Amazon Managed Streaming for Apache Kafka

Developer Guide

Amazon Managed Streaming for Apache Kafka: Developer Guide

Table of Contents

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.

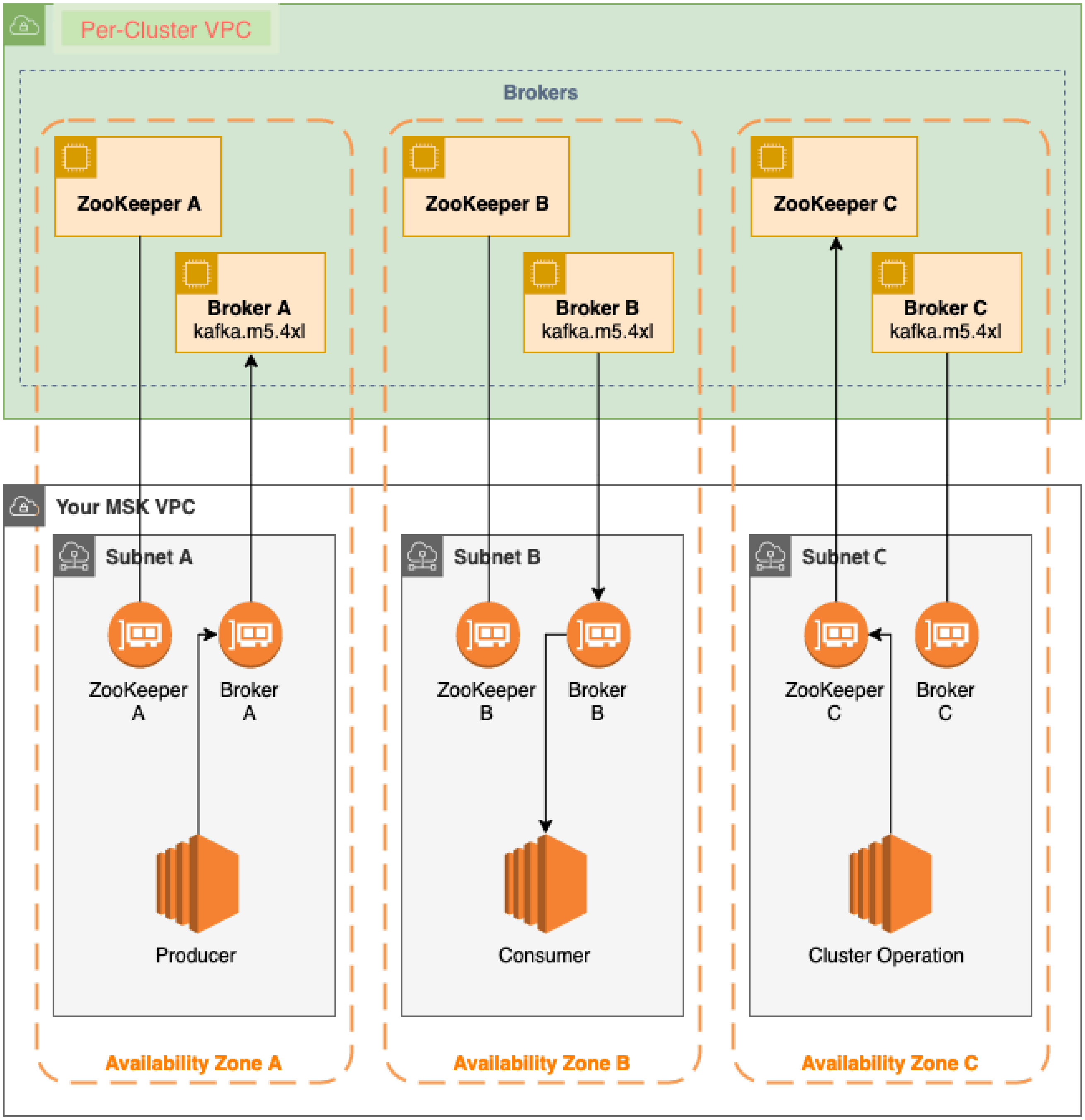
- Create an Amazon MSK cluster by following the Getting started using Amazon MSK (p. 5) tutorial.
- Dive deeper into the functionality of Amazon MSK in Amazon MSK: How it works (p. 10).
- Run Apache Kafka without having to manage and scale cluster capacity with MSK Serverless (p. 56).
- Use MSK Connect (p. 64) to stream data to and from your Apache Kafka cluster.

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” (p. 176).

The following diagram provides an overview of how Amazon MSK works.



The diagram demonstrates the interaction between the following components:

- **Broker nodes** — When creating an Amazon MSK cluster, you specify how many broker nodes you want Amazon MSK to create in each Availability Zone. In the example cluster shown in this diagram, there's one broker per Availability Zone. Each Availability Zone has its own virtual private cloud (VPC) subnet.
- **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.
- **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

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 an administrative 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.

Getting 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 Amazon MSK cluster

In this step of Getting Started Using Amazon MSK (p. 5), 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**, choose **Provisioned** as the **Cluster type**.
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 Cluster states (p. 104).

Next Step

Step 2: Create a client machine (p. 5)

Step 2: Create a client machine

In this step of Getting Started Using Amazon MSK (p. 5), 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. Choose **Launch instances**.
3. Enter a **Name** for your client machine, such as **`MSKTutorialClient`**.
4. Leave **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 **`MSKKeyPair`** 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.
9. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
10. In the navigation pane, choose **Security Groups**. Find the security group whose ID you saved in the section called "Step 1: Create a cluster" (p. 5).
11. In the **Inbound Rules** tab, choose **Edit inbound rules**.
12. Choose **Add rule**.
13. 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.
14. Choose **Save rules**. Now the cluster's security group can accept traffic that comes from the client machine's security group.

Next Step

Step 3: Create a topic (p. 6)

Step 3: Create a topic

In this step of Getting Started Using Amazon MSK (p. 5), you install Apache Kafka client libraries and tools on the client machine, and then you create a topic.

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 2: Create a client machine (p. 5).
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 install java-1.8.0
```

5. Run the following command to download Apache Kafka.

```
wget https://archive.apache.org/dist/kafka/2.6.2/kafka_2.12-2.6.2.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_2.12-2.6.2.tgz
```

7. Go to the **kafka_2.12-2.6.2** directory.
8. Open the Amazon MSK console at https://console.aws.amazon.com/msk/.
9. 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.
10. Choose **View client information**.
11. Copy the connection string for plaintext authentication.
12. Run the following command, replacing *BootstrapServerString* with the connection string that you obtained in the previous step.

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

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

Next Step

Step 4: Produce and consume data (p. 7)

Step 4: Produce and consume data

In this step of Getting Started Using Amazon MSK (p. 5), you produce and consume data.

To produce and consume messages

1. Go to the `bin` folder of the Apache Kafka installation on the client machine, and create a text file named `client.properties` with the following contents.

```
security.protocol=PLAINTEXT
```

2. Run the following command to start a console producer. Replace *BootstrapServerString* with the plaintext connection string that you obtained in the section called “Step 3: Create a topic” (p. 6). For instructions on how to retrieve this connection string, see Getting the bootstrap brokers for an Amazon MSK cluster (p. 21).

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

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. Keep the connection to the client machine open, and then open a second, separate connection to that machine in a new window.

5. 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 client.properties --topic
    MSKTutorialTopic --from-beginning
   ```

   You start seeing the messages you entered earlier when you used the console producer command.
6. Enter more messages in the producer window, and watch them appear in the consumer window.

Next Step

Step 5: Use Amazon CloudWatch to view Amazon MSK metrics (p. 8)

Step 5: Use Amazon CloudWatch to view Amazon MSK metrics

In this step of Getting Started Using Amazon MSK (p. 5), 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 6: Delete the AWS resources created for this tutorial (p. 8)

Step 6: Delete the AWS resources created for this tutorial

In the final step of Getting Started Using Amazon MSK (p. 5), 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**.

Amazon MSK: How it works

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

Creating an Amazon MSK cluster

> **Important**
> You can't change the VPC for an Amazon MSK cluster after you create the cluster.

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

You need two subnets in two different Availability Zones in the US West (N. California) Region. 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.

Broker types

When you create an Amazon MSK cluster, you specify the type of brokers that you want it to have. Amazon MSK supports the following broker types:

- 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

M5 brokers have higher baseline throughput performance than T3 brokers and are recommended for production workloads. M5 brokers can also have more partitions per broker than T3 brokers. Use M5 brokers if you are running larger production-grade workloads or require a greater number of partitions. To learn more about M5 instance types, see Amazon EC2 M5 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 instance types, see Amazon EC2T3 Instances.

For more information on how to choose broker types, see *Best practices* (p. 188).

Creating a cluster using the AWS Management Console

1. Open the Amazon MSK console at https://console.aws.amazon.com/msk/.
2. Choose **Create cluster**.
3. Specify a name for the cluster.
4. In the VPC list, choose the VPC you want to use for the cluster. You can also specify which version of Apache Kafka you want Amazon MSK to use to create the cluster.
5. Specify two subnets if you're using one of the following Regions: South America (São Paulo), Canada (Central), and US West (N. California). In other Regions where Amazon MSK is available, you can specify either two or three subnets. The subnets that you specify must be in different Availability Zones.
6. Choose the kind of configuration you want. For information about Amazon MSK configurations, see *Configuration* (p. 39).
7. Specify the type and 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.
8. (Optional) Assign tags to your cluster. Tags are optional. For more information, see the section called “Tagging a cluster” (p. 37).
9. You can adjust the storage volume per broker. After you create the cluster, you can increase the storage volume per broker but you can't decrease it.
10. Choose the settings you want for encrypting data in transit. By default, Amazon MSK encrypts data as it transits between brokers within a cluster. If you don't want to encrypt data as it transits between brokers, clear the check box labeled **Enable encryption within the cluster**.
11. Choose one of the three settings for encrypting data as it transits between clients and brokers. For more information, see the section called “Encryption in transit” (p. 107).
12. Choose the kind of KMS key that you want to use for encrypting data at rest. For more information, see the section called “Encryption at rest” (p. 107).
13. If you want to authenticate the identity of clients, choose **Enable TLS client authentication** by selecting the box next to it. For more information about authentication, see the section called “Mutual TLS authentication” (p. 131).
14. Choose the monitoring level you want. This determines the set of metrics you get. For more information, see *Monitoring a cluster* (p. 155).
15. (Optional) Choose **Advanced settings**, and then choose **Customize settings**. You can specify 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 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.
16. Choose **Create cluster**.

17. 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 Cluster states (p. 104).

