "Policy": "..."
```

```
}
```

2. Create a resource-based policy to allow MSK Replicator to access your source cluster. Use the following syntax as a template, replacing the placeholder with the actual source cluster ARN.

```
aws kafka put-cluster-policy --cluster-arn "<sourceClusterARN>" --policy '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "kafka.amazonaws.com"
        ]
      },
      "Action": [
        "kafka:CreateVpcConnection",
        "kafka:GetBootstrapBrokers",
        "kafka:DescribeClusterV2"
      ],
      "Resource": "<sourceClusterARN>"
    }
  ]
}
```

Prepare the Amazon MSK target cluster

Create an MSK target cluster (provisioned or serverless) with IAM access control turned on. The target cluster doesn't require multi-VPC private connectivity turned on. The target cluster can be in the same AWS Region or a different Region as the source cluster. Both the source and target clusters must be in the same AWS account. Your target cluster must have a minimum of three brokers.

Tutorial: Create an Amazon MSK Replicator

After you set up the source and target clusters, you can use those clusters to create an Amazon MSK Replicator. Before you create the Amazon MSK Replicator, make sure you have [IAM permissions required to create an MSK Replicator](#).

Topics

- [Considerations for creating an Amazon MSK Replicator](#)
 - [IAM permissions required to create an MSK Replicator](#)
 - [Supported cluster types and versions for MSK Replicator](#)
 - [Supported MSK Serverless cluster configuration](#)
 - [Cluster configuration changes](#)
- [Create replicator using the AWS console in the target cluster Region](#)
 - [Choose your source cluster](#)
 - [Choose your target cluster](#)
 - [Configure replicator settings and permissions](#)

Considerations for creating an Amazon MSK Replicator

The following sections give an overview of the prerequisites, supported configurations, and best practices for using the MSK Replicator feature. It covers the necessary permissions, cluster compatibility, and Serverless-specific requirements, as well as guidance on managing the Replicator after creation.

IAM permissions required to create an MSK Replicator

Here is an example of the IAM policy required to create an MSK Replicator. The action `kafka:TagResource` is only needed if tags are provided when creating the MSK Replicator. Replicator IAM policies should be attached to the IAM role that corresponds to your client. For information about creating authorization policies, see [Create authorization policies](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MSKReplicatorIAMPassRole",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::123456789012:role/MSKReplicationRole",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "kafka.amazonaws.com"
        }
      }
    }
  ],
}
```

```

{
  "Sid": "MSKReplicatorServiceLinkedRole",
  "Effect": "Allow",
  "Action": "iam:CreateServiceLinkedRole",
  "Resource": "arn:aws:iam::123456789012:role/aws-service-role/
kafka.amazonaws.com/AWSServiceRoleForKafka\"",
},
{
  "Sid": "MSKReplicatorEC2Actions",
  "Effect": "Allow",
  "Action": [
    "ec2:DescribeSubnets",
    "ec2:DescribeSecurityGroups",
    "ec2:DescribeVpcs",
    "ec2:CreateNetworkInterface"
  ],
  "Resource": [
    "arn:aws:ec2:us-east-1:123456789012:subnet/subnet-0abcd1234ef56789",
    "arn:aws:ec2:us-east-1:123456789012:security-group/sg-0123abcd4567ef89",
    "arn:aws:ec2:us-east-1:123456789012:network-interface/eni-0a1b2c3d4e5f67890",
    "arn:aws:ec2:us-east-1:123456789012:vpc/vpc-0a1b2c3d4e5f67890"
  ]
},
{
  "Sid": "MSKReplicatorActions",
  "Effect": "Allow",
  "Action": [
    "kafka:CreateReplicator",
    "kafka:TagResource"
  ],
  "Resource": [
    "arn:aws:kafka:us-
east-1:123456789012:cluster/myCluster/abcd1234-56ef-78gh-90ij-klmnopqrstuv",
    "arn:aws:kafka:us-
east-1:123456789012:replicator/myReplicator/wxyz9876-54vu-32ts-10rq-ponmlkjihgfe"
  ]
}
]
}

```

The following is an example IAM policy to describe replicator. Either the `kafka:DescribeReplicator` action or `kafka:ListTagsForResource` action is needed, not both.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor1",
      "Effect": "Allow",
      "Action": [
        "kafka:DescribeReplicator",
        "kafka:ListTagsForResource"
      ],
      "Resource": "*"
    }
  ]
}
```

Supported cluster types and versions for MSK Replicator

These are requirements for supported instance types, Kafka versions, and network configurations.

- MSK Replicator supports both MSK provisioned clusters and MSK Serverless clusters in any combination as source and target clusters. Other types of Kafka clusters are not supported at this time by MSK Replicator.
- MSK Serverless clusters require IAM access control, don't support Apache Kafka ACL replication and with limited support on-topic configuration replication. See [What is MSK Serverless?](#).
- MSK Replicator is supported only on clusters running Apache Kafka 2.7.0 or higher, regardless of whether your source and target clusters are in the same or in different AWS Regions.
- MSK Replicator supports clusters using instance types of m5.large or larger. t3.small clusters aren't supported.
- If you are using MSK Replicator with an MSK Provisioned cluster, you need a minimum of three brokers in both source and target clusters. You can replicate data across clusters in two Availability Zones, but you would need a minimum of four brokers in those clusters.
- Both your source and target MSK clusters must be in the same AWS account. Replication across clusters in different accounts is not supported.
- If the source and target MSK clusters are in different AWS Regions (cross-region), MSK Replicator requires the source cluster to have multi-VPC private connectivity turned on for its IAM Access Control method.

Multi-VPC isn't required for other authentication methods on the source cluster for MSK replication across AWS Regions.

Multi-VPC is also not required if you're replicating data between clusters in the same AWS Region. See [the section called “Multi-VPC private connectivity in a single Region”](#).

- Identical topic name replication (**Keep the same topics name** in console) requires an MSK cluster running Kafka version 2.8.1 or higher.
- For Identical topic name replication (**Keep the same topics name** in console) configurations, to avoid the risk of cyclic replication, do not make changes to the headers that MSK Replicator creates (`__mskmr`).

Supported MSK Serverless cluster configuration

- MSK Serverless supports replication of these topic configurations for MSK Serverless target clusters during topic creation: `cleanup.policy`, `compression.type`, `max.message.bytes`, `retention.bytes`, `retention.ms`.
- MSK Serverless supports only these topic configurations during topic configuration sync: `compression.type`, `max.message.bytes`, `retention.bytes`, `retention.ms`.
- Replicator uses 83 compacted partitions on target MSK Serverless clusters. Make sure that target MSK Serverless clusters have a sufficient number of compacted partitions. See [MSK Serverless quota](#).

Cluster configuration changes

- It's recommended that you do not turn tiered storage on or off after the MSK Replicator has been created. If your target cluster is not tiered, then MSK won't copy the tiered storage configurations, regardless of whether your source cluster is tiered or not. If you turn on tiered storage on the target cluster after Replicator is created, the Replicator needs to be recreated. If you want to copy data from a non-tiered to a tiered cluster, you should not copy topic configurations. See [Enabling and disabling tiered storage on an existing topic](#).
- Don't change cluster configuration settings after MSK Replicator creation. Cluster configuration settings are validated during MSK Replicator creation. To avoid problems with the MSK Replicator, don't change the following settings after the MSK Replicator is created.
 - Change MSK cluster to t3 instance type.
 - Change service execution role permissions.

- Disable MSK multi-VPC private connectivity.
- Change the attached cluster resource-based policy.
- Change cluster security group rules.

Create replicator using the AWS console in the target cluster Region

The following section explains the step-wise console workflow for creating a replicator.

Replicator details

1. In the AWS Region where your target MSK cluster is located, open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. Choose **Replicators** to display the list of replicators in the account.
3. Choose **Create replicator**.
4. In the **Replicator details** pane, give the new replicator a unique name.

Choose your source cluster

The source cluster contains the data you want to copy to a target MSK cluster.

1. In the **Source cluster** pane, choose the AWS Region where the source cluster is located.

You can look up a cluster's Region by going to **MSK Clusters** and looking at the **Cluster** details ARN. The Region name is embedded in the ARN string. In the following example ARN, ap-southeast-2 is the cluster region.

```
arn:aws:kafka:ap-southeast-2:123456789012:cluster/cluster-11/
eec93c7f-4e8b-4baf-89fb-95de01ee639c-s1
```

2. Enter the ARN of your source cluster or browse to choose your source cluster.
3. Choose subnet(s) for your source cluster.

The console displays the subnets available in the source cluster's Region for you to select. You must select a minimum of two subnets. For a same-region MSK Replicator, the subnets that you select set to access the source cluster and the subnets to access the target cluster must be in the same Availability Zone.

4. Choose security group(s) for the MSK Replicator to access your source cluster.

- For cross-region replication (CRR), you do not need to provide security group(s) for your source cluster.
- For same region replication (SRR), go to the Amazon EC2 console at <https://console.aws.amazon.com/ec2/> and ensure that the security groups you will provide for the Replicator have outbound rules to allow traffic to your source cluster's security groups. Also, ensure that your source cluster's security groups have inbound rules that allow traffic from the Replicator security groups provided for the source.

To add inbound rules to your source cluster's security group:

1. In the AWS console, go to your source cluster's details by selecting the the **Cluster name**.
2. Select the **Properties tab**, then scroll down to the **Network settings** pane to select the name of the **Security group** applied.
3. Go to the inbound rules and select **Edit inbound rules**.
4. Select **Add rule**.
5. In the **Type** column for the new rule, select **Custom TCP**.
6. In the **Port range** column, type 9098. MSK Replicator uses IAM access control to connect to your cluster which uses port 9098.
7. In the **Source** column, type the name of the security group that you will provide during Replicator creation for the source cluster (this may be the same as the MSK source cluster's security group), and then select **Save rules**.

To add outbound rules to Replicator's security group provided for the source:

1. In the AWS console for Amazon EC2, go to the security group that you will provide during Replicator creation for the source.
2. Go to the outbound rules and select **Edit outbound rules**.
3. Select **Add rule**.
4. In the **Type** column for the new rule, select **Custom TCP**.
5. In the **Port range** column, type 9098. MSK Replicator uses IAM access control to connect to your cluster which uses port 9098.

6. In the **Source** column, type the name of the MSK source cluster's security group, and then select **Save rules**.

Note

Alternately, if you do not want to restrict traffic using your security groups, you can add inbound and outbound rules allowing All Traffic.

1. Select **Add rule**.
2. In the **Type** column, select **All Traffic**.
3. In the **Source** column, type `0.0.0.0/0`, and then select **Save rules**.

Choose your target cluster

The target cluster is the MSK provisioned or serverless cluster to which the source data is copied.

Note

MSK Replicator creates new topics in the target cluster with an auto-generated prefix added to the topic name. For instance, MSK Replicator replicates data in "topic" from the source cluster to a new topic in the target cluster called `<sourceKafkaClusterAlias>.topic`. This is to distinguish topics that contain data replicated from source cluster from other topics in the target cluster and to avoid data being circularly replicated between the clusters. You can find the prefix that will be added to the topic names in the target cluster under the **sourceKafkaClusterAlias** field using `DescribeReplicator` API or the **Replicator details** page on the MSK Console. The prefix in the target cluster is `<sourceKafkaClusterAlias>`.

1. In the **Target cluster** pane, choose the AWS Region where the target cluster is located.
2. Enter the ARN of your target cluster or browse to choose your target cluster.
3. Choose subnet(s) for your target cluster.

The console displays subnets available in the target cluster's Region for you to select. Select a minimum of two subnets.

4. Choose security group(s) for the MSK Replicator to access your target cluster.

The security groups available in the target cluster's Region are displayed for you to select. The chosen security group is associated with each connection. For more information about using security groups, see the [Control traffic to your AWS resources using security groups](#) in the *Amazon VPC User Guide*.

- For both cross region replication (CRR) and same region replication (SRR), go to the Amazon EC2 console at <https://console.aws.amazon.com/ec2/> and ensure that the security groups you will provide to the Replicator have outbound rules to allow traffic to your target cluster's security groups. Also ensure that your target cluster's security groups have inbound rules that accept traffic from the Replicator security groups provided for the target.

To add inbound rules to your target cluster's security group:

1. In the AWS console, go to your target cluster's details by selecting the the **Cluster name**.
2. Select the **Properties** tab, then scroll down to the Network settings pane to select the name of the **Security group** applied.
3. Go to the inbound rules and select **Edit inbound rules**.
4. Select **Add rule**.
5. In the **Type** column for the new rule, select **Custom TCP**.
6. In the **Port range** column, type 9098. MSK Replicator uses IAM access control to connect to your cluster which uses port 9098.
7. In the **Source** column, type the name of the security group that you will provide during Replicator creation for the target cluster (this may be the same as the MSK target cluster's security group), and then select **Save rules**.

To add outbound rules to Replicator's security group provided for the target:

1. In the AWS console, go to the security group that you will provide during Replicator creation for the target.
2. Select the **Properties** tab, then scroll down to the Network settings pane to select the name of the **Security group** applied.
3. Go to the outbound rules and select **Edit outbound rules**.
4. Select **Add rule**.
5. In the **Type** column for the new rule, select **Custom TCP**.

6. In the **Port range** column, type 9098. MSK Replicator uses IAM access control to connect to your cluster which uses port 9098.
7. In the **Source** column, type the name of the MSK target cluster's security group, and then select **Save rules**.

Note

Alternately, if you do not want to restrict traffic using your security groups, you can add inbound and outbound rules allowing All Traffic.

1. Select **Add rule**.
2. In the **Type** column, select **All Traffic**.
3. In the Source column, type 0.0.0.0/0, and then select **Save rules**.

Configure replicator settings and permissions

1. In the **Replicator settings** pane, specify the topics you want to replicate using regular expressions in the allow and deny lists. By default, all topics are replicated.

Note

MSK Replicator only replicates up to 750 topics in sorted order. If you need to replicate more topics, we recommend that you create a separate Replicator. Go to the AWS console Support Center and [create a support case](#) if you need support for more than 750 topics per Replicator. You can monitor the number of topics being replicated using the "TopicCount" metric. See [Amazon MSK Standard broker quota](#).

2. By default, MSK Replicator starts replication from the *latest* (most recent) offset in the selected topics. Alternatively, you can start replication from the *earliest* (oldest) offset in the selected topics if you want to replicate existing data on your topics. Once the Replicator is created, you can't change this setting. This setting corresponds to the [startingPosition](#) field in the [CreateReplicator](#) request and [DescribeReplicator](#) response APIs.
3. Choose a topic name configuration:
 - **PREFIXED topic name replication (Add prefix to topics name in console):** The default setting. MSK Replicator replicates "topic1" from the source cluster to a new topic in the target cluster with the name <sourceKafkaClusterAlias>.topic1.

- **Identical topic name replication (Keep the same topics name in console):** Topics from the source cluster are replicated with identical topic names in the target cluster.

This setting corresponds to the `TopicNameConfiguration` field in the `CreateReplicator` request and `DescribeReplicator` response APIs. See [How Amazon MSK Replicator works](#).

 **Note**

By default, MSK Replicator creates new topics in the target cluster with an auto-generated prefix added to the topic name. This is to distinguish topics that contain data replicated from source cluster from other topics in the target cluster and to avoid data being circularly replicated between the clusters. Alternatively, you can create a MSK Replicator with Identical topic name replication (**Keep the same topics name in console**) so that topic names are preserved during replication. This configuration reduces the need for you to reconfigure client applications during setup and makes it simpler to operate multi-cluster streaming architectures.

4. By default, MSK Replicator copies all metadata including topic configurations, Access Control Lists (ACLs) and consumer group offsets for seamless failover. If you are not creating the Replicator for failover, you can optionally choose to turn off one or more of these settings available in the **Additional settings** section.

 **Note**

MSK Replicator does not replicate write ACLs since your producers should not be writing directly to the replicated topic in the target cluster. Your producers should write to the local topic in the target cluster after failover. See [Perform a planned failover to the secondary AWS Region](#) for details.

5. In the **Consumer group replication** pane, specify the consumer groups you want to replicate using regular expressions in the allow and deny lists. By default, all consumer groups are replicated.
6. In the **Compression** pane, you can optionally choose to compress the data written to the target cluster. If you're going to use compression, we recommend that you use the same compression method as the data in your source cluster.
7. In the **Access permissions** pane do either of the following:

- a. Select **Create or update IAM role with required policies**. MSK console will automatically attach the necessary permissions and trust policy to the service execution role required to read and write to your source and target MSK clusters.

Access permissions

Replicator uses IAM access control to connect to source and target MSK clusters. Your source and target clusters should be turned on for IAM access control with permissions for the IAM role. See [permissions required to successfully create a replicator](#).

You can't change the access permissions after you create the replicator.

Access to cluster resources

☒ Create or update IAM role **MSKReplicatorServiceRole-** with required policies

☐ Choose from IAM roles that Amazon MSK can assume

- b. Provide your own IAM role by selecting **Choose from IAM roles that Amazon MSK can assume**. We recommend that you attach the `AWSMSKReplicatorExecutionRole` managed IAM policy to your service execution role, instead of writing your own IAM policy.
- Create the IAM role that the Replicator will use to read and write to your source and target MSK clusters with the below JSON as part of the trust policy and the `AWSMSKReplicatorExecutionRole` attached to the role. In the trust policy, replace the placeholder `<yourAccountID>` with your actual account ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "kafka.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "<yourAccountID>"
        }
      }
    }
  ]
}
```

8. In the **Replicator tags** pane, you can optionally assign tags to the MSK Replicator resource. For more information, see [Tag an Amazon MSK cluster](#). For a cross-region MSK Replicator, tags are synced to the remote Region automatically when the Replicator is created. If you change tags after the Replicator is created, the change is not automatically synced to the remote Region, so you'll need to sync local replicator and remote replicator references manually.
9. Select **Create**.

If you want to restrict `kafka-cluster:WriteData` permission, refer to the *Create authorization policies* section of [How IAM access control for Amazon MSK works](#). You'll need to add `kafka-cluster:WriteDataIdempotently` permission to both the source and target cluster.

It takes approximately 30 minutes for the MSK Replicator to be successfully created and transitioned to `RUNNING` status.

If you create a new MSK Replicator to replace one that you deleted, the new Replicator starts replication from the latest offset.

If your MSK Replicator has transitioned to a `FAILED` status, refer to the troubleshooting section [Troubleshooting MSK Replicator](#).

Edit MSK Replicator settings

You can't change the source cluster, target cluster, Replicator starting position, or topic name replication configuration once the MSK Replicator has been created. You need to create a new replicator to use identical topic name replication configuration. However, you can edit other Replicator settings, such as topics and consumer groups to replicate.

1. Sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. In the left navigation pane, choose **Replicators** to display the list of Replicators in the account and select the MSK Replicator you want to edit.
3. Choose the **Properties** tab.
4. In the **Replicator settings** section, choose **Edit replicator**.
5. You can edit the MSK Replicator settings by changing any of these settings.
 - Specify the topics you want to replicate using regular expressions in the allow and deny lists. By default, MSK Replicator copies all metadata including topic configurations, Access Control

Lists (ACLs) and consumer group offsets for seamless failover. If you are not creating the Replicator for failover, you can optionally choose to turn off one or more of these settings available in the **Additional settings** section.

 **Note**

MSK Replicator does not replicate write ACLs since your producers should not be writing directly to the replicated topic in the target cluster. Your producers should write to the local topic in the target cluster after failover. See [Perform a planned failover to the secondary AWS Region](#) for details.

- For **Consumer group replication**, you can specify the consumer groups you want to replicate using regular expressions in the allow and deny lists. By default, all consumer groups are replicated. If allow and deny lists are empty, consumer group replication is turned off.
- Under **Target compression type**, you can choose whether to compress the data written to the target cluster. If you're going to use compression, we recommend that you use the same compression method as the data in your source cluster.

6. Save your changes.

It takes approximately 30 minutes for the MSK Replicator to be successfully created and transitioned to running state. If your MSK Replicator has transitioned to a FAILED status, refer to the troubleshooting section [???](#).

Delete an MSK Replicator

You may need to delete a MSK Replicator if it fails to create (FAILED status). The source and target clusters assigned to an MSK Replicator can't be changed once the MSK Replicator is created. You can delete an existing MSK Replicator and create a new one. If you create a new MSK Replicator to replace the deleted one, the new Replicator starts replication from the latest offset.

1. In the AWS Region where your source cluster is located, sign in to the AWS Management Console, and open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.
2. In the navigation pane, select **Replicators**.
3. From the list of MSK Replicators, select the one you want to delete and choose **Delete**.

Monitor replication

You can use <https://console.aws.amazon.com/cloudwatch/> in the target cluster Region to view metrics for ReplicationLatency, MessageLag, and ReplicatorThroughput at a topic and aggregate level for each Amazon MSK Replicator. Metrics are visible under **ReplicatorName** in the “AWS/Kafka” namespace. You can also see ReplicatorFailure, AuthError and ThrottleTime metrics to check for issues.

The MSK console displays a subset of CloudWatch metrics for each MSK Replicator. From the console **Replicator** list, select the name of a Replicator and select the **Monitoring** tab.

MSK Replicator metrics

The following metrics describes performance or connection metrics for the MSK Replicator.

AuthError metrics do not cover topic-level auth errors. To monitor your MSK Replicator’s topic-level auth errors, monitor Replicator’s ReplicationLatency metrics and the source cluster’s topic-level metrics, MessagesInPerSec. If a topic’s ReplicationLatency dropped to 0 but the topic still has data being produced to it, it indicates that the Replicator has an Auth issue with the topic. Check that the Replicator’s service execution IAM role has sufficient permission to access the topic.

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
Performance	ReplicationLatency	Time it takes records to replicate from the source to target cluster; duration between record produce time at source and replicate d to target.	ReplicatorName	Milliseconds	Partition	Maximum	
			ReplicatorName, Topic	Milliseconds	Partition	Maximum	

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
		If ReplicationLatency increases, check if clusters have enough partitions to support replication. High replication latency can occur when the partition count is too low for high throughput.					

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
Performance	MessageLag	Monitors the sync between the MSK Replicator and the source cluster. MessageLag indicates the lag between the messages produced to the source cluster and messages consumed by the replicator. It is not the lag between the source and target cluster. Even if the source cluster is unavailable/interrupted, the replicator will finish writing the message it has consumed to the target cluster. After an outage,	ReplicatorName	Count	Partition	Sum	
			ReplicatorName, Topic	Count	Partition	Sum	

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
		MessageLag shows an increase indicating the number of messages the replicator is behind the source cluster and this can be monitored until the number of messages is 0, showing that the replicator has caught up with the source cluster.					

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
Performance	ReplicatorBytesInPerSec	Average number of bytes processed by the replicator per second. Data processed by MSK Replicator consists of all the data that MSK Replicator receives which includes the data replicated to target cluster and the data filtered by MSK Replicator (only if your Replicator is configured with Identical topic name configuration) to prevent the data being copied back to the same topic it originated from. If your Replicator is configured with	ReplicatorName	BytesPer second	ReplicatorName	Sum	

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
		"Prefixed" topic name configuration, both ReplicatorBytesInPerSec and ReplicatorThroughput metrics will have the same value as no data will be filtered by MSK Replicator.					