Creating a 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
Specify exactly two subnets if you are using one of the following Regions: South America (São Paulo), Canada (Central), and US West (N. California). For other Regions where Amazon MSK is available, you can specify either two or three subnets. The subnets that you specify must be in distinct Availability Zones. When you create a cluster, Amazon MSK distributes the broker nodes evenly across 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 ??? (p. 155). 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 "2.2.1" --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 Cluster states (p. 104).

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

Creating a cluster with a custom Amazon MSK configuration using the AWS CLI

For information about custom Amazon MSK configurations and how to create them, see *Configuration* (p. 39).

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 "1.1.1" --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"
}
```

Creating a cluster using the API

To create a cluster using the API, see CreateCluster.

Tiered storage

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.

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.
- It best serves applications that must retain data for longer than a day, or more than 1-2 TB per broker.
- Data between brokers and the tiered storage moves within the VPC and doesn't travel through the internet.
- You can use tiered storage at a cluster level, but this doesn't automatically enable tiered storage for all topics by default.
- 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

- 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 3: Create a topic (p. 6).
- The Amazon MSK cluster with tiered storage enabled must use version 2.8.2.tiered.

Tiered storage constraints and limitations

Tiered storage has the following constraints and limitations:

- Tiered storage applies only to provisioned mode clusters.
- Tiered storage doesn’t support broker type 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).
- Compacted topics can’t use the tiered storage feature.
- You can’t re-enable tiered storage at a topic level after you have disabled it. You can't edit the storage mode for a cluster that uses tiered storage. Amazon MSK only supports editing the cluster storage mode when a cluster uses EBS storage.

- 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.

How log segments are copied to tiered storage

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 :

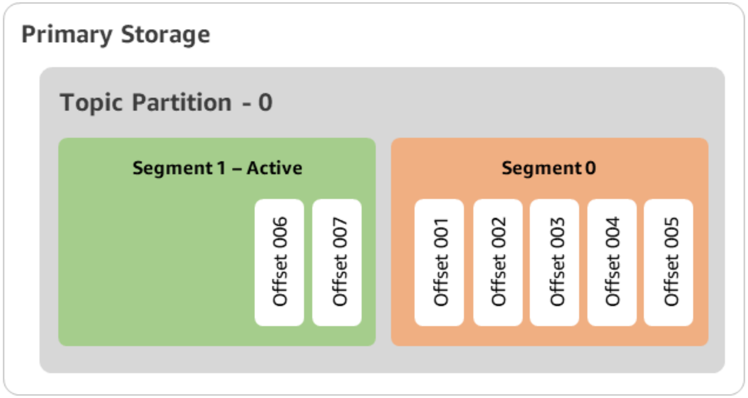
- You define retention in Apache Kafka with two settings: log.retention.ms (time) and log.retention.bytes (size). These settings determine the total duration and size of the data that Apache Kafka retains in the cluster. Whether or not you enable tiered storage mode, you set these configurations at the 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 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.

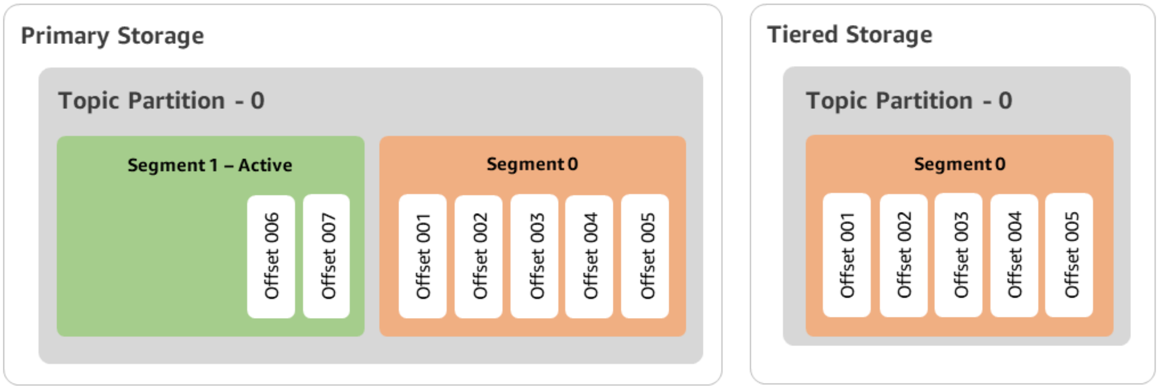
Time T0 - Before you enable tiered storage.

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.



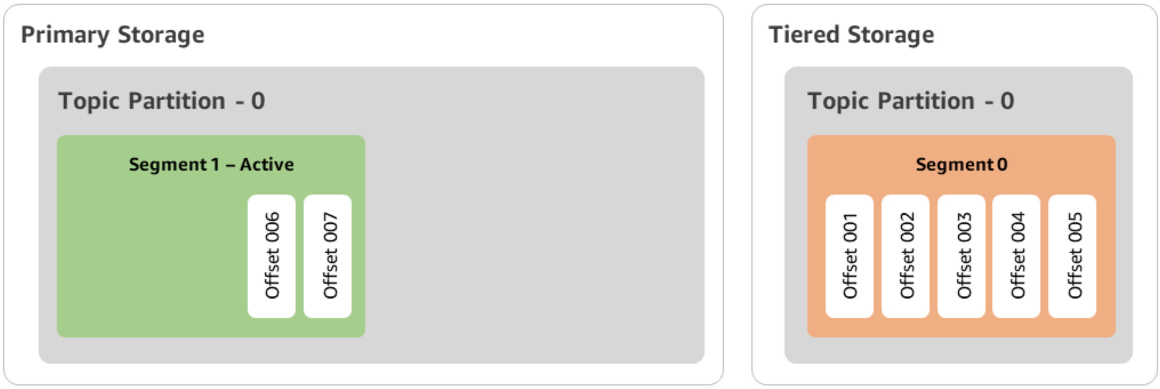
Time T1 (< 2 days) - Tiered storage enabled. Segment 0 copied to tiered storage.

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 rerention settings yet for any of the messages in segment 0 and segment 1. (local.retention.bytes/ms, retention.ms/bytes)



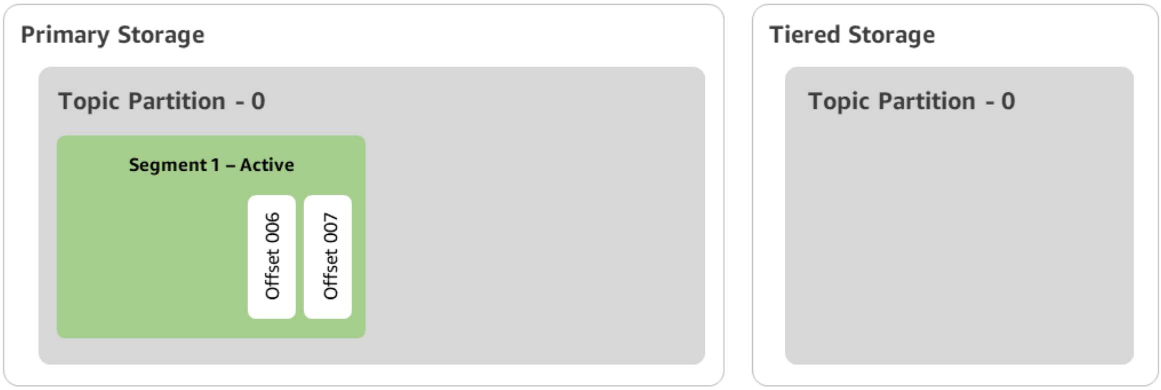
Time T2 - Local retention in effect.

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.



Creating a Amazon MSK cluster with tiered storage with 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 you want Amazon MSK to use to create the cluster. Version **2.8.2.tiered** supports tiered storage.
7. Specify a type 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**.

Creating 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 Creating a Kafka topic with tiered storage enabled (p. 18).

See create-cluster for a complete list of supported attributes for cluster creation.

```
aws tiered-storage create-cluster \
 –cluster-name "MessagingCluster" \
 –broker-node-group-info file://brokernodegroupinfo.json \
 –number-of-broker-nodes 3 \
--kafka-version "2.8.2.tiered" \
--storage-mode "TIERED"
```

Creating a Kafka topic with tiered storage enabled

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 topic (p. 18).

```
bin/kafka-topics.sh --create --bootstrap-server $bs --replication-factor 2 --
partitions 6 --topic MSKTutorialTopic --config remote.storage.enable=true --config
 local.retention.ms=100000 --config retention.ms=604800000 --config segment.bytes=134217728
```

Enabling and disabling tiered storage on an existing 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 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 $bs --alter --entity-type topics --entity-name
 MSKTutorialTopic --add-config 'remote.storage.enable=true'
```

Disabling tiered storage on an existing 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.

Deleting an Amazon MSK cluster

Note
If your cluster has an auto-scaling policy, we recommend that you remove the policy before you delete the cluster. For more information, see Automatic scaling (p. 25).

Deleting a cluster using the AWS Management Console

1. Open the Amazon MSK console at https://console.aws.amazon.com/msk/.
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.

Deleting a cluster 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 “Listing clusters” (p. 22).

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

Deleting a cluster using the API

To delete a cluster using the API, see DeleteCluster.

Getting the Apache ZooKeeper connection string for an Amazon MSK cluster

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 “Listing clusters” (p. 22).
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.

Getting the bootstrap brokers for an Amazon MSK cluster

Getting the bootstrap brokers 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. 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 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 "Listing clusters" (p. 22).

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

For an MSK cluster that uses the section called "IAM access control" (p. 122), 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).

Getting the bootstrap brokers using the API

To get the bootstrap brokers using the API, see GetBootstrapBrokers.

Listing Amazon MSK clusters

Listing clusters 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 details.

Listing clusters using the AWS CLI

Run the following command.

```
aws kafka list-clusters
```

Listing clusters using the API

To list clusters using the API, see ListClusters.

Provisioning storage throughput

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 type `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.

Throughput bottlenecks

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 types, as shown in the following table.

Broker type	Amazon EC2 to Amazon EBS network throughput (MBps)
kafka.m5.4xlarge	593.75
kafka.m5.8xlarge	850
kafka.m5.12xlarge	1187.5
kafka.m5.16xlarge	1700
kafka.m5.24xlarge	2375

Measuring storage throughput

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.

```
Average storage throughput in MiB/s = (Sum(VolumeReadBytes) + Sum(VolumeWriteBytes)) / (60
 * 1024 * 1024)
```

For information about the `VolumeReadBytes` and `VolumeWriteBytes` metrics, see the section called “PER_BROKER Level monitoring” (p. 160).