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
Performance	ReplicatorThroughput	Average number of bytes replicated per second. If ReplicatorThroughput drops for a topic, check KafkaClusterPingSuccessCount and AuthError metrics to ensure the Replicator can communicate with clusters, then check cluster metrics to ensure the cluster is not down.	ReplicatorName	BytesPerSecond	Partition	Sum	
			ReplicatorName, Topic	BytesPerSecond	Partition	Sum	

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
Debug	AuthError	The number of connections with failed authentication per second. If this metric is above 0, you can check if the service execution role policy for the replicator is valid and make sure there aren't deny permissions set for the cluster permissions. Based on clusterAlias dimension, you can identify if the source or target cluster is experiencing auth errors.	ReplicatorName, ClusterAlias	Count	Worker	Sum	

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
Debug	ThrottleTime	The average time in ms a request was throttled by brokers on the cluster. Set throttling to avoid having the MSK Replicator overwhelm the cluster. If this metric is 0, replicationLatency is not high, and replicationThroughput is as expected, then throttling is working as expected. If this metric is above 0, you can adjust throttling accordingly.	ReplicatorName, ClusterAlias	Milliseconds	Worker	Maximum	
Debug	ReplicatorFailure	Number of failures that the replicator is experiencing.	ReplicatorName	Count		Sum	

Metric type	Metric	Description	Dimensions	Unit	Raw Metric Granularity	Raw Metric Aggregation Stat	
Debug	KafkaClusterPingSuccessCount	Indicates the health of the replicator connection to the kafka cluster. If this value is 1, the connection is healthy. If the value is 0 or no datapoint, the connection is unhealthy . If the value is 0, you can check network or IAM permission settings for the Kafka cluster. Based on ClusterAlias dimension, you can identify whether this metric is for source or target cluster.	ReplicatorName, ClusterAlias	Count		Sum	

Use replication to increase the resiliency of a Kafka streaming application across Regions

You can use MSK Replicator to set up active-active or active-passive cluster topologies to increase resiliency of your Apache Kafka application across AWS Regions. In an active-active setup, both MSK clusters are actively serving reads and writes. In an active-passive setup, only one MSK cluster at a time is actively serving streaming data, while the other cluster is on standby.

Considerations for building multi-Region Apache Kafka applications

Your consumers must be able to reprocess duplicate messages without downstream impact. MSK Replicator replicates data at-least-once which may result in duplicates in the standby cluster. When you switch over to the secondary AWS Region, your consumers may process the same data more than once. MSK Replicator prioritizes copying data over consumer offsets for better performance. After a failover, the consumer may start reading from earlier offsets resulting in duplicate processing.

Producers and consumers must also tolerate losing minimal data. Since MSK Replicator replicates data asynchronously, when the primary AWS Region starts experiencing failures, there is no guarantee that all data is replicated to the secondary Region. You can use the replication latency to determine maximum data that was not copied into the secondary Region.

Using active-active versus active-passive cluster topology

An active-active cluster topology offers near zero recovery time and the capability for your streaming application to operate simultaneously in multiple AWS Regions. When a cluster in one Region is impaired, applications connected to the cluster in the other Region continue processing data.

Active-passive setups are suited to applications that can run in only one AWS Region at a time, or when you need more control over the data processing order. Active-passive setups require more recovery time than active-active setups, as you must start your entire active-passive setup, including your producers and consumers, in the secondary Region to resume streaming data after a failover.

Create an active-passive Kafka cluster setup with recommended topic naming configurations

For an active-passive setup, we recommend you to operate a similar setup of producers, MSK clusters, and consumers (with the same consumer group name) in two different AWS Regions. It is important that the two MSK clusters have identical read and write capacity to ensure reliable data replication. You need to create a MSK Replicator to continuously copy data from the primary to the standby cluster. You also need to configure your producers to write data into topics on a cluster in the same AWS Region.

For an active-passive setup, create a new Replicator with Identical topic name replication (**Keep the same topics name** in console) to start replicating data from your MSK cluster in the primary region to your cluster in the secondary region. We recommend that you operate a duplicate set of producers and consumers in the two AWS Regions, each connecting to the cluster in their own region using its bootstrap string. This simplifies the failover process since it won't require changes to the bootstrap string. To ensure that consumers read from near where they left off, consumers in the source and target clusters should have the same consumer group ID.

If you use Identical topic name replication (**Keep the same topics name** in console) for your MSK Replicator, it will replicate your topics with the same name as the corresponding source topics.

We recommend that you configure cluster level settings and permissions for your clients on the target cluster. You do not need to configure topic level settings and literal read ACLs as MSK Replicator automatically copies them if you have selected the option to copy access control lists. See [Metadata replication](#).

Failover to the secondary AWS Region

We recommend that you monitor replication latency in the secondary AWS Region using Amazon CloudWatch. During a service event in the primary AWS Region, replication latency may suddenly increase. If the latency keeps increasing, use the AWS Service Health Dashboard to check for service events in the primary AWS Region. If there's an event, you can failover to the secondary AWS Region.

Perform a planned failover to the secondary AWS Region

You can conduct a planned failover to test the resiliency of your application against an unexpected event in your primary AWS region which has your source MSK cluster. A planned failover should not result in data loss.

If you're using Identical topic name replication configuration, follow these steps:

1. Shutdown all producers and consumers connecting to your source cluster.
2. Create a new MSK Replicator to replicate data from your MSK cluster in the secondary Region to your MSK cluster in the primary Region with Identical topic name replication (**Keep the same topics name** in console). This is required to copy the data that you will be writing to the secondary region back to the primary Region so that you can failback to the primary Region after the unexpected event has ended.
3. Start producers and consumers connected to the target cluster in the secondary AWS Region.

If you're using Prefixed topic name configuration, follow these steps to failover:

1. Shutdown all producers and consumers connecting to your source cluster.
2. Create a new MSK Replicator to replicate data from your MSK cluster in the secondary Region to your MSK cluster in the primary Region. This is required to copy the data that you will be writing to the secondary region back to the primary Region so that you can failback to the primary Region after the unexpected event has ended.
3. Start producers on target cluster in the secondary AWS Region.
4. Depending on your application's message ordering requirements, follow the steps in one of the following tabs.

No message ordering

If your application does not require message ordering, start consumers in the secondary AWS Region that read from both the local (for example, `topic`) and replicated topics (for example, `<sourceKafkaClusterAlias>.topic`) using a wildcard operator (for example, `.*topic`).

Message ordering

If your application requires message ordering, start consumers only for the replicated topics on target cluster (for example, `<sourceKafkaClusterAlias>.topic`) but not the local topics (for example, `topic`).

5. Wait for all the consumers of replicated topics on the target MSK cluster to finish processing all data, so that consumer lag is 0 and the number of records processed is also 0. Then, stop consumers for the replicated topics on target cluster. At this point, all records that were replicated from the source MSK cluster to target MSK cluster have been consumed.

6. Start consumers for the local topics (for example, `topic`) on the target MSK cluster.

Perform an unplanned failover to the secondary AWS Region

You can conduct an unplanned failover when there is a service event in the primary AWS Region which has your source MSK cluster and you want to temporarily redirect your traffic to the secondary Region which has your target MSK cluster. An unplanned failover could result in some data loss as MSK Replicator replicates data asynchronously. You can track the message lag using the metrics in [???](#).

If you're using Identical topic name replication configuration (**Keep the same topics name** in console), follow these steps:

1. Attempt to shut down all producers and consumers connecting to the source MSK cluster in the primary Region. This operation might not succeed due to impairments in that region.
2. Start producers and consumers connecting to the target MSK cluster in the secondary AWS Region to complete the failover. As MSK Replicator also replicates metadata including read ACLs and consumer group offsets, your producers and consumers will seamlessly resume processing from near where they left off before failover.

If you're using PREFIX topic name configuration, follow these steps to failover:

1. Attempt to shut down all producers and consumers connecting to the source MSK cluster in the primary Region. This operation might not succeed due to impairments in that region.
2. Start producers and consumers connecting to the target MSK cluster in the secondary AWS Region to complete the failover. As MSK Replicator also replicates metadata including read ACLs and consumer group offsets, your producers and consumers will seamlessly resume processing from near where they left off before failover.
3. Depending on your application's message ordering requirements, follow the steps in one of the following tabs.

No message ordering

If your application does not require message ordering, start consumers in the target AWS Region that read from both the local (for example, `topic`) and replicated topics (for example, `<sourceKafkaClusterAlias>.topic`) using a wildcard operator (for example, `.*topic`).

Message ordering

1. Start consumers only for the replicated topics on target cluster (for example, `<sourceKafkaClusterAlias>.topic`) but not the local topics (for example, `topic`).
2. Wait for all the consumers of replicated topics on the target MSK cluster to finish processing all data, so that offset lag is 0 and the number of records processed is also 0. Then, stop consumers for the replicated topics on target cluster. At this point, all records that were replicated from the source MSK cluster to target MSK cluster have been consumed.
3. Start consumers for the local topics (for example, `topic`) on the target MSK cluster.
4. Once the service event has ended in the primary Region, create a new MSK Replicator to replicate data from your MSK cluster in the secondary Region to your MSK cluster in the primary Region with Replicator starting position set to *earliest*. This is required to copy the data that you will be writing to the secondary Region back to the primary Region so that you can failback to the primary Region after the service event has ended. If you don't set the Replicator starting position to *earliest*, any data you produced to the cluster in the secondary region during the service event in the primary region will not be copied back to the cluster in the primary region.

Perform failback to the primary AWS Region

You can failback to the primary AWS region after the service event in that region has ended.

If you're using Identical topic name replication configuration, follow these steps:

1. Create a new MSK Replicator with your secondary cluster as source and primary cluster as target, starting position set to *earliest* and Identical topic name replication (**Keep the same topics name** in console).

This will start the process of copying all data written to the secondary cluster after failover back to the primary region.

2. Monitor the MessageLag metric on the new replicator in Amazon CloudWatch until it reaches 0, which indicates all data has been replicated from secondary to primary.
3. After all data has been replicated, stop all producers connecting to the secondary cluster and start producers connecting to the primary cluster.

4. Wait for `MaxOffsetLag` metric for your consumers connecting to secondary cluster to become 0 to ensure they have processed all the data. See [Monitor consumer lags](#).
5. Once all data has been processed, stop consumers in the secondary region and start consumers connecting to the primary cluster to complete the failback.
6. Delete the Replicator you created in the first step that is replicating data from your secondary cluster to primary.
7. Verify that your existing Replicator copying data from primary to secondary cluster has status as "RUNNING" and `ReplicatorThroughput` metric in Amazon CloudWatch 0.

Note that when you create a new Replicator with starting position as *Earliest* for failback, it starts reading all data in your secondary clusters' topics. Depending on your data retention settings, your topics may have data that came from your source cluster. While MSK Replicator automatically filters those messages, you will still incur data processing and transfer charges for all the data in your secondary cluster. You can track the total data processed by replicator using `ReplicatorBytesInPerSec`. See [MSK Replicator metrics](#).

If you're using Prefixed topic name configuration, follow these steps:

You should initiate failback steps only after replication from the cluster in the secondary Region to the cluster in the primary Region has caught up and the `MessageLag` metric in Amazon CloudWatch is close to 0. A planned failback should not result in any data loss.

1. Shut down all producers and consumers connecting to the MSK cluster in the secondary Region.
2. For active-passive topology, delete the Replicator that is replicating data from cluster in the secondary Region to primary Region. You do not need to delete the Replicator for active-active topology.
3. Start producers connecting to the MSK cluster in the primary Region.
4. Depending on your application's message ordering requirements, follow the steps in one of the following tabs.

No message ordering

If your application does not require message ordering, start consumers in the primary AWS Region that read from both the local (for example, `topic`) and replicated topics (for example, `<sourceKafkaClusterAlias>.topic`) using a wildcard operator (for example, `.*topic`). The consumers on local topics (e.g.: `topic`) will resume from the last offset they

consumed before the failover. If there was any unprocessed data from before the failover, it will get processed now. In the case of a planned failover, there should be no such record.

Message ordering

1. Start consumers only for the replicated topics on primary Region (for example, `<sourceKafkaClusterAlias>.topic`) but not the local topics (for example, `topic`).
2. Wait for all the consumers of replicated topics on the cluster in the primary Region to finish processing all data, so that offset lag is 0 and the number of records processed is also 0. Then, stop consumers for the replicated topics on cluster in the primary Region. At this point, all records that were produced in the secondary Region after failover have been consumed in the primary Region.
3. Start consumers for the local topics (for example, `topic`) on the cluster in the primary Region.
5. Verify that the existing Replicator from cluster in primary to cluster in secondary Region is in `RUNNING` state and working as expected using the `ReplicatorThroughput` and latency metrics.

Create an active-active setup using MSK Replicator

If you want to create an active-active setup where both MSK clusters are actively serving reads and writes, we recommend that you use an MSK Replicator with Prefixed topic name replication (**Add prefix to topics name** in console). However, this will require you to reconfigure your consumers to read the replicated topics.

Follow these steps to set up active-active topology between source MSK cluster A and target MSK cluster B.

1. Create a MSK Replicator with MSK cluster A as source and MSK cluster B as target.
2. After the above MSK Replicator has been successfully created, create a Replicator with cluster B as source and cluster A as target.
3. Create two sets of producers, each writing data at the same time into the local topic (for example, `"topic"`) in the cluster in the same region as the producer.
4. Create two sets of consumers, each reading data using a wildcard subscription (such as `".*topic"`) from the MSK cluster in the same AWS Region as the consumer. This way your consumers will automatically read data produced locally in the Region from the local topic (for example, `topic`), as well as data replicated from other Region in topic with the prefix

`<sourceKafkaClusterAlias>.topic`). These two sets of consumers should have different consumer group IDs so that consumer group offsets are not overwritten when MSK Replicator copies them to the other cluster.

If you want to avoid reconfiguring your clients, instead of the Prefixed topic name replication (**Add prefix to topics name** in console), you can create the MSK Replicators using Identical topic name replication (**Keep the same topics name** in console) to create an active-active setup. However, you will pay additional data processing and data transfer charges for each Replicator. This is because each Replicator will need to process twice the usual amount of data, once for replication and again to prevent infinite loops. You can track the total amount of data processed by each replicator using the `ReplicatorBytesInPerSec` metric. See [Monitor replication](#). This metric includes the data replicated to target cluster as well as the data filtered by MSK Replicator to prevent the data being copied back to the same topic it originated from.

Note

If you're using Identical topic name replication (**Keep the same topics name** in console) to set up active-active topology, wait at least 30 seconds after deleting a topic before re-creating a topic with the same name. This waiting period helps to prevent duplicated messages being replicated back to the source cluster. Your consumers must be able to reprocess duplicate messages without downstream impact. See [Considerations for building multi-Region Apache Kafka applications](#).

Migrate from one Amazon MSK cluster to another using MSK Replicator

You can use Identical topic name replication for cluster migration, but your consumers must be able to handle duplicate messages without downstream impact. This is because MSK Replicator provides at-least-once replication, which can lead to duplicate messages in rare scenarios. If your consumers meet this requirement, follow these steps.

1. Create a Replicator that replicates data from your old cluster to the new cluster with Replicator's starting position set to *Earliest* and using Identical topic name replication (**Keep the same topics name** in console).

2. Configure cluster-level settings and permissions on the new cluster. You do not need to configure topic-level settings and “literal” read ACLs, as MSK Replicator automatically copies them.
3. Monitor the MessageLag metric in Amazon CloudWatch until it reaches 0 which indicates all data has been replicated.
4. After all data has been replicated, stop producers from writing data to the old cluster.
5. Reconfigure those producers to connect to the new cluster and start them.
6. Monitor MaxOffsetLag metric for your consumers reading data from the old cluster until it becomes 0, which indicates all existing data has been processed.
7. Stop consumers that are connecting to the old cluster.
8. Reconfigure consumers to connect to the new cluster and start them.

Migrate from self-managed MirrorMaker2 to MSK Replicator

To migrate from MirrorMaker (MM2) to MSK Replicator, follow these steps:

1. Stop the producer that is writing to your source Amazon MSK cluster.
2. Allow MM2 to replicate all the messages on your source clusters' topics. You can monitor the consumer lag for MM2 consumer on your source MSK cluster to determine when all data has been replicated.
3. Create a new Replicator with starting position set to *Latest* and topic name configuration set to IDENTICAL (**Same topic names replication** in console).
4. Once your Replicator is in the RUNNING state, you can start the producers writing to the source cluster again.

Troubleshoot MSK Replicator

The following information can help you troubleshoot problems that you might have with MSK Replicator. See [Troubleshoot your Amazon MSK cluster](#) for problem solving information about other Amazon MSK features. You can also post your issue to [AWS re:Post](#).

MSK Replicator state goes from CREATING to FAILED

Here are some common causes for MSK Replicator creation failure.

1. Verify that the security groups you provided for the Replicator creation in the Target cluster section have outbound rules to allow traffic to your target cluster's security groups. Also verify that your target cluster's security groups have inbound rules that accept traffic from the security groups you provide for the Replicator creation in the Target cluster section. See [Choose your target cluster](#).
2. If you are creating Replicator for cross-region replication, verify that your source cluster has multi-VPC connectivity turned on for IAM Access Control authentication method. See [Amazon MSK multi-VPC private connectivity in a single Region](#). Also verify that the cluster policy is setup on the source cluster so that the MSK Replicator can connect to the source cluster. See [Prepare the Amazon MSK source cluster](#).
3. Verify that the IAM role that you provided during MSK Replicator creation has the permissions required to read and write to your source and target clusters. Also, verify that the IAM role has permissions to write to topics. See [Configure replicator settings and permissions](#)
4. Verify that your network ACLs are not blocking the connection between the MSK Replicator and your source and target clusters.
5. It's possible that source or target clusters are not fully available when the MSK Replicator tried to connect to them. This might be due to excessive load, disk usage or CPU usage, which causes the Replicator to be unable to connect to the brokers. Fix the issue with the brokers and retry Replicator creation.

After you have performed the validations above, create the MSK Replicator again.

MSK Replicator appears stuck in the CREATING state

Sometimes MSK Replicator creation can take up to 30 minutes. Wait for 30 minutes and check the state of the Replicator again.

MSK Replicator is not replicating data or replicating only partial data

Follow these steps to troubleshoot data replication problems.

1. Verify that your Replicator is not running into any authentication errors using the AuthError metric provided by MSK Replicator in Amazon CloudWatch. If this metric is above 0, check if the policy of the IAM role you provided for the replicator is valid and there aren't deny permissions set for the cluster permissions. Based on clusterAlias dimension, you can identify if the source or target cluster is experiencing authentication errors.

2. Verify that your source and target clusters are not experiencing any issues. It is possible that the Replicator is not able to connect to your source or target cluster. This might happen due to too many connections, disk at full capacity or high CPU usage.
3. Verify that your source and target clusters are reachable from MSK Replicator using the `KafkaClusterPingSuccessCount` metric in Amazon CloudWatch. Based on `clusterAlias` dimension, you can identify if the source or target cluster is experiencing auth errors. If this metric is 0 or has no datapoint, the connection is unhealthy. You should check network and IAM role permissions that MSK Replicator is using to connect to your clusters.
4. Verify that your Replicator is not running into failures due to missing topic-level permissions using the `ReplicatorFailure` metric in Amazon CloudWatch. If this metric is above 0, check the IAM role you provided for topic-level permissions.
5. Verify that the regular expression you provided in the allow list while creating the Replicator matches the names of the topics you want to replicate. Also, verify that the topics are not being excluded from replication due to a regular expression in the deny list.
6. Note that it may take up to 30 seconds for the Replicator to detect and create the new topics or topic partitions on the target cluster. Any messages produced to the source topic before the topic has been created on the target cluster will not be replicated if replicator starting position is latest (default). Alternatively, you can start replication from the earliest offset in the source cluster topic partitions if you want to replicate existing messages on your topics on the target cluster. See [Configure replicator settings and permissions](#).

Message offsets in the target cluster are different than the source cluster

As part of replicating data, MSK Replicator consumes messages from the source cluster and produces them to the target cluster. This can lead to messages having different offsets on your source and target clusters. However, if you have turned on consumer groups offsets syncing during Replicator creation, MSK Replicator will automatically translate the offsets while copying the metadata so that after failing over to the target cluster, your consumers can resume processing from near where they left off in the source cluster.

MSK Replicator is not syncing consumer groups offsets or consumer group does not exist on target cluster

Follow these steps to troubleshoot metadata replication problems.

1. Verify that your data replication is working as expected. If not, see [MSK Replicator is not replicating data or replicating only partial data](#).
2. Verify that the regular expression you provided in the allow list while creating the Replicator matches the names of the consumer groups you want to replicate. Also, verify that the consumer groups are not being excluded from replication due to a regular expression in the deny list.
3. Verify that MSK Replicator has created the topic on the target cluster. It may take up to 30 seconds for the Replicator to detect and create the new topics or topic partitions on the target cluster. Any messages produced to the source topic before the topic has been created on the target cluster will not be replicated if the replicator starting position is *latest* (default). If your consumer group on the source cluster has only consumed the messages that have not been replicated by MSK Replicator, the consumer group will not be replicated to the target cluster. After the topic is successfully created on the target cluster, MSK Replicator will start replicating newly written messages on the source cluster to the target. Once your consumer group starts reading these messages from the source, MSK Replicator will automatically replicate the consumer group to the target cluster. Alternatively, you can start replication from the earliest offset in the source cluster topic partitions if you want to replicate existing messages on your topics on the target cluster. See [Configure replicator settings and permissions](#).

Note

MSK Replicator optimizes consumer groups offset syncing for your consumers on the source cluster which are reading from a position closer to the end of the topic partition. If your consumer groups are lagging on the source cluster, you may see higher lag for those consumer groups on the target as compared to the source. This means after failover to the target cluster, your consumers will reprocess more duplicate messages. To reduce this lag, your consumers on the source cluster would need to catch up and start consuming from the tip of the stream (end of the topic partition). As your consumers catch up, MSK Replicator will automatically reduce the lag.

Replication latency is high or keeps increasing

Here are some common causes for high replication latency.

1. Verify that you have the right number of partitions on your source and target MSK clusters. Having too few or too many partitions can impact performance. For guidance on choosing the number of partitions, see [Best practices for using MSK Replicator](#). The following table shows the recommended minimum number of partitions for getting the throughput you want with MSK Replicator.