Configuration update

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 type	num.replica.fetchers
kafka.m5.4xlarge	4
kafka.m5.8xlarge	8
kafka.m5.12xlarge	14

Broker type	num.replica.fetchers
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.

Provisioning storage throughput 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. 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**.

Provisioning storage throughput using the AWS CLI

This section 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":"2.2.1",
        "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
```

Provisioning storage throughput using the API

To configure provisioned storage throughput while creating a cluster, use CreateClusterV2.

Scaling up 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 *Best practices* (p. 188).

You can use tiered storage to scale up to unlimited amounts of storage for your broker. See, Tiered storage (p. 13).

Topics

Automatic scaling

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 brokers per cluster (p. 188).
- 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 Migration (p. 152).
- 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*.

Auto-scaling policy details

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 (p. 173).

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.

Setting 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.

Setting up automatic scaling 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 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.

Setting up automatic scaling using the CLI

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

Setting up automatic-scaling using the API

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

Manual scaling

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 “Listing clusters” (p. 22).

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.

Updating the broker type

You can scale your MSK cluster on demand by changing the type (the size or family) of your brokers without reassigning Apache Kafka partitions. Changing the type 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 type for all the brokers in a given cluster. This section describes how to update the broker type for your MSK cluster. The broker-type 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-type update. For information about how to make a cluster highly available during a broker-type update, see the section called “Build highly available clusters” (p. 189). To further reduce any potential impact on productivity, you can perform the broker-type update during a period of low traffic.

During a broker-type 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 type, 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 type if the number of partitions per broker exceeds the maximum number specified in the section called “ Right-size your cluster: Number of partitions per broker” (p. 188).

Updating the broker type 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 the broker type.
3. On the details page for the cluster, find the **Brokers summary** section, and choose **Edit broker type**.
4. Choose the broker type you want from the list.
5. Save changes.

Updating the broker type 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 “Listing clusters” (p. 22).

   Replace *`Current-Cluster-Version`* with the current version of the cluster and *`TargetType`* with the new type that you want the brokers to be. To learn more about broker types, see the section called “Broker types” (p. 10).

```
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 type using the API

To update the broker type using the API, see UpdateBrokerType.

Updating 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 broker" (p. 188). 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 *Configuration* (p. 39).

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 "Listing clusters" (p. 22).

   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.

Updating the configuration of a cluster using the API

To use the API to update the configuration of a cluster, see UpdateClusterConfiguration.

Expanding 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” (p. 191).

Expanding a 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 number of brokers you want to increase.
3. On the cluster details page, choose the **Edit** button next to the **Cluster-Level Broker Details** heading.
4. Enter the number of brokers that you want the cluster to have per Availability Zone and then choose **Save changes**.

Expanding a 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 “Listing clusters” (p. 22).

   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.

Expanding a cluster using the API

To increase the number of brokers in a cluster using the API, see UpdateBrokerCount.

Updating security settings of a cluster

Use this 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 *Security* (p. 106).

Updating a cluster's security settings using the AWS Management Console

1. Open the Amazon MSK console at https://console.aws.amazon.com/msk/.
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 a cluster's 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.

Updating a cluster's security settings using the API

To update the security settings for a cluster using the API, see UpdateSecurity.

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.

Rebooting 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 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.

Rebooting a broker 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.

Rebooting a broker 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 "Listing clusters" (p. 22).

   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`.

Rebooting a broker using the API

To reboot a broker in a cluster using the API, see RebootBroker.

Tagging 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

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`

Tracking 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: `_ . / = + - @`.

Tagging 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

Amazon MSK configuration

Amazon Managed Streaming for Apache Kafka provides a default configuration for brokers, topics, and Apache ZooKeeper 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.

Topics

Custom 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 configuration” (p. 45). For more information about configuration properties, see Apache Kafka Configuration.

Apache Kafka configuration properties

Name	Description
allow.everyone.if.no.acl.found	If you want to set this property tofalse, first make sure you define Apache Kafka ACLs for your cluster. If you set this property to falseand 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.
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.

Name	Description
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.
group.min.session.timeout.ms	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.
leader.imbalance.per.broker.percentage	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.
log.cleaner.delete.retention.ms	Amount of time that you want Apache Kafka to retain deleted records. The minimum value is 0.
log.cleaner.min.cleanable.ratio	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.
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`.
log.flush.interval.messages	Number of messages that accumulate on a log partition before messages are flushed to disk.
log.flush.interval.ms	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 log.flush.scheduler.interval.ms is used. The minimum value is 0.

Name	Description
log.message.timestamp.difference.max.ms	The maximum time difference between the timestamp when a broker receives a message and the timestamp specified in the message. If log.message.timestamp.type=CreateTime, a message is rejected if the difference in timestamp exceeds this threshold. This configuration is ignored if log.message.timestamp.type=LogAppendTime.
log.message.timestamp.type	Specifies if the timestamp in the message is the message creation time or the log append time. The allowed values are `CreateTime` and `LogAppendTime`.
log.retention.bytes	Maximum size of the log before deleting it.
log.retention.hours	Number of hours to keep a log file before deleting it, tertiary to the log.retention.ms property.
log.retention.minutes	Number of minutes to keep a log file before deleting it, secondary to log.retention.ms property. If you don't set this value, the value in log.retention.hours is used.
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	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 single record. You can set this value per topic with the topic level max.message.bytes config.

Name	Description
min.insync.replicas	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 NotEnoughReplicas or NotEnoughReplicasAfterAppend). 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.
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 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.
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.
offsets.topic.replication.factor	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.
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.

Name	Description
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 message.max.bytes (broker config) or max.message.bytes (topic config) properties specify the maximum record batch size that the broker accepts.
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. MinValue: 10000 MaxValue (inclusive) = 30000
replica.selector.class	The fully-qualified class name that implements ReplicaSelector. 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 `org.apache.kafka.common.replica.RackAwareReplicaS` For more information, see the section called "Apache Kafka version 2.4.1 (use 2.4.1.1 instead)" (p. 178).
replica.socket.receive.buffer.bytes	The socket receive buffer for network requests.
socket.receive.buffer.bytes	The SO_RCVBUF 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.
socket.request.max.bytes	The maximum number of bytes in a socket request.
socket.send.buffer.bytes	The SO_SNDBUF 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.
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 InitProducerIdRequest. This prevents a client from too large of a timeout, and this can stall consumers that read from topics included in the transaction.
transaction.state.log.min.isr	Overridden min.insync.replicas configuration for the transaction topic.

Name	Description
transaction.state.log.replication.factor	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.
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 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.
unclean.leader.election.enable	Indicates if replicas not in the ISR set should serve as leader as a last resort, even though this might result in data loss.
zookeeper.connection.timeout.ms	Maximum time that the client waits to establish a connection to ZooKeeper. If you don't set this value, the value in zookeeper.session.timeout.ms is used.
zookeeper.session.timeout.ms	The Apache ZooKeeper session timeout in milliseconds. MinValue = 6000 MaxValue (inclusive) = 18000

To learn how you can create a custom MSK configuration, list all configurations, or describe them, see the section called “Configuration operations” (p. 51). To create an MSK cluster with a custom MSK configuration, or to update a cluster with a new custom configuration, see *How it works* (p. 10).

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 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 propeties 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 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.

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`

The 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
allow.everyone.if.no.acl.found	If no resource patterns match a specific resource, the resource has no associated ACLs. In this case, if you set this property ito true, everyone ican access the resource, not just the super users.		`true`
auto.create.topics.enable	Enables autocreation of a topic on the server.		`false`
auto.leader.rebalance.enable	Enables auto leader balancing. A background thread checks and initiates leader balance at regular intervals as if necessary.		`true`
default.replication.factor	Default replication factors for automatically created topics.		3 for 3-AZ clusters, 2 for 2-AZ clusters
local.retention.bytes	The maximum size of local log segments for a partition before		-2 for unlimited

Name	Description		Default value
	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. This configuration is used to determine how long the log segmentsshould remain in local storage. Existing log.retention.* configurations are retention configurations for the topic partition. This includes both local and remote storage. Valid values: integers in [-2; +Inf]		

Name	Description		Default value
local.retention.ms	The number of milliseconds to retain the local log segment before it gets deleted. If you don't set this value, the value in log.retention.ms 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 values local.retention.ms and local.retention.bytes are similar to log.retention. This configuration is used to determine how long the log segments should remain in local storage. Existing log.retention.* configurations are retention configurations for the topic partition. This includes both local and remote storage. Valid values: > 0		-2 for unlimited

Name	Description		Default value
min.insync.replicas	When a producer sets the value of 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 value doesn't meet this minimum, the producer raises an exception (either NotEnoughReplicas or NotEnoughReplicasAfterAppend). 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.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.		2 for 3-AZ clusters, 1 for 2-AZ clusters
num.io.threads	Number of threads that the server uses to produce requests, which may include disk I/O.		8
num.network.threads	Number of threads that the server uses to receive requests from the network andsend responses to the network.		5
num.partitions	Default number of log partitions per topic.		1

Name	Description		Default value
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
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.		DELETE
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.msk.disable.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
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

Name	Description		Default value
retention.ms	Mandatory field. Minimum time is 3 days. There is no default because the setting is mandatory. 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 ifrom tiered storage (that is, removed from the cluster). Valid values: integers in [-1; +Inf]		Minimum 259,200,000 milliseconds (3 days). -1 for infinite retention.
socket.receive.buffer.bytes	The SO_RCVBUF buffer of the socket sever 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 sever sockets. If the value is -1, the OS default is used.		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`
zookeeper.session.timeout.ms	The Apache ZooKeeper session timeout in milliseconds.		18000
zookeeper.set.acl	The set client to use secure ACLs.		`false`

For information on how to specify custom configuration values, see the section called “Custom configurations” (p. 39).

Guidelines for 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 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.

Amazon MSK configuration operations

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 *How it works* (p. 10).

This topic contains the following sections:

To create an MSK configuration

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

zookeeper.connection.timeout.ms = 1000
```

```
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.

To update an 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

   zookeeper.connection.timeout.ms = 1000

   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
    }
}
```

To delete an MSK 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"
}
```

To describe an MSK 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"
}
```

To describe an 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":
   "YXV0by5jcmVhdGUudG9waWNzLmVuYWJsZSA9IHRydWUKCgp6b29rZWVwZXIuY29ubmVjdGlvbi50aW1lb3V0Lm1zID0gMTAwM
  }
  ```

  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

  zookeeper.connection.timeout.ms = 1000

  log.roll.ms = 604800000
  ```

To list all MSK configurations in your account for the current 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"
        }
    ]
}
```

MSK Serverless

Note
MSK Serverless is available in the US East (Ohio), US East (N. Virginia), US West (Oregon), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), Europe (Frankfurt), Europe (Stockholm) and Europe (Ireland) 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
- AWS Glue Schema Registry for schema management
- Amazon Kinesis Data Analytics for Apache Flink-based stream processing
- AWS Lambda for event processing

MSK Serverless requires IAM access control for all clusters. For more information, see the section called “IAM access control” (p. 122).