Throughput and recommended minimum number of partitions

Throughput (MB/s)	Minimum number of partitions required
50	167
100	334
250	833
500	1666
1000	3333

2. Verify that you have enough read and write capacity in your source and target MSK clusters to support the replication traffic. MSK Replicator acts as a consumer for your source cluster (egress) and as a producer for your target cluster (ingress). Therefore, you should provision cluster capacity to support the replication traffic in addition to other traffic on your clusters. See [???](#) for guidance on sizing your MSK clusters.
3. Replication latency might vary for MSK clusters in different source and destination AWS Region pairs, depending on how geographically far apart the clusters are from each other. For example, Replication latency is typically lower when replicating between clusters in the Europe (Ireland) and Europe (London) Regions compared to replication between clusters in the Europe (Ireland) and Asia Pacific (Sydney) Regions.
4. Verify that your Replicator is not getting throttled due to overly aggressive quotas set on your source or target clusters. You can use the ThrottleTime metric provided by MSK Replicator in Amazon CloudWatch to see the average time in milliseconds a request was throttled by brokers on your source/target cluster. If this metric is above 0, you should adjust Kafka quotas to reduce throttling so that Replicator can catch-up. See [Managing MSK Replicator throughput using Kafka quotas](#) for information on managing Kafka quotas for the Replicator.
5. ReplicationLatency and MessageLag might increase when an AWS Region becomes degraded. Use the [AWS Service Health Dashboard](#) to check for an MSK service event in the Region where

your primary MSK cluster is located. If there's a service event, you can temporarily redirect your application reads and writes to the other Region.

Troubleshooting MSK Replicator failures using ReplicatorFailure metric

The ReplicatorFailure metric helps you monitor and detect replication issues in MSK Replicator. A non-zero value of this metric typically indicates replication failure issue, which might result from the following factors:

- message size limitations
- timestamp range violations
- record batch size problems

If the ReplicatorFailure metric reports a non-zero value, follow these steps to troubleshoot the issue.

Note

For more information about this metric, see [MSK Replicator metrics](#).

1. Configure a client that is able to connect to the target MSK cluster and has Apache Kafka CLI tools setup. For information about setting up the client and Kafka CLI tool, see [Connect to an Amazon MSK Provisioned cluster](#).
2. Open the Amazon MSK console at <https://console.aws.amazon.com/msk/home?region=us-east-1#/home/>.

Then, do the following:

- a. Obtain the ARNs of MSK Replicator and target MSK cluster.
 - b. [Obtain the broker endpoints](#) of the target MSK cluster. You'll use these endpoints in the following steps.
3. Run the following commands to export the MSK Replicator ARN and broker endpoints you obtained in the previous step.

Make sure that you replace the placeholder values for `<ReplicatorARN>`, `<BootstrapServerString>`, and `<ConsumerConfigFile>` used in the following examples with their actual values.

```
export TARGET_CLUSTER_SERVER_STRING=<BootstrapServerString>
```

```
export REPLICATOR_ARN=<ReplicatorARN>
```

```
export CONSUMER_CONFIG_FILE=<ConsumerConfigFile>
```

4. In your `<path-to-your-kafka-installation>/bin` directory, do the following:
 - a. Save the following script and name it **query-replicator-failure-message.sh**.

```
#!/bin/bash

# Script: Query MSK Replicator Failure Message
# Description: This script queries exceptions from AWS MSK Replicator status
#             topics
# It takes a replicator ARN and bootstrap server as input and searches for
# replicator exceptions
# in the replicator's status topic, formatting and displaying them in a
# readable manner
#
# Required Arguments:
#   --replicator-arn: The ARN of the AWS MSK Replicator
#   --bootstrap-server: The Kafka bootstrap server to connect to
#   --consumer.config: Consumer config properties file
# Usage Example:
#   ./query-replicator-failure-message.sh ./query-replicator-failure-message.sh
#   --replicator-arn <replicator-arn> --bootstrap-server <bootstrap-server> --
#   consumer.config <consumer.config>

print_usage() {
    echo "USAGE: $0 ./query-replicator-failure-message.sh --replicator-arn
    <replicator-arn> --bootstrap-server <bootstrap-server> --consumer.config
    <consumer.config>"
    echo "--replicator-arn <String: MSK Replicator ARN>          REQUIRED: The ARN of
    AWS MSK Replicator."
```

```

    echo "--bootstrap-server <String: server to connect to> REQUIRED: The Kafka
server to connect to."
    echo "--consumer.config <String: config file> REQUIRED: Consumer
config properties file."
    exit 1
}

# Initialize variables
replicator_arn=""
bootstrap_server=""
consumer_config=""

# Parse arguments
while [[ $# -gt 0 ]]; do
    case "$1" in
        --replicator-arn)
            if [ -z "$2" ]; then
                echo "Error: --replicator-arn requires an argument."
                print_usage
            fi
            replicator_arn="$2"; shift 2 ;;
        --bootstrap-server)
            if [ -z "$2" ]; then
                echo "Error: --bootstrap-server requires an argument."
                print_usage
            fi
            bootstrap_server="$2"; shift 2 ;;
        --consumer.config)
            if [ -z "$2" ]; then
                echo "Error: --consumer.config requires an argument."
                print_usage
            fi
            consumer_config="$2"; shift 2 ;;
        *) echo "Unknown option: $1"; print_usage ;;
    esac
done

# Check for required arguments
if [ -z "$replicator_arn" ] || [ -z "$bootstrap_server" ] || [ -z
"$consumer_config" ]; then
    echo "Error: --replicator-arn, --bootstrap-server, and --consumer.config are
required."
    print_usage
fi

```

```
# Extract replicator name and suffix from ARN
replicator_arn_suffix=$(echo "$replicator_arn" | awk -F'/' '{print $NF}')
replicator_name=$(echo "$replicator_arn" | awk -F'/' '{print $(NF-1)}')
echo "Replicator name: $replicator_name"

# List topics and find the status topic
topics=$(./kafka-topics.sh --command-config client.properties --list --
bootstrap-server "$bootstrap_server")
status_topic_name="__amazon_msk_replicator_status_${replicator_name}_
${replicator_arn_suffix}"

# Check if the status topic exists
if echo "$topics" | grep -Fq "$status_topic_name"; then
    echo "Found replicator status topic: '$status_topic_name'"
    ./kafka-console-consumer.sh --bootstrap-server "$bootstrap_server" --
consumer.config "$consumer_config" --topic "$status_topic_name" --from-
beginning | stdbuf -oL grep "Exception" | stdbuf -oL sed -n 's/.*Exception:\(.*\
\) Topic: \([^,]*\), Partition: \([^\\]*\).*/ReplicatorException:\1 Topic: \2,
Partition: \3/p'
else
    echo "No topic matching the pattern '$status_topic_name' found."
fi
```

- b. Run this script to query the MSK Replicator failure messages.

```
<path-to-your-kafka-installation>/bin/query-replicator-failure-message.sh --
replicator-arn $REPLICATOR_ARN --bootstrap-server $TARGET_CLUSTER_SERVER_STRING
--consumer.config $CONSUMER_CONFIG_FILE
```

This script outputs all the errors with their exception messages and affected topic-partitions. You can use this exception information to mitigate the failures as described in [Common MSK Replicator failures and their solutions](#). Because the topic contains all the historical failure messages, start investigation using the last message. The following is an example of a failure message.

```
ReplicatorException: The request included a message larger than the max message
size the server will accept. Topic: test, Partition: 1
```

Common MSK Replicator failures and their solutions

The following list describes some of the MSK Replicator failures that you might experience and how to mitigate them.

Message size larger than `max.request.size`

Cause

This failure occurs when the MSK Replicator fails to replicate data because the individual message size exceeds 10 MB. By default, MSK Replicator replicates messages up to 10 MB in size.

The following is an example of this failure message type.

```
ReplicatorException: The message is 20635370 bytes when serialized which is larger than 10485760, which is the value of the max.request.size configuration. Topic: test, Partition: 1
```

Solution

Reduce the individual message sizes in your topic. If you're unable to do so, follow these instructions for [requesting a limit increase](#).

Message size larger than the max message size the server will accept

Cause

This failure occurs when the message size exceeds the target cluster's maximum message size.

The following is an example of this failure message type.

```
ReplicatorException: The request included a message larger than the max message size the server will accept. Topic: test, Partition: 1
```

Solution

Increase the `max.message.bytes` configuration on the target cluster or corresponding target cluster topic. Set the target cluster's `max.message.bytes` configuration to match your largest uncompressed message size. For information about doing this, see [max.message.bytes](#).

Timestamp is out of range

Cause

This failure occurs because the individual message timestamp falls outside of the target cluster's allowed range.

The following is an example of this failure message type.

```
ReplicatorException: Timestamp 1730137653724 of message with offset 0 is out of range. The timestamp should be within [1730137892239, 1731347492239] Topic: test, Partition: 1
```

Solution

Update the target cluster's `message.timestamp.before.max.ms` configuration to allow for messages with older timestamps. For information about doing this, see [message.timestamp.before.max.ms](#).

Record batch too large

Cause

This failure occurs because the record batch size exceeds the segment size set for the topic on the target cluster. MSK Replicator supports a maximum batch size of 1 MB.

The following is an example of this failure message type.

```
ReplicatorException: The request included message batch larger than the configured segment size on the server. Topic: test, Partition: 1
```

Solution

The target cluster's `segment.bytes` configuration must be at least as large as the batch size (1 MB) for Replicator to proceed without errors. Update the target cluster's `segment.bytes` to be at least 1048576 (1 MB). For information about doing this, see [segment.bytes](#).

Note

If the `ReplicatorFailure` metric continues to emit non-zero values after applying these solutions, repeat the troubleshooting process until the metric emits a value of zero.

Best practices for using MSK Replicator

This section covers common best practices and implementation strategies for using Amazon MSK Replicator.

Topics

- [Managing MSK Replicator throughput using Kafka quotas](#)
- [Setting cluster retention period](#)

Managing MSK Replicator throughput using Kafka quotas

Since MSK Replicator acts as a consumer for your source cluster, replication can cause other consumers to be throttled on your source cluster. The amount of throttling depends on the read capacity you have on your source cluster and the throughput of data you're replicating. We recommend that you provision identical capacity for your source and target clusters, and account for the replication throughput when calculating how much capacity you need.

You can also set Kafka quotas for the Replicator on your source and target clusters to control how much capacity the MSK Replicator can use. A network bandwidth quota is recommended. A network bandwidth quota defines a byte rate threshold, defined as bytes per second, for one or more clients sharing a quota. This quota is defined on a per-broker basis.

Follow these steps to apply a quota.

1. Retrieve the bootstrap server string for the source cluster. See [Get the bootstrap brokers for an Amazon MSK cluster](#).
2. Retrieve the service execution role (SER) used by the MSK Replicator. This is the SER you used for a `CreateReplicator` request. You can also pull the SER from the `DescribeReplicator` response from an existing Replicator.
3. Using Kafka CLI tools, run the following command against the source cluster.

```
./kafka-configs.sh --bootstrap-server <source-cluster-bootstrap-server> --alter --  
add-config 'consumer_byte_  
rate=<quota_in_bytes_per_second>' --entity-type users --entity-name  
arn:aws:sts::<customer-account-id>:assumed-role/<ser-role-name>/<customer-account-  
id> --command-config <client-properties-for-iam-auth></programlisting>
```

4. After executing the above command, verify that the `ReplicatorThroughput` metric does not cross the quota you have set.

Note that if you re-use a service execution role between multiple MSK Replicators they are all subject to this quota. If you want to maintain separate quotas per Replicator, use separate service execution roles.

For more information on using MSK IAM authentication with quotas, see [Multi-tenancy Apache Kafka clusters in Amazon MSK with IAM access control and Kafka Quotas – Part 1](#).

 **Warning**

Setting an extremely low `consumer_byte_rate` may cause your MSK Replicator to act in unexpected ways.

Setting cluster retention period

You can set the log retention period for MSK provisioned and serverless clusters. The recommended retention period is 7 days. See [Cluster configuration changes](#) or [Supported MSK Serverless cluster configuration](#).

MSK integrations

This section provides references to AWS features that integrate with Amazon MSK.

Topics

- [Amazon Athena connector for Amazon MSK](#)
- [Amazon Redshift streaming data ingestion for Amazon MSK](#)
- [Firehose integration for Amazon MSK](#)
- [Access Amazon EventBridge Pipes through the Amazon MSK console](#)
- [Using Kafka Streams with MSK Express brokers and MSK Serverless](#)
- [Real-time vector embedding blueprints](#)

Amazon Athena connector for Amazon MSK

The Amazon Athena connector for Amazon MSK enables Amazon Athena to run SQL queries on Apache Kafka topics. Use this connector to view Apache Kafka topics as tables and messages as rows in Athena.

For more information, see [Amazon Athena MSK Connector](#) in the *Amazon Athena User Guide*.

Amazon Redshift streaming data ingestion for Amazon MSK

Amazon Redshift supports streaming ingestion from Amazon MSK. The Amazon Redshift streaming ingestion feature provides low-latency, high-speed ingestion of streaming data from Amazon MSK into an Amazon Redshift materialized view. Because it doesn't need to stage data in Amazon S3, Amazon Redshift can ingest streaming data at a lower latency and at a reduced storage cost. You can configure Amazon Redshift streaming ingestion on an Amazon Redshift cluster using SQL statements to authenticate and connect to an Amazon MSK topic.

For more information, see [Streaming ingestion](#) in the *Amazon Redshift Database Developer Guide*.

Firehose integration for Amazon MSK

Amazon MSK integrates with Firehose to provide a serverless, no-code solution to deliver streams from Apache Kafka clusters to Amazon S3 data lakes. Firehose is a streaming extract,

transform, and load (ETL) service that reads data from your Amazon MSK Kafka topics, performs transformations such as conversion to Parquet, and aggregates and writes the data to Amazon S3. With few click from the console, you can setup a Firehose stream to read from a Kafka topic and deliver to an S3 location. There is no code to write, no connector applications, and no resources to provision. Firehose automatically scales based on the amount of data published to the Kafka topic, and you only pay for the bytes ingested from Kafka.

See the following for more information about this feature.

- [Writing to Kinesis Data Firehose Using Amazon MSK - Amazon Kinesis Data Firehose](#) in the *Amazon Data Firehose Developer Guide*
- Blog: [Amazon MSK Introduces Managed Data Delivery from Apache Kafka to Your Data Lake](#)
- Lab: [Delivery to Amazon S3 using Firehose](#)

Access Amazon EventBridge Pipes through the Amazon MSK console

Amazon EventBridge Pipes connects sources to targets. Pipes are intended for point-to-point integrations between supported sources and targets, with support for advanced transformations and enrichment. EventBridge Pipes provide a highly scalable way to connect your Amazon MSK cluster to AWS services such as Step Functions, Amazon SQS, and API Gateway, as well as third-party software as a service (SaaS) applications like Salesforce.

To set up a pipe, you choose the source, add optional filtering, define optional enrichment, and choose the target for the event data.

On the details page for an Amazon MSK cluster, you can view the pipes that use that cluster as their source. From there, you can also:

- Launch the EventBridge console to view pipe details.
- Launch the EventBridge console to create a new pipe with the cluster as its source.

For more information on configuring an Amazon MSK cluster as a pipe source, see [Amazon Managed Streaming for Apache Kafka cluster as a source](#) in the *Amazon EventBridge User Guide*. For more information about EventBridge Pipes in general, see [EventBridge Pipes](#).

To access EventBridge pipes for a given Amazon MSK cluster

1. Open the [Amazon MSK console](#) and choose **Clusters** .
2. Select a cluster.
3. On the cluster detail page, choose the **Integration** tab.

The **Integration** tab includes a list of any pipes currently configured to use the selected cluster as a source, including:

- pipe name
 - current status
 - pipe target
 - when the pipe was last modified
4. Manage the pipes for your Amazon MSK cluster as desired:

To access more details about a pipe

- Choose the pipe.

This launches the **Pipe details** page of the EventBridge console.

To create a new pipe

- Choose **Connect Amazon MSK cluster to pipe**.

This launches the **Create pipe** page of the EventBridge console, with the Amazon MSK cluster specified as the pipe source. For more information, see [Creating an EventBridge pipe](#) in the *Amazon EventBridge User Guide*.

- You can also create a pipe for a cluster from the **Clusters** page. Select the cluster, and, from the **Actions** menu, select **Create EventBridge Pipe**.

Using Kafka Streams with MSK Express brokers and MSK Serverless

Kafka Streams supports stateless and stateful transformations. Stateful transformations, such as count, aggregate, or join, use operators which store their state in internal Kafka topics. Furthermore, some stateless transformations such as groupBy or repartition store their results in internal Kafka topics. By default, Kafka Streams names these internal topics based on the corresponding operator. If these topics don't exist, Kafka Streams creates internal Kafka topics. For creating the internal topics, Kafka Streams hardcodes the segment.bytes configuration and sets it to 50 MB. [MSK Provisioned with Express brokers](#) and MSK Serverless protects some [topic configurations](#), including segment.size during topic creation. Therefore, a Kafka Streams application with stateful transformations fails to create the internal topics using MSK Express brokers or MSK Serverless.

To run such Kafka Streams applications on MSK Express brokers or MSK Serverless, you must create the internal topics yourself. To do this, first identify and name the Kafka Streams operators, which require topics. Then, create the corresponding internal Kafka topics.

Note

- It's a best practice to name the operators manually in Kafka Streams, especially the ones which depend on internal topics. For information about naming operators, see [Naming Operators in a Kafka Streams DSL Application](#) in the Kafka Streams documentation.
- The internal topic name for a stateful transformation depends on the `application.id` of the Kafka Streams application and the name of the stateful operator, `application.id-statefuloperator_name`.

Topics

- [Creating a Kafka Streams application using MSK Express brokers or MSK Serverless](#)

Creating a Kafka Streams application using MSK Express brokers or MSK Serverless

If your Kafka Streams application has its `application.id` set to `msk-streams-processing`, you can create a Kafka Streams application using MSK Express brokers or MSK Serverless. To do

this, use the `count()` operator, which requires an internal topic with the name. For example, `msk-streams-processing-count-store`.

To create a Kafka Streams application, do the following:

Topics

- [Identify and name the operators](#)
- [Create the internal topics](#)
- [\(Optional\) Check the topic name](#)
- [Examples of naming operators](#)

Identify and name the operators

1. Identify the stateful processors using the [Stateful transformations](#) in the Kafka Streams documentation.

Some examples of stateful processors include `count`, `aggregate`, or `join`.

2. Identify the processors that create topics for repartitioning.

The following example contains a `count()` operation, which needs a state.

```
var stream =  
    paragraphStream  
        .groupByKey()  
        .count()  
        .toStream();
```

3. To name the topic, add a name for each stateful processor. Based on the processor type, the naming is done by a different naming class. For example, `count()` operation is an aggregation operation. Therefore, it needs the `Materialized` class.

For information about the naming classes for the stateful operations, see [Conclusion](#) in the Kafka Streams documentation.

The following example sets the name of `count()` operator to `count-store` using the `Materialized` class.

```
var stream =  
    paragraphStream
```

```
.groupByKey()  
.count(Materialized.<String, Long, KeyValueStore<Bytes, byte[]>>as("count-  
store") // descriptive name for the store  
.withKeySerde(Serdes.String())  
.withValueSerde(Serdes.Long()))  
.toStream();
```

Create the internal topics

Kafka Streams prefixes `application.id` to names of internal topics, where `application.id` is user-defined. For example, `application.id-internal_topic_name`. The internal topics are normal Kafka topics, and you can create the topics using the information available in [Create an Apache Kafka topic](#) or `AdminClient` of the Kafka API.

Depending on your use case, you can use the Kafka Streams' default cleanup and retention policies, or customize their values. You define these in `cleanup.policy` and `retention.ms`.

The following example creates the topics with the `AdminClient` API and sets the `application.id` to **msk-streams-processing**.

```
try (AdminClient client = AdminClient.create(configs.kafkaProps())) {  
    Collection<NewTopic> topics = new HashSet<>();  
    topics.add(new NewTopic("msk-streams-processing-count-store", 3, (short) 3));  
    client.createTopics(topics);  
}
```

After the topics are created on the cluster, your Kafka Streams application can use the `msk-streams-processing-count-store` topic for the `count()` operation.

(Optional) Check the topic name

You can use the *topography describer* to describe the topology of your stream and view the names of the internal topics. The following example shows how to run the topology describer.

```
final StreamsBuilder builder = new StreamsBuilder();  
Topology topology = builder.build();  
System.out.println(topology.describe());
```

The following output shows the topology of the stream for the preceding example.

Topology Description:**Topologies:**

Sub-topology: 0

Source: KSTREAM-SOURCE-0000000000 (topics: [input_topic])

--> KSTREAM-AGGREGATE-0000000001

Processor: KSTREAM-AGGREGATE-0000000001 (stores: [count-store])

--> KTABLE-TOSTREAM-0000000002

<-- KSTREAM-SOURCE-0000000000

Processor: KTABLE-TOSTREAM-0000000002 (stores: [])

--> KSTREAM-SINK-0000000003

<-- KSTREAM-AGGREGATE-0000000001

Sink: KSTREAM-SINK-0000000003 (topic: output_topic)

<-- KTABLE-TOSTREAM-0000000002

For information about how to use the topology describer, see [Naming Operators in a Kafka Streams DSL Application](#) in the Kafka Streams documentation.

Examples of naming operators

This section provides some examples of naming operators.

Example of naming operator for `groupByKey()`

```
groupByKey() -> groupByKey(Grouped.as("kafka-stream-groupby"))
```

Example of naming operator for `normal count()`

```
normal count() -> .count(Materialized.<String, Long, KeyValueStore<Bytes,
    byte[]>>as("kafka-streams-window") // descriptive name for the store
    .withKeySerde(Serdes.String())
    .withValueSerde(Serdes.Long()))
```

Example of naming operator for `windowed count()`

```
windowed count() -> .count(Materialized.<String, Long, WindowStore<Bytes,
    byte[]>>as("kafka-streams-window") // descriptive name for the store
    .withKeySerde(Serdes.String())
    .withValueSerde(Serdes.Long()))
```

Example of naming operator for `windowed suppressed()`

```
windowed suppressed() ->
Suppressed<Windowed> suppressed = Suppressed
    .untilWindowCloses(Suppressed.BufferConfig.unbounded())
    .withName("kafka-suppressed");
    .suppress(suppressed)
```

Real-time vector embedding blueprints

Amazon MSK (Managed Streaming for Apache Kafka) supports Amazon Managed Service for Apache Flink blueprints to generate vector-embeddings using Amazon Bedrock, streamlining the process to build real-time AI applications powered by up-to-date, contextual data. The MSF blueprint simplifies the process of incorporating the latest data from your Amazon MSK streaming pipelines into your generative AI models, eliminating the need to write custom code to integrate real-time data streams, vector databases, and large language models.

You can configure the MSF blueprint to continuously generate vector embeddings using Bedrock's embedding models, then index those embeddings in OpenSearch Service for their Amazon MSK data streams. This allows you to combine the context from real-time data with Bedrock's powerful large language models to generate accurate, up-to-date AI responses without writing custom code. You can also choose to improve the efficiency of data retrieval using built-in support for data chunking techniques from LangChain, an open-source library, supporting high-quality inputs for model ingestion. The blueprint manages the data integration and processing between MSK, the chosen embedding model, and the OpenSearch vector store, allowing you to focus on building your AI applications, rather than managing the underlying integration.