For information about the service quota that apply to MSK Serverless, see the section called “Quota for serverless clusters” (p. 173).

To help you get started with serverless clusters, and to learn more about configuration and monitoring options for serverless clusters, see the following.

Topics

Getting started using 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

- Step 4: Create an Apache Kafka topic (p. 60)
- Step 5: Produce and consume data (p. 61)
- Step 6: Delete resources (p. 61)

Step 1: 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

Step 2: Create an IAM role (p. 57)

Step 2: Create an IAM role

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.

   Replace *region* with the code of the AWS Region where you created your cluster. Replace *Account-ID* with your account ID. Replace *msk-serverless-tutorial-cluster* with the name of your serverless cluster.

```
{
    "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/msk-serverless-tutorial-
cluster/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:*Topic*",
                "kafka-cluster:WriteData",
                "kafka-cluster:ReadData"
            ],
            "Resource": [
                "arn:aws:kafka:region:Account-ID:topic/msk-serverless-tutorial-cluster/
*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeGroup"
            ],
            "Resource": [
                "arn:aws:kafka:region:Account-ID:group/msk-serverless-tutorial-cluster/
*"
            ]
        }
    ]
}
```

   For instructions on how to write secure policies, see the section called “IAM access control” (p. 122).

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

Step 3: Create a client machine (p. 59)

Step 3: Create a client machine

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 Step 2: Create an IAM role (p. 57).
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
   ```

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/v1.1.1/aws-msk-iam-
   auth-1.1.1-all.jar
   ```

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

Step 4: Create an Apache Kafka topic (p. 60)

Step 4: 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

Step 5: Produce and consume data (p. 61)

Step 5: Produce and consume data

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.

Next Step

Step 6: Delete resources (p. 61)

Step 6: Delete resources

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 for serverless clusters

Amazon MSK sets broker configuration properties for serverless clusters. You can't change these broker configuration property settings. However, you can set the following topic configuration properties.

Configuration property	Default	Editable	Maximum allowed value
cleanup.policy	Delete	Yes, but only at topic creation time	
compression.type	Producer	Yes	
max.message.bytes	1048588	Yes	8 MiB
message.timestamp.difference.max.ms	long.max	Yes	
message.timestamp.type	CreateTime	Yes	
retention.bytes	250 GiB	Yes	250 GiB
retention.ms	1 day	Yes	1 day

You can also use Apache Kafka commands to set or modify topic-level configuration properties for new or existing topics. For more information about topic-level configuration properties and examples of how to set them, see Topic-Level Configs in the official Apache Kafka documentation.

Monitoring 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

Name	When visible	Dimensions	Description
			available for each broker and also for each topic.
`BytesOutPerSec`	After a consumer group consumes from a topic	Cluster Name, Topic	The number of bytes per second sent to clients. This metric is available for each broker and also for each topic.
`FetchMessageConversionsPerSec`	After a consumer group consumes from a topic	Cluster Name, Topic	The number of fetch message conversions per second for the broker.
`MaxEstimatedTimeLag`	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 broker.
`ProduceMessageConversionsPerSec`	After a producer writes to a topic	Cluster Name, Topic	The number of produce message conversions per second for the broker.
`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.

MSK Connect

What is 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 2.7.1, an open-source framework 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” (p. 64).

To learn about the AWS resources that you can create with MSK Connect, see the section called “Connectors” (p. 70), the section called “Plugins” (p. 73), and the section called “Workers” (p. 73).

For information about the MSK Connect API, see the Amazon MSK Connect API Reference.

Getting started using 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

Step 1: Set up required resources

In this step you create the following resources that you need for this getting-started scenario:

- An 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 role that allows the connector to write to the destination S3 bucket.
- 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 `mkc-tutorial-destination-bucket`.
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. Choose **Custom create**.
5. For the cluster name enter `mkc-tutorial-cluster`.
6. Under General cluster properties, choose **Provisioned** for the cluster type.
7. 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.
8. Under **Access control methods** ensure that only **Unauthenticated access** is selected.
9. Under **Encryption** ensure that only **Plaintext** is selected.
10. 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 the IAM role that can write to the destination bucket

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left pane, under **Access management**, choose **Roles**.
3. Choose **Create role**.
4. Under **Or select a service to view its use cases**, choose **S3**.
5. Scroll down and under **Select your use case**, again choose **S3**.
6. Choose **Next: Permissions**.
7. Choose **Create policy**. This opens a new tab in your browser where you will create the policy. Leave the original role-creation tab open because we'll get back to it later.

8. Choose the **JSON** tab, and then replace the text in the window with the following policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets"
      ],
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::<my-tutorial-destination-bucket>"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts",
        "s3:ListBucketMultipartUploads"
      ],
      "Resource": "*"
    }
  ]
}
```

9. Choose **Next: Tags**.
10. Choose **Next: Review**.
11. Enter `mkc-tutorial-policy` for the policy name, then scroll down and choose **Create policy**.
12. Back in the browser tab where you were creating the role, choose the refresh button.
13. Find the `mkc-tutorial-policy` and select it by choosing the button to its left.
14. Choose **Next: Tags**.
15. Choose **Next: Review**.
16. Enter `mkc-tutorial-role` for the role name, and delete the text in the description box.
17. 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

Step 2: Create custom plugin (p. 67)

Step 2: 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 Lenses Amazon S3 Sink Connector JAR.
2. Compress the downloaded JAR file to turn it into a ZIP file.
3. 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.
4. Open the Amazon MSK console at https://console.aws.amazon.com/msk/.
5. In the left pane expand **MSK Connect**, then choose **Custom plugins**.
6. Choose **Create custom plugin**.
7. Choose **Browse S3**.
8. In the list of buckets find the bucket where you uploaded the ZIP file, and choose that bucket.
9. 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**.
10. 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

Step 3: Create client machine and Apache Kafka topic (p. 68)

Step 3: 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 “Step 1: Set up required resources” (p. 64). 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

Step 4: Create connector (p. 69)

Step 4: Create connector

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.

   ```
   connector.class=io.lenses.streamreactor.connect.aws.s3.sink.S3SinkConnector
   key.converter.schemas.enable=false
   connect.s3.kcql=INSERT INTO <my-tutorial-destination-bucket>:tutorial SELECT * FROM
    mkc-tutorial-topic WITH_FLUSH_COUNT = 1
   ```

```
aws.region=us-east-1
tasks.max=2
topics=mkc-tutorial-topic
schema.enable=false
value.converter=org.apache.kafka.connect.storage.StringConverter
errors.log.enable=true
key.converter=org.apache.kafka.connect.storage.StringConverter
```

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

Step 5: Send data (p. 70)

Step 5: Send data

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=PLAINTEXT
   ```

2. Run the following command to create a console producer. Replace *`BootstrapBrokerString`* with the value that you obtained when you ran the previous command.

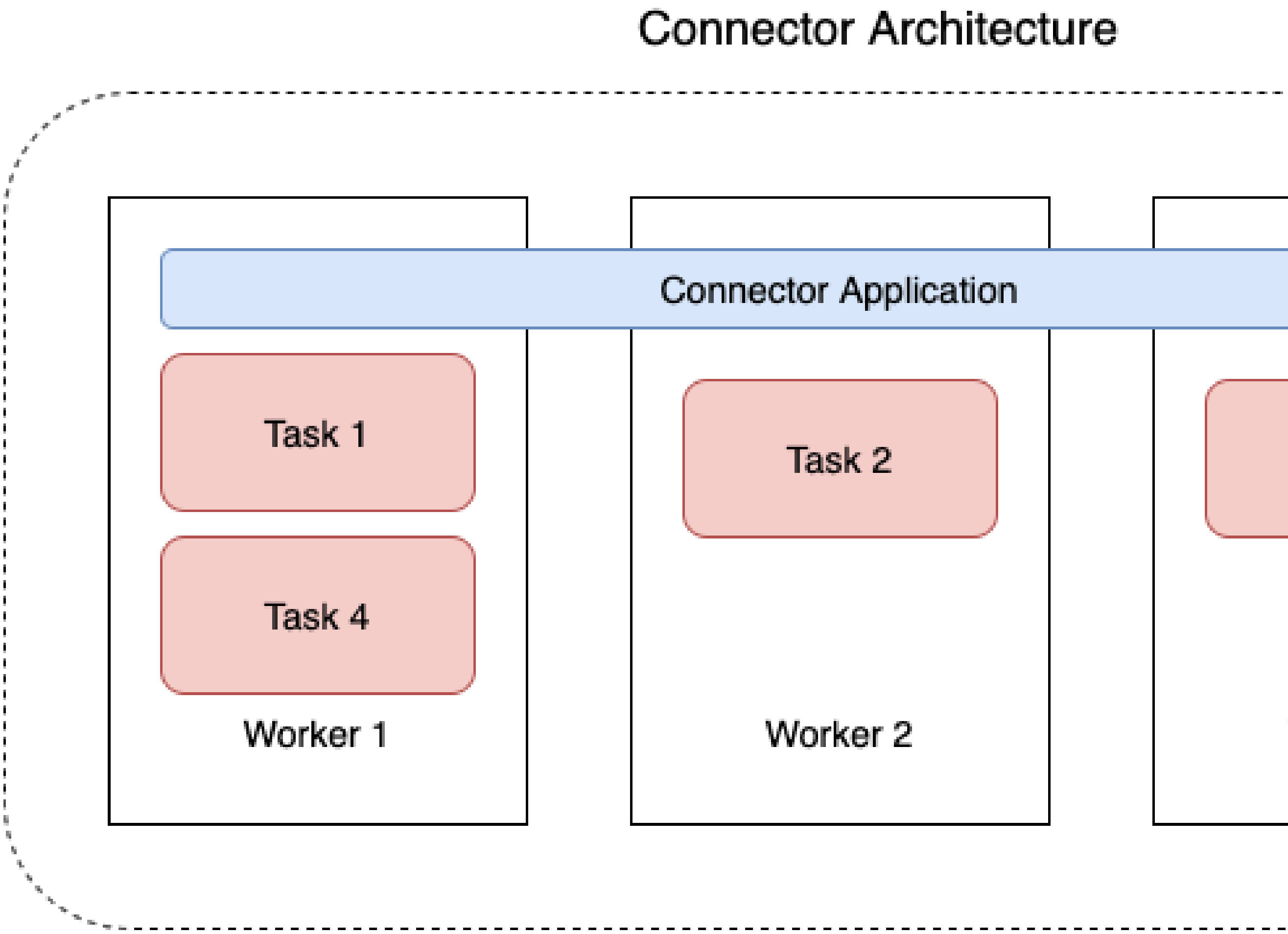
   ```
   <path-to-your-kafka-installation>/bin/kafka-console-producer.sh --broker-list BootstrapBrokerString --producer.config client.properties --topic mkc-tutorial-topic
   ```

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.

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 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. 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 “Workers” (p. 73). To learn about MSK Connect metrics, see the section called “Monitoring” (p. 90).

Creating a connector

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 “Plugins” (p. 73). 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=my-destination-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.partitioner.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 “Capacity” (p. 71).
9. Choose either the default worker configuration or a custom worker configuration. For information about creating custom worker configurations, see the section called “Workers” (p. 73).
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 “Service execution role” (p. 78).
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” (p. 89).

13. Choose **Create connector**.

To use the MSK Connect API to create a connector, see CreateConnector.

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 Guidein 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.

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 “Capacity” (p. 71).

Topics

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.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.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.

Creating a custom worker configuration

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.

Managing 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.

Using 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.

Specifying your own 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 Getting the bootstrap brokers for an Amazon MSK cluster (p. 21).

   ```
   <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 (p. 75) 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 Creating a custom worker configuration (p. 75).

   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=us-east-1
   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.SecretsManagerConfig
   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 Creating a connector (p. 72).

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.
- 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.
- 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.

Using a configuration provider to externalize secrets

To externalize sensitive configuration values with a service like AWS Secrets Manager, you can set up a configuration provider that implements the ConfigProvider class interface. A configuration provider 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 appearing as plaintext in a connector configuration.

You can develop your own configuration provider plugin or use a third-party plugin. For an example of setting up a third-party configuration provider that integrates with AWS Secrets Manager, see the instructions for configuring an Amazon Redshift sink connector with configuration provider (p. 93).

To develop your own configuration provider plugin, use the following guidelines:

- Implement the ConfigProvider interface, which is discovered using the Java ServiceLoader facility.
- Create a file named `META-INF/services/org.apache.kafka.common.config.provider.ConfigProvider` that contains the fully qualified name of your ConfigProvider implementation class.

Package this file into a JAR with your implementation classes. For example, see `org.apache.kafka.common.config.provider.ConfigProvider` in the open-source AWS Secrets Manager Config Provider plugin.

- Use the Kafka Connect enum constant `ConfigDef.Type.PASSWORD` to define sensitive configuration values. For more information, see Preventing secrets from appearing in connector logs (p. 90).

Considerations

Consider the following when you use a configuration provider with Amazon MSK Connect.

- 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 (p. 90).
- 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.

IAM roles and policies for MSK Connect

Topics

Service execution role

Note

Amazon MSK Connect does not support using the Service-linked role (p. 85) 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 are 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 “Cross-service confused deputy prevention” (p. 81).

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

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 on how to find your cluster's UUID and how to construct topic ARNs, see the section called “Resources” (p. 128).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:Connect",
                "kafka-cluster:DescribeCluster"
            ],
            "Resource": [
                "cluster-arn"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:ReadData",
                "kafka-cluster:DescribeTopic"
            ],
            "Resource": [
                "ARN of the topic that you want a sink connector to read from"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:WriteData",
                "kafka-cluster:DescribeTopic"
            ],
            "Resource": [
                "ARN of the topic that you want a source connector to write to"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:CreateTopic",
                "kafka-cluster:WriteData",
                "kafka-cluster:ReadData",
                "kafka-cluster:DescribeTopic"
            ],
            "Resource": [
                "arn:aws:kafka:region:account-id:topic/cluster-name/cluster-uuid/
__amazon_msk_connect_*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeGroup"
            ],
            "Resource": [
                "arn:aws:kafka:region:account-id:group/cluster-name/cluster-uuid/
__amazon_msk_connect_*",
                "arn:aws:kafka:region:account-id:group/cluster-name/cluster-uuid/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.

Examples of IAM policies 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": [
        {
            "Effect": "Allow",
            "Action": [
                "kafkaconnect:*",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DescribeSecurityGroups",
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/kafkaconnect.amazonaws.com/
AWSServiceRoleForKafkaConnect*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "kafkaconnect.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/kafkaconnect.amazonaws.com/
AWSServiceRoleForKafkaConnect*"
        },
        {
            "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"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutBucketPolicy",
                "s3:GetBucketPolicy"
            ],
            "Resource": "ARN of the Amazon S3 bucket to which you want MSK Connect to
 deliver logs"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "ARN of the service execution role"
        },
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "ARN of the Amazon S3 object that corresponds to the custom plugin
 that you want to use for creating connectors"
        },
        {
            "Effect": "Allow",
            "Action": "firehose:TagDeliveryStream",
            "Resource": "ARN of the Kinesis Data Firehose delivery stream to which you want
 MSK Connect to deliver logs"
        }
    ]
}
```

Cross-service confused deputy prevention

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

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the `ViewOnlyAccess` AWS managed policy provides read-only access to many AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions 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 AmazonMSKConnectReadOnlyAccess policy to remove the restrictions on listing operations.	October 13, 2021

Change	Description	Date
MSK Connect started tracking changes	MSK Connect started tracking changes for its AWS managed policies.	September 14, 2021

Using 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” (p. 83).

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.

Enabling 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.

Workaround for public subnets

If you configure your cluster to use public subnets, you can attach an Elastic IP address (EIP) to the elastic network interface (ENI) that Amazon MSK Connect creates in your VPC. We don't recommend this method, because ENI attachments are not permanent, and the service may create or remove ENIs during maintenance or scaling operations.

Setting 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.

Prerequisites

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.

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	• **Destination**: 0.0.0.0/0 • **Target**: Your NAT gateway ID. For example, *nat-12a345bc6789efg1h*.
A local route for internal traffic	• **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	• **Destination**: 0.0.0.0/0 • **Target**: Your internet gateway ID. For example, *igw-1a234bc5*.
A local route for internal traffic	• **Destination**: 10.0.0.0/16. This value may differ depending on your VPC's CIDR block. • **Target**: Local

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 referenced 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.

Topics

Configuring 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.

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.

Failure handling

This section describes possible connector creation failures associated with DNS resolution and suggested actions to resolve the issues.

Failure	Suggested action
Connector creation fails if a DNS resolution query fails, or if DNS servers are unreachable from the connector.	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. Check the DNS server configurations and ensure network connectivity to the DNS servers from the connector.
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.	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. 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.

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 Kinesis 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 Kinesis Data Firehose delivery stream in the *Kinesis Data 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.

Level	Description
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.

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
`BytesInPerSec`	The total number of bytes received by the connector.
`BytesOutPerSec`	The total number of bytes delivered by the connector.
`CpuUtilization`	The percentage of CPU consumption by system and user.
`ErroredTaskCount`	The number of tasks that have errored out.
`MemoryUtilization`	The percentage of memory consumption for the connector.
`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.
`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

This section includes examples to help you set up Amazon MSK Connect resources such as common third-party connectors and configuration providers.

Topics

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” (p. 78).

```
{
    "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.partitioner.DefaultPartitioner",
        "storage.class": "io.confluent.connect.s3.storage.S3Storage",
        "s3.bucket.name": "my-test-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"
   }
   ```

Amazon Redshift sink connector with configuration provider

This example shows how to use the Confluent Amazon Redshift Sink Connector plugin and the open-source AWS Secrets Manager Config Provider plugin to create a connector for MSK Connect. Using a configuration provider lets you externalize secrets such as database credentials. To learn more about configuration providers, see Using a configuration provider to externalize secrets (p. 77).

> **Note**
> You can test and evaluate the Confluent Amazon Redshift Sink Connector for free for 30 days. After the 30-day evaluation period, you must have an appropriate Confluent license. For more information, see Confluent Platform Licenses.

Before you begin

> **Note**
> Amazon MSK Connect does not currently support connections to AWS Secrets Manager using VPC endpoints for AWS PrivateLink.

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 Enabling internet access for Amazon MSK Connect (p. 86).

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
A route to enable internet access for MSK Connect	• **Destination**: 0.0.0.0/0 • **Target**: Your NAT gateway ID. For example, *nat-12a345bc6789efg1h*.
A local route for internal traffic	• **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	• **Destination**: 0.0.0.0/0

Property	Value
	• **Target**: Your internet gateway ID. For example, *igw-1a234bc5*.
A local route for internal traffic	• **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.

Creating an Amazon Redshift sink connector

1. **Prepare a custom plugin**

   a. Download and extract the Amazon Redshift Sink Connector from the Confluent site.

   b. Download and extract the AWS Secrets Manager Config Provider.

   c. Place the following archives into the same directory:

      - The `confluentinc-kafka-connect-aws-redshift-1.2.0` 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, `redshift-sink-custom-plugin.json`. Replace *`<example-custom-plugin-name>`* with the name that you want the plugin to have, *`<arn-of-your-s3-bucket>`* with the ARN of the S3 bucket where you uploaded the ZIP file, and *`<file-key-of-ZIP-object>`* with the file key of the ZIP that you uploaded to S3.

```
{
    "name": "<example-custom-plugin-name>",
    "contentType": "ZIP",
    "location": {
        "s3Location": {
            "bucketArn": "<arn-of-your-s3-bucket>",
            "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://<redshift-sink-
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 permissions and create a secret for your Amazon Redshift credentials**

   a. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.

   b. Create a new secret to store your Amazon Redshift user name and password. 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 Service execution role (p. 78). Replace *<arn:aws:secretsmanager:us-east-1:123456789000:secret:MyRedshiftSecret-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:MyRedshiftSecret-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.SecretsManagerCo
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. Use the following example JSON to define properties for your connector. Replace the *`<placeholders>`* with values that correspond to your scenario and resources.

      Note that the configuration uses variables like `${secretManager:MyRedshiftSecret-1234:user name}` instead of plaintext to specify Redshift credentials. Replace *`MyRedshiftSecret-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.