Real-time vector embedding blueprints is available in the following AWS Regions:

- N. Virginia - us-east-1
- Ohio - us-east-2
- Oregon - us-west-2
- Mumbai - ap-south-1
- Seoul - ap-northeast-2
- Singapore - ap-southeast-1
- Sydney - ap-southeast-2
- Tokyo - ap-northeast-1

- Canada Central - ca-central-1
- Frankfurt - eu-central-1
- Ireland - eu-west-1
- London - eu-west-2
- Paris - eu-west-3
- Sao Paulo - sa-east-1

Topics

- [Logging and observability](#)
- [Notes before enabling real-time vector embedding blueprints](#)
- [Deploy streaming data vectorization blueprint](#)

Logging and observability

All logs and metrics for real-time vector embedding blueprints can be enabled using CloudWatch logs.

All metrics that are available for a regular MSF application and Amazon Bedrock can monitor your [application](#) and [Bedrock metrics](#).

There are two additional metrics for monitoring performance of generating embeddings. These metrics are part of the EmbeddingGeneration operation name in CloudWatch.

- **BedrockTitanEmbeddingTokenCount:** monitors the number of tokens present in a single request to Bedrock.
- **BedrockEmbeddingGenerationLatencyMs:** reports the time taken to send and receive a response from Bedrock for generating embeddings in milliseconds.

For OpenSearch Service, you can use the following metrics:

- **OpenSearch Serverless collection metrics:** see [Monitoring OpenSearch Serverless with Amazon CloudWatch](#) in the *Amazon OpenSearch Service Developer Guide*.
- **OpenSearch provisioned metrics:** see [Monitoring OpenSearch cluster metrics with Amazon CloudWatch](#) in the *Amazon OpenSearch Service Developer Guide*.

Notes before enabling real-time vector embedding blueprints

The Managed Service for Apache Flink application will only support unstructured text or JSON data in the input stream.

Two modes of input processing are supported:

- When input data is **unstructured text**, the entire text message is embedded. The vector DB contains the original text and the generated embedding.
- When the input data is in **JSON** format, the application gives you the ability to configure and specify one or more keys within the JSON object value to use for the embedding process. If there is more than one key, all keys are vectorized together and indexed in the vector DB. The vector DB will contain the original message and the generated embedding.

Embedding Generation: The application supports all text embedding models exclusively provided by Bedrock.

Persist in vector DB store: The application uses an existing OpenSearch cluster (provisioned or Serverless) in the customer's account as a destination for persisting embedded data. When using OpenSearch Serverless to create a vector index, always use the vector field name `embedded_data`.

Similar to MSF blueprints, you are expected to manage the infrastructure to run the code associated with the real-time vector embedding blueprint.

Similar to MSF Blueprints, once an MSF application is created, it must be exclusively started in the AWS account using the console or CLI. AWS will not start the MSF application for you. You have to call the `StartApplication` API (through CLI or console) to get the application running.


Cross-account movement of data: The application does not allow you to move data between input stream and vector destinations that live in different AWS accounts.

Deploy streaming data vectorization blueprint

This topic describes how to deploy a streaming data vectorization blueprint.

Deploy streaming data vectorization blueprint

1. Ensure following resources are setup correctly:
 - Provisioned or Serverless MSK cluster with one or more topics containing data.

2. Bedrock Setup: [Access to desired Bedrock Model](#). Currently supported Bedrock models are:
 - Amazon Titan Embeddings G1 - Text
 - Amazon Titan Text Embeddings V2
 - Amazon Titan Multimodal Embeddings G1
 - Cohere Embed English
 - Cohere Embed Multilingual
 3. AWS OpenSearch collection:
 - You may use a provisioned or Serverless OpenSearch Service collection.
 - The OpenSearch Service collection must have at least one index.
 - If you plan to use an **OpenSearch Serverless collection**, make sure to create a vector search collection. For details on how to setup a vector index, see [Prerequisites for your own vector store for a knowledge base](#). To learn more about vectorization, see [Amazon OpenSearch Service's vector database capabilities explained](#).
-  **Note**

When creating a vector index, you must use the vector field name `embedded_data`.
- If you plan to use an **OpenSearch Provisioned collection**, you need to add the MSF application role (that contains the Opensearch access policy) that was created by the blueprint, as a master user to your OpenSearch collection. Also, confirm that the access policy in OpenSearch is set to "Allow" actions. This is needed to [enable fine grain access control](#).
 - Optionally, you can enable access to the OpenSearch dashboard to view results. Refer to [enable fine grain access control](#).
4. Login using a role that allows [aws:CreateStack](#) permissions.
 5. Go to the MSF console dashboard and select **Create Streaming Application**.
 6. In **Choose a method to setup the stream processing application** select **Use a Blueprint**.
 7. Select **Real-time AI application blueprint** from the blueprints drop-down menu.
 8. Provide desired configurations. See [Create page configurations](#).
 9. Select **Deploy Blueprint** to start a CloudFormation deployment.
 10. Once the CloudFormation deployment is complete, go to the deployed Flink application. Check Runtime properties of the application.

11. You can choose to change/add runtime properties to your application. See [Runtime Properties Configuration](#) for details to configure these properties.

 **Note**

Note:

If you are using OpenSearch provisioned, please ensure you enabled [fine grain access control](#).

If your provisioned cluster is private, add `https://` to your OpenSearch Provisioned VPC endpoint URL and change `sink.os.endpoint` to point to this endpoint.

If your provisioned cluster is public, ensure your MSF application can access the internet. For more information, see [express-brokers-publication-merge type="documentation" url="managed-flink/latest/java/vpc-internet.html" >Internet and service access for a VPC-connected Managed Service for Apache Flink application](#).

12. Once you are satisfied with all the configurations, select Run. The application will start running.
13. Pump messages in your MSK cluster.
14. Navigate to the OpenSearch cluster and go to the OpenSearch dashboard.
15. On the dashboard, select **Discover** in the left menu. You should see persisted documents along with their vector embeddings.
16. Refer to [Working with vector search collections](#) to see how you can use the vectors stored in the index.

Create page configurations

This topic describes create page configurations to refer to when specifying configurations for real-time AI application blueprints.

Application name

Existing field in MSF, give any name to your application.

MSK cluster

Select the MSK cluster you created during setup from the dropdown list.

Topics

Add the name of the topic(s) which you created in the setup.

Input stream data type

Choose **String** if you will supply string input to the MSK stream.

Choose **JSON** if the input in the MSK stream is JSON. In **JSON keys embedded**, write the names of the fields in your input JSON whose value you want to send to Bedrock for generating embeddings.

Bedrock embedding model

Select one from the list. Ensure that you have model access for the model you choose, otherwise the stack might fail. See [Add or remove access to Amazon Bedrock foundation models](#).

OpenSearch cluster

Select the cluster you created from the dropdown.

OpenSearch vector index name

Select the vector index that you created in the above step.

Amazon MSK quota

Your AWS account has default quotas for Amazon MSK. Unless otherwise stated, each per-account quota is Region-specific within your AWS account.

Requesting a quota increase in Amazon MSK

You can request a quota increase for each Region using the Service Quotas console, AWS CLI, or a support case. If an adjustable quota isn't available in the Service Quotas console, use the AWS Support Center Console to create a [service quota increase case](#).

Support could approve, deny, or partially approve your quota increase requests. Increases aren't granted immediately, and can take a few days to take effect.

To request an increase using the Service Quotas console

1. Open the Service Quotas console at <https://console.aws.amazon.com/servicequotas/>.
2. From the navigation bar, at the top of the screen, select a Region.
3. In the left navigation pane, choose **AWS services**.
4. In the Find services box, type **msk**, and then choose **Amazon Managed Streaming for Apache Kafka (MSK)**.
5. In **Service quotas**, choose the **Quota name** for which you want to request an increase. For example, **Number of brokers per account**.
6. Choose **Request increase at account level**.
7. For **Increase quota value**, enter a new quota value.
8. Choose **Request**.
9. (Optional) To view any pending or recently resolved requests in the console, choose **Dashboard** in the left navigation pane. For pending requests, choose the status of the request to open the request receipt. The initial status of a request is **Pending**. After the status changes to **Quota requested**, you'll see the case number with Support. Choose the case number to open the ticket for your request.

For more information, including how to use the AWS CLI or SDKs to request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

Amazon MSK Standard broker quota

Standard broker quota

Dimension	Quota	Notes
Brokers per account	90	To request higher quota, go to the Service Quotas console .
Brokers per cluster	30 for ZooKeeper-based clusters 60 for KRaft-based clusters	To request higher quota, go to the Service Quotas console .
Minimum storage per broker	1 GiB	
Maximum storage per broker	16384 GiB	
Maximum TCP connections per broker (IAM Access control)	3000	To increase this limit, you can adjust the <code>listener.name.client_iam.max.connections</code> or the <code>listener.name.client_iam_public.max.connections</code> configuration property using the Kafka <code>AlterConfig</code> API or the <code>kafka-configs.sh</code> tool. It's important to note that increasing either property to a high value can result in unavailability.
Maximum TCP connections rate per broker (IAM)	100 per second (M5 and M7g instance sizes) 4 per second (t3 instance size)	To handle retries on failed connections, you can set the <code>reconnect.backoff.ms</code> configuration parameter on the client side. For example, if you want a client to retry

Dimension	Quota	Notes
		connections after 1 second, set <code>reconnect.backoff.ms</code> to 1000. For more information, see reconnect.backoff.ms in the Apache Kafka documentation.
Maximum TCP connections per broker (non-IAM)	N/A	MSK does not enforce connection limits for non-IAM auth. You should monitor other metrics such as CPU and memory usage to ensure you do not overload your cluster because of excessive connections.
Configurations per account	100	<p>To request higher quota, go to the Service Quotas console.</p> <p>To update the configuration or the Apache Kafka version of an MSK cluster, first ensure the number of partitions per broker is under the limits described in Right-size your cluster: Number of partitions per Standard broker.</p>
Configuration revisions per account	50	

Amazon MSK Express broker quota

Express broker quotas

Dimension	Quota	Notes
Brokers per account	90	To request higher quota, go to the Service Quotas console .
Brokers per cluster	30	To request higher quota, go to the Service Quotas console .
Maximum storage	Unlimited	
Maximum TCP connections per broker (IAM Access control)	3000	<p>To increase the connection limit, adjust one of the following configuration properties using the Kafka AlterConfig API or the kafka-configs.sh tool:</p> <ul style="list-style-type: none"><code>listener.name.client_iam.max.connections</code><code>listener.name.client_iam_public.max.connections</code> <p>Setting these properties to a high value might result in cluster unavailability.</p>
Maximum TCP connections rate per broker (IAM)	100 per second	To handle retries on failed connections, you can set the <code>reconnect.backoff.ms</code> configuration parameter on

Dimension	Quota	Notes
		the client side. For example, if you want a client to retry connections after 1 second, set <code>reconnect.backoff.ms</code> to 1000. For more information, see reconnect.backoff.ms in the Apache Kafka documentation.
Maximum TCP connections per broker (non-IAM)	N/A	MSK does not enforce connection limits for non-IAM auth. You should however monitor other metrics such as CPU and memory usage to ensure you do not overload your cluster because of excessive connections.
Configurations per account	100	To request higher quota, go to the Service Quotas console . To update the configuration or the Apache Kafka version of an MSK cluster, first ensure the number of partitions per broker is under the limits described in Right-size your cluster: Number of partitions per Express broker .
Configuration revisions per account	50	
Max Ingress per broker	Recommended: 15.6 - 500.0 MBps	Based on instance size.

Dimension	Quota	Notes
Max Egress per broker	Recommended: 31.2 - 1000.0 MBps	Based on instance size.