```
{
    "connectorName": "example-redshift-sink-connector",
    "kafkaConnectVersion": "2.7.1",
    "serviceExecutionRoleArn": "<arn-of-msk-connect-service-execution-role>",
    "plugins": [
        {
            "customPlugin": {
                "customPluginArn": "<arn-of-custom-plugin>",
                "revision": 1
            }
        }
    ],
    "capacity": {
        "provisionedCapacity": {
            "workerCount": 2,
            "mcuCount": 1
        }
    },
    "kafkaCluster": {
        "apacheKafkaCluster": {
            "bootstrapServers": "<my-cluster-bootstrap-servers-string>",
            "vpc": {
                "subnets": [
                    "<subnet-abcd1234>",
                    "<subnet-cdef0123>",
                    "<subnet-abcd0101>"
                ]
            }
        }
    },
    "kafkaClusterClientAuthentication": {"authenticationType": "NONE"},
    "kafkaClusterEncryptionInTransit": {"encryptionType": "PLAINTEXT"},
    "logDelivery": {
        "workerLogDelivery": {
            "cloudWatchLogs": {
                "logGroup": "<example-log-group-name>",
                "enabled": true
            }
        }
    },
    "connectorConfiguration": {
        "confluent.topic.bootstrap.servers": "<bootstrap-servers-string>",
        "confluent.topic.replication.factor": "1",
        "connector.class":
 "io.confluent.connect.aws.redshift.RedshiftSinkConnector",
        "tasks.max": "2",
        "topics": "<example-topic>",
        "aws.redshift.domain": "my-example-redshift-cluster.abcdefghijkl.us-east-1-
integ.redshift-dev.amazonaws.com",
        "aws.redshift.port": "5439",
        "aws.redshift.database": "dev",
        "aws.redshift.user": "${secretManager:<MyRedshiftSecret-1234>:<username>}",
        "aws.redshift.password":
 "${secretManager:<MyRedshiftSecret-1234>:<password>}",
        "pk.mode": "kafka",
        "auto.create": "true",
        "key.converter": "org.apache.kafka.connect.json.JsonConverter",
        "value.converter": "org.apache.kafka.connect.json.JsonConverter",
        "key.converter.schemas.enable": "true",
        "value.converter.schemas.enable": "true"
    },
    "workerConfiguration": {
        "workerConfigurationArn": "<arn-of-config-provider-worker-configuration>",
        "revision": 1
    }
}
```

b. Run the following command to create the connector.

```
aws kafkaconnect create-connector --cli-input-json file://redshift-sink-
connector.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-
redshift-sink-connector/abc12345-abcd-4444-a8b9-123456f513ed-2",
    "ConnectorState": "CREATING",
    "ConnectorName": "example-redshift-sink-connector"
}
```

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 Using a configuration provider to externalize secrets (p. 77).

> **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 Connector capacity (p. 71).

Before you begin

> **Note**
> Amazon MSK Connect does not currently support connections to AWS Secrets Manager using VPC endpoints for AWS PrivateLink.

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 Enabling internet access for Amazon MSK Connect (p. 86).

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
A route to enable internet access for MSK Connect	• **Destination**: 0.0.0.0/0 • **Target**: Your NAT gateway ID. For example, *nat-12a345bc6789efg1h*.
A local route for internal traffic	• **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	• **Destination**: 0.0.0.0/0 • **Target**: Your internet gateway ID. For example, *igw-1a234bc5*.
A local route for internal traffic	• **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.

Creating a Debezium source connector

1. **Create a custom plugin**

   a. Download the MySQL connector plugin for the latest stable release from the Debezium site.

   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, *`<arn-of-your-s3-bucket>`* with the ARN of the 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": "<arn-of-your-s3-bucket>",
            "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 user name and password. 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 Service execution role (p. 78). 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.SecretsManagerCo
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 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" (p. 78).

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.

```
{
    "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"
}
```

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.

Cluster states

The following table shows the possible states of a cluster and describes what they mean. It also describes what actions you can and cannot perform when a 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 cluster. The description of a cluster includes its state.

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 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 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 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 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.
MAINTENANCE	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.
REBOOTING_BROKER	Amazon MSK is rebooting a broker. 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.
UPDATING	A user-initiated Amazon MSK API or AWS CLI operation is updating the cluster. You can still use the cluster to produce and consume data.

Cluster state	Meaning and possible actions
	However, you cannot perform any additional Amazon MSK API or AWS CLI update operations on the cluster until it returns to the ACTIVE state.

Security in Amazon Managed Streaming for Apache Kafka

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

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. This content includes 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 user accounts with 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 recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- 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 personal data that is stored in Amazon S3.
- If you require FIPS 140-2 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) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form 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 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

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. It sets the state of the cluster to `MAINTENANCE` when it starts the certificate-update operation. It sets it back to `ACTIVE` when the update is done. While a cluster is in the `MAINTENANCE` state, you can continue to produce and consume data, but you can't perform any update operations on it.

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*.

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.

How do I get started with 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`.

To 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.

```
aws kafka create-cluster --cluster-name "ExampleClusterName" --broker-node-group-info
 file://brokernodegroupinfo.json --encryption-info file://encryptioninfo.json --kafka-
version "2.2.1" --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"
}
```

To test TLS encryption

1. Create a client machine following the guidance in the section called “Step 2: Create a client machine” (p. 5).
2. Install Apache Kafka on the client machine.
3. Run the following command on a machine that has the AWS CLI installed, replacing *clusterARN* with the ARN of your cluster (a cluster created with ClientBroker set to TLS like the example in the previous procedure).

   ```
   aws kafka describe-cluster --cluster-arn clusterARN
   ```

   In the result, look for the value of ZookeeperConnectString and save it because you need it in the next step.
4. Run the following command on your client machine to create a topic. Replace *ZookeeperConnectString* with the value you obtained for ZookeeperConnectString in the previous step.

   ```
   <path-to-your-kafka-installation>/bin/kafka-topics.sh --create --
   zookeeper ZookeeperConnectString --replication-factor 3 --partitions 1 --topic
    TLSTestTopic
   ```
5. 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
   ```
6. 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
   ```
7. 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"
   }
   ```
8. Run the following command to create a console producer on your client machine. Replace *BootstrapBrokerStringTls* with the value you obtained in the previous step. Leave this producer command running.

   ```
   <path-to-your-kafka-installation>/bin/kafka-console-producer.sh --broker-
   list BootstrapBrokerStringTls --producer.config client.properties --topic TLSTestTopic
   ```
9. 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
```

10. 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.

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.

This page describes how you can use IAM to control who can perform Amazon MSK operations on your cluster. For information on how to control who can perform Apache Kafka operations on your cluster, see the section called “Authentication and authorization for Apache Kafka APIs” (p. 122).

Topics

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

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

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

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

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

To view examples of Amazon MSK identity-based policies, see Amazon MSK identity-based policy examples (p. 113).

Amazon MSK resource-based policies

Amazon MSK does not support resource-based policies.

AWS managed policies

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 more information about tagging Amazon MSK resources, see the section called "Tagging a cluster" (p. 37).

To view an example identity-based policy for limiting access to a cluster based on the tags on that cluster, see Accessing Amazon MSK clusters based on tags (p. 115).

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 IAM 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" (p. 116).

Amazon MSK identity-based policy examples

By default, IAM users and roles don't have permission to execute Amazon MSK API actions. An IAM 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

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 IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or root users in your account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

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*.

Using 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

Amazon MSK uses the service-linked role named **AWSServiceRoleForKafka** – Allows Amazon MSK to access AWS resources on your behalf.

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 the specified resources:

- Action: `ec2:CreateNetworkInterface` on *
- Action: `ec2:DescribeNetworkInterfaces` on *
- Action: `ec2:CreateNetworkInterfacePermission` on *
- Action: `ec2:AttachNetworkInterface` on *
- Action: `ec2:DeleteNetworkInterface` on *

- Action: `ec2:DetachNetworkInterface` on *
- Action: `acm-pca:GetCertificateAuthorityCertificate` on *
- Action: `secretsmanager:ListSecrets` on *
- Action: `secretsmanager:GetResourcePolicy` on secrets with the prefix `AmazonMSK_` that you create for Amazon MSK
- Action: `secretsmanager:PutResourcePolicy` on secrets with the prefix `AmazonMSK_` that you create for Amazon MSK
- Action: `secretsmanager:DeleteResourcePolicy` on secrets with the prefix `AmazonMSK_` that you create for Amazon MSK
- Action: `secretsmanager:DescribeSecret` on secrets with the prefix `AmazonMSK_` that you create for Amazon MSK

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 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.

Editing 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

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the `ViewOnlyAccess` AWS managed policy provides read-only access to many AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions 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.
- Some of the 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.
- The 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.
- The CloudWatch Logs, Amazon S3, and Amazon Kinesis 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.

```
    {
        "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: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 exiting 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:

- The Amazon MSK permissions allow you to list Amazon MSK resources, describe them, and get information about them.
- The Amazon EC2 permissions are used to describe the Amazon VPC, subnets, security groups, and ENIs that are associated with a cluster.
- The 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 on your behalf. For more information, see the section called “Service-linked roles” (p. 116).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:AttachNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DetachNetworkInterface",
                "acm-pca:GetCertificateAuthorityCertificate",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:DeleteResourcePolicy",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "secretsmanager:SecretId":
 "arn:*:secretsmanager:*:*:secret:AmazonMSK_*"
                }
            }
        }
    ]
}
```

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
AmazonMSKFullAccess (p. 118) – 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 (p. 118) – Update to an existing policy	Amazon MSK added a new permission to allow it to describe Amazon EC2 route tables.	November 19, 2021
Amazon MSK started tracking changes	Amazon MSK started tracking changes for its AWS managed policies.	November 19, 2021

Troubleshooting 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

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the mateojackson 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" (p. 110).

Topics

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.

Amazon MSK logs access events so you can audit them. For more information, see the section called "CloudTrail events" (p. 143).

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 the section called "Controlling access to Apache ZooKeeper" (p. 139).
>
> **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 stored in Apache ZooKeeper have no effect on authorization for IAM roles. You must use IAM policies to control access for IAM roles.

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 the rest of this section.

- the section called "Create a cluster that uses IAM access control" (p. 123)
- the section called "Configure clients for IAM access control" (p. 123)
- the section called "Create authorization policies" (p. 124)
- the section called "Get the bootstrap brokers for IAM access control" (p. 125)

Create a 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 cluster that uses IAM access control. For information about how to turn on IAM access control for an existing cluster, see the section called "Updating security" (p. 33).

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, configure them as described in the following steps.

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

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. the section called “Configure clients for IAM access control” (p. 123) 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 the section called “Semantics of actions and resources” (p. 125).

> **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-1:0123456789012:topic/MyTestCluster/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeGroup"
            ],
            "Resource": [
                "arn:aws:kafka:us-east-1:0123456789012: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 the section called “Common use cases” (p. 129).

Get the bootstrap brokers for IAM access control

See the section called “Getting the bootstrap brokers” (p. 21).

Semantics of 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 the section called “Create authorization policies” (p. 124).

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**
`kafka-cluster:Connect`	Grants permission to connect and authenticate to the cluster.	None	cluster	Yes
`kafka-cluster:DescribeCluster`	Grants permission to describe various aspects of the cluster, equivalent to Apache Kafka's DESCRIBE CLUSTER ACL.	`kafka-cluster:Connect`	cluster	Yes
`kafka-cluster:AlterCluster`	Grants permission to alter various aspects of the cluster, equivalent to Apache Kafka's ALTER CLUSTER ACL.	`kafka-cluster:Connect` `kafka-cluster:DescribeCluster`	cluster	No
`kafka-cluster:DescribeClusterDynamicConfiguration`	Grants permission to describe	`kafka-cluster:Connect`	cluster	No

Action	Description	Required actions	Required resources	Applicable to serverless clusters
	the dynamic configuration of a cluster, equivalent to Apache Kafka's DESCRIBE_CONFIGS CLUSTER ACL.			
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
kafka-cluster:WriteDataIdempotently	Grants permission to write data idempotently on a cluster, equivalent to Apache Kafka's IDEMPOTENT_WRITE CLUSTER ACL.	kafka-cluster:Connect kafka-cluster:WriteData	cluster	Yes
kafka-cluster:CreateTopic	Grants permission to create topics on a cluster, equivalent to Apache Kafka's CREATE CLUSTER/TOPIC ACL.	kafka-cluster:Connect	topic	Yes
kafka-cluster:DescribeTopic	Grants permission to describe topics on a cluster, equivalent to Apache Kafka's DESCRIBE TOPIC ACL.	kafka-cluster:Connect	topic	Yes
kafka-cluster:AlterTopic	Grants permission to alter topics on a cluster, equivalent to Apache Kafka's ALTER TOPIC ACL.	kafka-cluster:Connect kafka-cluster:DescribeTopic	topic	Yes
kafka-cluster:DeleteTopic	Grants permission to delete topics on a cluster, equivalent to Apache Kafka's DELETE TOPIC ACL.	kafka-cluster:Connect kafka-cluster:DescribeTopic	topic	Yes

Action	Description	Required actions	Required resources	Applicable to serverless clusters
`kafka-cluster:DescribeTopicDynamicConfiguration`	Grants permission to describe the dynamic configuration of topics on a cluster, equivalent to Apache Kafka's DESCRIBE_CONFIGS TOPIC ACL.	`kafka-cluster:Connect`	topic	Yes
`kafka-cluster:AlterTopicDynamicConfiguration`	Grants permission to alter the dynamic configuration of topics on a cluster, equivalent to Apache Kafka's ALTER_CONFIGS TOPIC ACL.	`kafka-cluster:Connect` `kafka-cluster:DescribeTopicDynamicConfiguration`	topic	Yes

`kafka-cluster:ReadData`	Grants permission to read data from topics on a cluster, equivalent to Apache Kafka's READ TOPIC ACL.	`kafka-cluster:Connect`
		`kafka-cluster:DescribeTopic`
		`kafka-cluster:AlterGroup`
			topic	Yes

`kafka-cluster:WriteData`	Grants permission to write data to topics on a cluster, equivalent to Apache Kafka's WRITE TOPIC ACL	`kafka-cluster:Connect` `kafka-cluster:DescribeTopic`	topic	Yes
`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

Action	Description	Required actions	Required resources	Applicable to serverless clusters
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
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.

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	arn:aws:kafka:*region*:*account-id*:cluster/*cluster-name*/*cluster-uuid*
Topic	arn:aws:kafka:*region*:*account-id*:topic/*cluster-name*/*cluster-uuid*/*topic-name*

Resource	ARN format
Group	arn:aws:kafka:*region*:*account-id*:group/*cluster-name*/*cluster-uuid*/*group-name*
Transactional ID	arn:aws:kafka:*region*:*account-id*:transactional-id/*cluster-name*/*cluster-uuid*/*transactional-id*

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

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 the section called “Semantics of actions and resources” (p. 125).

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	`kafka-cluster:*`
Create a topic	`kafka-cluster:Connect` `kafka-cluster:CreateTopic`

Produce data	`kafka-cluster:Connect`
	`kafka-cluster:DescribeTopic`
	`kafka-cluster:WriteData`

Consume data	`kafka-cluster:Connect`
	`kafka-cluster:DescribeTopic`
	`kafka-cluster:DescribeGroup`

Use case	Required actions
	kafka-cluster:AlterGroup kafka-cluster:ReadData
Produce data idempotently	kafka-cluster:Connect kafka-cluster:DescribeTopic kafka-cluster:WriteData kafka-cluster:WriteDataIdempotently
Produce data transactionally	kafka-cluster:Connect kafka-cluster:DescribeTopic kafka-cluster:WriteData kafka-cluster:DescribeTransactionalId kafka-cluster:AlterTransactionalId
Describe the configuration of a cluster	kafka-cluster:Connect kafka-cluster:DescribeClusterDynamicConfiguration
Update the configuration of a cluster	kafka-cluster:Connect kafka-cluster:DescribeClusterDynamicConfiguration kafka-cluster:AlterClusterDynamicConfiguration
Describe the configuration of a topic	kafka-cluster:Connect kafka-cluster:DescribeTopicDynamicConfiguration
Update the configuration of a topic	kafka-cluster:Connect kafka-cluster:DescribeTopicDynamicConfiguration kafka-cluster:AlterTopicDynamicConfiguration
Alter a topic	kafka-cluster:Connect kafka-cluster:DescribeTopic kafka-cluster:AlterTopic

Mutual TLS authentication

You can enable client authentication with TLS for connections from your applications to your Amazon MSK brokers and ZooKeeper nodes. 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” (p. 137).

This topic contains the following sections:

- To create a cluster that supports client authentication (p. 131)
- To set up a client to use authentication (p. 132)
- To produce and consume messages using authentication (p. 133)

To create a 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 “Creating a cluster using the AWS CLI” (p. 12).
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 “Encryption” (p. 107).
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 "2.2.1" --number-of-broker-
nodes 3
```

To set up a client to use authentication

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 "Step 2: Create a client machine" (p. 5) 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 "Step 3: Create a topic" (p. 6).
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
   ```
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`.

    ```
    security.protocol=SSL
    ssl.truststore.location=/tmp/kafka_2.12-2.2.1/kafka.client.truststore.jks
    ssl.keystore.location=/tmp/kafka_2.12-2.2.1/kafka.client.keystore.jks
    ssl.keystore.password=Your-Store-Pass
    ssl.key.password=Your-Key-Pass
    ```

To produce and consume messages using authentication

1. Run the following command to create a topic.

   ```
   <path-to-your-kafka-installation>/bin/kafka-topics.sh --create --zookeeper ZooKeeper-Connection-String --replication-factor 3 --partitions 1 --topic ExampleTopic
   ```

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 --broker-list 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.

Username and password authentication with AWS Secrets Manager

You can control access to your Amazon MSK clusters using usernames and passwords 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.

This topic contains the following sections:

How it works

Username and password authentication for Amazon MSK uses SASL/SCRAM (Simple Authentication and Security Layer/ Salted Challenge Response Mechanism) authentication. To set up username and password authentication for a cluster, you create a Secret resource in AWS Secrets Manager, and associate user names and passwords with that secret.

SASL/SCRAM is defined in RFC 5802. SCRAM uses secured hashing algorithms, and does not transmit plaintext passwords 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.

Setting 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 user and password 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 broker” (p. 188).
- 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.

  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 a username and password

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. Retrieve your cluster details with the following command. Replace *`ClusterArn`* with the Amazon Resource Name (ARN) of your cluster:

   ```
   aws kafka describe-cluster --cluster-arn "ClusterArn"
   ```

   From the JSON result of the command, save the value associated with the string named `ZookeeperConnectString`.
2. To create an example topic, run the following command on your client machine. Replace *`ZookeeperConnectString`* with the string you recorded in the previous step.

   ```
   <path-to-your-kafka-installation>/bin/kafka-topics.sh --create --
   zookeeper ZookeeperConnectString --replication-factor 3 --partitions 1 --
   topic ExampleTopicName
   ```
3. 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";
};
```

4. 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
   ```

5. Create a file named `kafka.client.truststore.jks` in a `./tmp` directory.
6. 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/jre/lib/security/cacerts /tmp/kafka.client.truststore.jks
   ```

7. 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
   ssl.truststore.location=<path-to-keystore-file>/kafka.client.truststore.jks
   ```

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 username and password 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 (p. 137).

We recommend that you restrict access to your zookeeper nodes to prevent users from modifying ACLs. For more information, see Controlling access to Apache ZooKeeper (p. 139).

Limitations

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 that uses Apache ZooKeeper to store all ACLs. Amazon MSK enables this authorizer in the `server.properties` file on the brokers. For Apache Kafka version 2.4.1, the authorizer is AclAuthorizer. For earlier versions of Apache Kafka, it is SimpleAclAuthorizer.

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.

In addition to the client, you also need to grant all your brokers access to your topics so that the brokers can replicate messages from the primary partition. If the brokers don't have access to a topic, replication for the topic fails.

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 *ZooKeeper-Connection-String* with your Apache ZooKeeper connection string. For information on how to get this string, see the section called “Getting the Apache ZooKeeper connection string” (p. 19).

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 “Getting the bootstrap brokers” (p. 21).

```
<path-to-your-kafka-installation>/bin/kafka-acls.sh --authorizer-properties
 zookeeper.connect=ZooKeeper-Connection-String --add --allow-principal
 "User:CN=Distinguished-Name" --operation Read --group=* --topic Topic-Name
```

2. To grant 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 --authorizer-properties
 zookeeper.connect=ZooKeeper-Connection-String --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 --authorizer-properties
 zookeeper.connect=ZooKeeper-Connection-String --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.

Controlling access to Apache ZooKeeper

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.

This topic contains the following sections:

To place your Apache ZooKeeper nodes in a separate security group

1. Get the Apache ZooKeeper connection string for your cluster. To learn how, see the section called “Getting the Apache ZooKeeper connection string” (p. 19). 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 How do I get started with encryption? (p. 108).
- 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.

Logging

You can deliver Apache Kafka broker logs to one or more of the following destination types: Amazon CloudWatch Logs, Amazon S3, Amazon Kinesis Data Firehose. You can also log Amazon MSK API calls with AWS CloudTrail.

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 Kinesis Data Firehose delivery stream. Through Kinesis Data 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 Kinesis Data Firehose

Note
Amazon MSK does not support delivering broker logs to Kinesis Data Firehose in the Asia Pacific (Osaka) Region.

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 (p. 118) 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 Console 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": "*"
}
```

Configuring 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.