Express broker throughput throttle limits by broker size

The following table lists the recommended and maximum throughput throttle limit related to ingress and egress for different broker sizes. In this table, the recommended throughput is represented as Sustained performance, which is the threshold up to which your applications won't experience any performance degradation. If you operate beyond these limits on either dimension, you might get more throughput, but you might also experience performance degradation. Maximum quota is the threshold at which your cluster will throttle read/write traffic. Your applications won't be able to operate beyond this threshold.

Instance size	Sustained performance (MBps) for ingress	Maximum quota (MBps) for ingress	Sustained performance (MBps) for egress	Maximum quota (MBps) for egress
express.m7g.large	15.6	23.4	31.2	58.5
express.m7g.xlarge	31.2	46.8	62.5	117
express.m7g.2xlarge	62.5	93.7	125	234.2
express.m7g.4xlarge	124.9	187.5	249.8	468.7
express.m7g.8xlarge	250	375	500	937.5
express.m7g.12xlarge	375	562.5	750	1406.2

Instance size	Sustained performance (MBps) for ingress	Maximum quota (MBps) for ingress	Sustained performance (MBps) for egress	Maximum quota (MBps) for egress
express.m7g.16xlarge	500	750	1000	1875

MSK Replicator quotas

- A maximum of 15 MSK Replicators per account.
- MSK Replicator only replicates up to 750 topics in sorted order. If you need to replicate more topics, we recommend that you create a separate Replicator. Go to the [Service Quotas console](#), if you need support for more than 750 topics per Replicator. You can monitor the number of topics being replicated using the "TopicCount" metric.
- A maximum ingress throughput of 1GB per second per MSK Replicator. Request a higher quota by going through the [Service Quotas console](#).
- MSK Replicator Record Size - A maximum of 10MB record size (message.max.bytes). Request a higher quota by going through the [Service Quotas console](#).

MSK Serverless quota

The quotas specified in the following table are per cluster, unless otherwise stated.

Note

If you experience any issue with service quota limits, create a support case with your use case and requested limit.

Dimension	Quota	Quota violation result
Maximum ingress throughput	200 MBps	Slowdown with throttle duration in response

Dimension	Quota	Quota violation result
Maximum egress throughput	400 MBps	Slowdown with throttle duration in response
Maximum retention duration	Unlimited	N/A
Maximum number of client connections	3000	Connection close
Maximum connection attempts	100 per second	Connection close
Maximum message size	8 MiB	Request fails with ErrorCode: <i>INVALID_REQUEST</i>
Maximum request rate	15,000 per second	Slowdown with throttle duration in response
Maximum rate of topic management APIs requests rate	2 per second	Slowdown with throttle duration in response
Maximum fetch bytes per request	55 MB	Request fails with ErrorCode: <i>INVALID_REQUEST</i>
Maximum number of consumer groups	500	JoinGroup request fails
Maximum number of partitions (leaders)	2400 for non-compacted topics. 120 for compacted topics. To request a service quota adjustment, create a support case with your use case and requested limit.	Request fails with ErrorCode: <i>INVALID_REQUEST</i>
Maximum rate of partition creation and deletion	250 in 5 minutes	Request fails with ErrorCode : <i>THROUGHPUT_QUOTA_EXCEEDED</i>

Dimension	Quota	Quota violation result
Maximum ingress throughput per partition	5 MBps	Slowdown with throttle duration in response
Maximum egress throughput per partition	10 MBps	Slowdown with throttle duration in response
Maximum partition size (for compacted topics)	250 GB	Request fails with ErrorCode : THROUGHPUT_QUOTA_EXCEEDED
Maximum number of client VPCs per serverless cluster	5	
Maximum number of serverless clusters per account	10. To request a service quota adjustment, create a support case with your use case and requested limit.	

MSK Connect quota

- Up to 100 custom plugins.
- Up to 100 worker configurations.
- Up to 60 connect workers. If a connector is set up to have auto scaled capacity, then the maximum number of workers that the connector is set up to have is the number MSK Connect uses to calculate the quota for the account.
- Up to 10 workers per connector.

To request higher quota for MSK Connect, go to the [Service Quotas console](#).

Document history for Amazon MSK Developer Guide

The following table describes the important changes to the Amazon MSK Developer Guide.

Latest documentation update: June 25, 2024

Change	Description	Date
Express broker feature added. Developer Guide topics reorganized.	MSK supports Standard and new Express brokers.	2024-11-6
Graviton upgrade in place feature added.	You can update your cluster broker size from M5 or T3 to M7g, or from M7g to M5.	2024-6-25
3.4.0 end of support date announced.	End of support date for Apache Kafka version 3.4.0 is June 17, 2025.	2024-6-24
Broker removal feature added.	You can reduce your provisioned cluster's storage and compute capacity by removing sets of brokers, with no availability impact, data durability risk, or disruption to your data streaming applications.	2024-5-16
WriteDataIdempotently added to AWSMSKReplicatorExecutionRole	WriteDataIdempotently permission is added to AWSMSKReplicatorExecutionRole policy to support data replication between MSK clusters.	2024-5-16
Graviton M7g brokers released in Brazil and Bahrain.	Amazon MSK now supports South America (sa-east-	2024-2-07

Change	Description	Date
	1, São Paulo) and Middle East (me-south-1, Bahrain) region availability of M7g brokers using AWS Graviton processors (custom Arm-based processors built by Amazon Web Services).	
Release Graviton M7g brokers to China region	Amazon MSK now supports China region availability of M7g brokers using AWS Graviton processors (custom Arm-based processors built by Amazon Web Services).	2024-01-11
Amazon MSK Kafka version support policy	Added explanation of the Amazon MSK supported Kafka version support policy. For more information, see Apache Kafka versions .	2023-12-08
New service execution role policy to support Amazon MSK Replicator.	Amazon MSK added new <code>AWSMSKReplicatorExecutionRole</code> policy to support Amazon MSK Replicator. For more information, see AWS managed policy: AWSMSKReplicatorExecutionRole .	2023-12-06
M7g Graviton support	Amazon MSK now supports M7g brokers using AWS Graviton processors (custom Arm-based processors built by Amazon Web Services).	2023-11-27

Change	Description	Date
Amazon MSK Replicator	Amazon MSK Replicator is a new feature that you can use to replicate data between Amazon MSK clusters. Amazon MSK Replicator includes an update to the AmazonMSKFullAccess policy. For more information, see AWS managed policy: AmazonMSKFullAccess .	2023-09-28
Updated for IAM best practices.	Updated guide to align with the IAM best practices . For more information, see Security best practices in IAM .	2023-03-08
Service-linked role updates to support multi-VPC private connectivity	Amazon MSK now includes AWSServiceRoleForKafka service-linked role updates to manage network interfaces and VPC endpoints in your account that make cluster brokers accessible to clients in your VPC. Amazon MSK uses permissions to DescribeVpcEndpoints , ModifyVpcEndpoint and DeleteVpcEndpoints . For more information, see Service-linked roles for Amazon MSK .	2023-03-08

Change	Description	Date
Support for Apache Kafka 2.7.2	Amazon MSK now supports Apache Kafka version 2.7.2. For more information, see Supported Apache Kafka versions .	2021-12-21
Support for Apache Kafka 2.6.3	Amazon MSK now supports Apache Kafka version 2.6.3. For more information, see Supported Apache Kafka versions .	2021-12-21
MSK Serverless Prerelease	MSK Serverless is a new feature that you can use to create serverless clusters. For more information, see MSK Serverless .	2021-11-29
Support for Apache Kafka 2.8.1	Amazon MSK now supports Apache Kafka version 2.8.1. For more information, see Supported Apache Kafka versions .	2021-09-30
MSK Connect	MSK Connect is a new feature that you can use to create and manage Apache Kafka connectors. For more information, see Understand MSK Connect .	2021-09-16
Support for Apache Kafka 2.7.1	Amazon MSK now supports Apache Kafka version 2.7.1. For more information, see Supported Apache Kafka versions .	2021-05-25

Change	Description	Date
Support for Apache Kafka 2.8.0	Amazon MSK now supports Apache Kafka version 2.8.0. For more information, see Supported Apache Kafka versions .	2021-04-28
Support for Apache Kafka 2.6.2	Amazon MSK now supports Apache Kafka version 2.6.2. For more information, see Supported Apache Kafka versions .	2021-04-28
Support for Updating Broker Type	You can now change the broker type for an existing cluster. For more information, see Update the Amazon MSK cluster broker size .	2021-01-21
Support for Apache Kafka 2.6.1	Amazon MSK now supports Apache Kafka version 2.6.1. For more information, see Supported Apache Kafka versions .	2021-01-19
Support for Apache Kafka 2.7.0	Amazon MSK now supports Apache Kafka version 2.7.0. For more information, see Supported Apache Kafka versions .	2020-12-29

Change	Description	Date
No New Clusters with Apache Kafka Version 1.1.1	You can no longer create a new Amazon MSK cluster with Apache Kafka version 1.1.1. However, if you have existing MSK clusters that are running Apache Kafka version 1.1.1, you can continue using all of the currently-supported features on those existing clusters. For more information, see Apache Kafka versions .	2020-11-24
Consumer-Lag Metrics	Amazon MSK now provides metrics that you can use to monitor consumer lag. For more information, see Monitor an Amazon MSK Provisioned cluster .	2020-11-23
Support for Cruise Control	Amazon MSK now supports LinkedIn's Cruise Control. For more information, see Use LinkedIn's Cruise Control for Apache Kafka with Amazon MSK .	2020-11-17
Support for Apache Kafka 2.6.0	Amazon MSK now supports Apache Kafka version 2.6.0. For more information, see Supported Apache Kafka versions .	2020-10-21

Change	Description	Date
Support for Apache Kafka 2.5.1	Amazon MSK now supports Apache Kafka version 2.5.1. With Apache Kafka version 2.5.1, Amazon MSK supports encryption in transit between clients and ZooKeeper endpoints. For more information, see Supported Apache Kafka versions .	2020-09-30
Application Auto-Expansion	You can configure Amazon Managed Streaming for Apache Kafka to automatically expand your cluster's storage in response to increased usage. For more information, see Automatic scaling for clusters .	2020-09-30
Support for Username and Password Security	Amazon MSK now supports logging into clusters using a username and password. Amazon MSK stores credentials in AWS Secrets Manager. For more information, see SASL/SCRAM authentication .	2020-09-17
Support for Upgrading the Apache Kafka Version of an Amazon MSK Cluster	You can now upgrade the Apache Kafka version of an existing MSK cluster.	2020-05-28
Support for T3.small Broker Nodes	Amazon MSK now supports creating clusters with brokers of Amazon EC2 type T3.small.	2020-04-08

Change	Description	Date
Support for Apache Kafka 2.4.1	Amazon MSK now supports Apache Kafka version 2.4.1.	2020-04-02
Support for Streaming Broker Logs	Amazon MSK can now stream broker logs to CloudWatch Logs, Amazon S3, and Amazon Data Firehose. Firehose can, in turn, deliver these logs to the destinations that it supports, such as OpenSearch Service.	2020-02-25
Support for Apache Kafka 2.3.1	Amazon MSK now supports Apache Kafka version 2.3.1.	2019-12-19
Open Monitoring	Amazon MSK now supports open monitoring with Prometheus.	2019-12-04
Support for Apache Kafka 2.2.1	Amazon MSK now supports Apache Kafka version 2.2.1.	2019-07-31
General Availability	New features include tagging support, authentication, TLS encryption, configurations, and the ability to update broker storage.	2019-05-30
Support for Apache Kafka 2.1.0	Amazon MSK now supports Apache Kafka version 2.1.0.	2019-02-05