Configuring 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": "ExampleBucketName",
      "Prefix": "ExamplePrefix",
      "Enabled": true
    },
    "Firehose": {
      "DeliveryStream": "ExampleDeliveryStreamName",
      "Enabled": true
    },
    "CloudWatchLogs": {
      "Enabled": true,
      "LogGroup": "ExampleLogGroupName"
    }
  }
}
```

Configuring 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 Kinesis Data 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.

Logging API calls with AWS CloudTrail

> **Note**
> AWS CloudTrail logs are available for Amazon MSK only when you use IAM access control (p. 122).

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 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"
}
```

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.

Connecting to an Amazon MSK cluster

By default, clients can access an MSK cluster only if they're in the same VPC as the cluster. To connect to your MSK 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 *Getting started* (p. 5).

To connect to your MSK cluster from a client that's outside the cluster's VPC, see the following topics:

Topics

Public access

Amazon MSK gives you the option to turn on public access to the brokers of MSK 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.

To turn on public access to a cluster, first ensure that the cluster meets all of the following conditions:

- The subnets that are associated with the cluster must be public. 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 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 "Updating security" (p. 33).
- 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 "Updating security" (p. 33).
- If you are using the SASL/SCRAM or mTLS access-control methods, you must set Apache Kafka ACLs for your cluster. 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` to false for the cluster. For information about how to update the configuration of a cluster, see the section called "Configuration operations" (p. 51). 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" (p. 122). For information about Apache Kafka ACLs, see the section called "Apache Kafka ACLs" (p. 137).

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 "Getting the bootstrap brokers" (p. 21).

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" (p. 151). For instructions on how to change a cluster's security group, see the section called "Changing security groups" (p. 138).

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" (p. 185).

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 `KTVPDKIKX0DER`.

   ```
   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 nodes. For information about how to control access to the Apache ZooKeeper nodes of your MSK cluster from within AWS, see the section called "Controlling access to Apache ZooKeeper" (p. 139).

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.

EC2-Classic

Use the following procedure to connect to your cluster from an EC2-Classic instance.

1. Follow the guidance described in ClassicLink to connect your EC2-Classic instance to your cluster's VPC.
2. Find and copy the private IP associated with your EC2-Classic instance.
3. Using the AWS CLI, run the following command, replacing *`ClusterArn`* with the Amazon Resource Name (ARN) for your MSK cluster.

```
aws kafka describe-cluster --cluster-arn "ClusterArn"
```

4. In the output of the `describe-cluster` command, look for `SecurityGroups` and save the ID of the security group for your MSK cluster.
5. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
6. In the left pane, choose **Security Groups**.
7. Choose the security group whose ID you saved after you ran the `describe-cluster` command. Select the box at the beginning of the row corresponding to this security group.
8. In the lower half of the page, choose **Inbound Rules**.
9. Choose **Edit rules**, then choose **Add Rule**.
10. For the **Type** field, choose **All traffic** in the drop-down list.
11. Leave the **Source** set to **Custom** and enter the private IP of your EC2-Classic instance, followed immediately by **/32** with no intervening spaces.

12. Choose **Save rules**.

Port information

The following list provides the numbers of the ports that Amazon MSK uses to communicate with client machines.

- To communicate with brokers in plaintext, use port 9092.
- To communicate with brokers by using TLS encryption, use port 9094 for access from within AWS and port 9194 for public access.
- To communicate with brokers by using SASL/SCRAM, use port is 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" (p. 122), use port 9098 for access from within AWS and port 9198 for public access.
- Apache ZooKeeper nodes use port 2181 by default. To communicate with Apache ZooKeeper by using TLS encryption, use port 2182.

Migrating clusters using Apache Kafka's MirrorMaker

You can mirror or migrate your cluster using MirrorMaker, which is part of Apache Kafka. For example, you can use it to migrate your Apache Kafka cluster to Amazon MSK or to migrate from one MSK cluster to another. 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.

Migrating 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`.

Migrating from one Amazon MSK cluster to another

You can use Apache MirrorMaker to migrate an MSK cluster to another cluster. For example, you can migrate from one version of Apache Kafka to another. For an example of how to use AWS CloudFormation to do this, see AWS::MSK::Cluster Examples (search for the example titled `Create Two MSK Clusters To Use With Apache MirrorMaker`.

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.Max_Value
  - acks=all
  - max.block.ms = Long.Max_Value
- 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.

MirrorMaker 2.* advantages

- 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.

Monitoring an Amazon MSK cluster

Amazon Managed Streaming for Apache Kafka 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 cluster with Prometheus, see the section called “Open monitoring with Prometheus” (p. 167).

Topics

Amazon MSK metrics for monitoring with CloudWatch

Amazon MSK integrates with Amazon CloudWatch so that you can collect, view, and analyze CloudWatch metrics for your Amazon MSK cluster. The metrics that you configure for your MSK cluster are automatically collected and pushed to CloudWatch. You can set the monitoring level for an MSK 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.

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.

Metrics available at the DEFAULT monitoring level

Name	When visible	Dimensio	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	The remaining balance of input-output burst credits for EBS volumes in the cluster. Use it to investigate latency or decreased throughput. BurstBalance is not reported for EBS volumes when the baseline performance of a volume is higher than the maximum burst performance. For

Name	When visible	Dimensio	Description
			more information, see I/O Credits and burst performance.
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.
CPUCreditBalance	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	This metric can help you monitor CPU credit balance on the brokers. If your CPU usage is sustained above the baseline level of 20% utilization, you can run out of the CPU credit balance, which can have a negative impact on cluster 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.

Name	When visible	Dimensio	Description
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.
EstimatedMaxTimeLag	After consumer group consumes from a topic.	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.
KafkaDataLogsDiskUsed (Cluster Name dimension)	After the cluster gets to the ACTIVE state.	Cluster Name	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.
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.

Name	When visible	Dimensio	Description
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.
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.
OfflinePartitionsCount	After the cluster gets to the ACTIVE state.	Cluster Name	Total number of partitions that are offline in the cluster.
PartitionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The total number of topic partitions per broker, including replicas.

Name	When visible	Dimensio	Description
ProduceTotalTimeMsMean	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.
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.	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.
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.
ZooKeeperRequestLatencyMsMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean latency in milliseconds for Apache ZooKeeper requests from broker.

Name	When visible	Dimensio	Description
ZooKeeperSessionState	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	Connection status of broker's ZooKeeper session which may be one of the following: NOT_CONNECTED: '0.0', ASSOCIATING: '0.1', CONNECTING: '0.5', CONNECTEDREADONLY: '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.

Additional metrics that are available starting at the PER_BROKER monitoring level

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 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.	This metric can help you monitor CPU credit usage on the instances. If your CPU usage is sustained above the baseline level of 20%, you can run out of the CPU credit balance, which can have a negative impact on cluster performance. You can monitor and alarm on this metric to take corrective actions.

Name	When visible	Description
`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.
`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.
`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.

Name	When visible	Description
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.
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.
ProduceTotalTimeMsMean	After the cluster gets to the ACTIVE state.	The mean produce time in milliseconds.
RemoteBytesInPerSec	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.
RemoteBytesOutPerSec	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.
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.
RemoteReadErrorPerSec	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.
RemoteReadRequestsPerSec	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.
RemoteWriteErrorPerSec	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.
`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.
TotalTierBytesLag	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.
`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.

Additional metrics that are available starting at the PER_TOPIC_PER_BROKER monitoring level

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.
RemoteBytesInPerSec	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.
RemoteBytesOutPerSec	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.
RemoteReadErrorPerSec	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.
RemoteReadRequestsPerSec	After you create a topic and the topic is producing/ consuming.	The number of read requests that the specifies 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.
RemoteWriteErrorPerSec	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.

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.

Additional metrics that are available starting at the PER_TOPIC_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.

Viewing 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.

Consumer-lag monitoring

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 "Amazon MSK metrics for monitoring with CloudWatch" (p. 155).

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

> **Note**
> To turn on consumer-lag monitoring for a cluster that was created before November 23, 2020, ensure that the cluster is running Apache Kafka 2.2.1 or a later version, then create a support case.

Open monitoring with Prometheus

You can monitor your MSK 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.

Creating an Amazon MSK cluster with open monitoring enabled

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.

Enabling open monitoring for an existing Amazon MSK cluster

To enable open monitoring, make sure that the 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 the 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.

Setting up a Prometheus host on an Amazon EC2 instance

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.

```
[
  {
    "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"
    ]
  }
]
```

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.

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.

Storing 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 &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.

Using 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 "Open monitoring with Prometheus" (p. 167).
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 Apache ZooKeeper connection 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
zookeeper.connect=z-1.test-cluster.2skv42.c1.kafka.us-east-1.amazonaws.com:2181,z-2.test-cluster.2skv42.c1.kafka.us-east-1.amazonaws.com:2181,z-3.test-cluster.2skv42.c1.kafka.us-east-1.amazonaws.com:2181

# 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.PrometheusMetricS

# 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
```

2. Edit the `config/capacityCores.json` file to specify the right disk size and CPU cores and network in/out limits. You can use the DescribeCluster API operation (or its CLI equivalent) to obtain the disk size. For CPU cores and network in/out limits, see Amazon EC2 Instance Types.

```
{
  "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."
    }
  ]
}
```

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-kafka-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.

Amazon MSK quota

Amazon MSK quota

- Up to 90 brokers per account and 30 brokers per cluster. To request higher quota, create a support case.
- A minimum of 1 GiB of storage per broker.
- A maximum of 16384 GiB of storage per broker.
- A cluster that uses the section called "IAM access control" (p. 122) can have up to 3000 TCP connections per broker at any given time. To increase this limit, you can adjust the `listener.name.client_iam.max.connections` or the `listener.name.client_iam_public.max.connections` configuration property using the Kafka AlterConfig API or the `kafka-configs.sh` tool. It's important to note that increasing either property to a high value can result in unavailability.
- Limits on TCP connections. A cluster that uses the section called "IAM access control" (p. 122) can accept new connections at a rate of up to 20 TCP connections per broker per second for all broker types, except for the type kafka.t3.small. Brokers of type kafka.t3.small are limited to 4 TCP connections per broker per second. If you created your cluster after May 25, 2022, it also supports connection rate bursts. If you want an older cluster to support connection rate bursts, you can create a support case.

  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, see reconnect.backoff.ms in the Apache Kafka documentation.
- Up to 100 configurations per account. To request higher quota, create a support case.
- A maximum of 50 revisions per configuration.
- 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 the section called " Right-size your cluster: Number of partitions per broker" (p. 188).

MSK Serverless quota

Dimension	Quota
Maximum ingress throughput	200 MBps
Maximum egress throughput	400 MBps
Maximum retention duration	24 hours. To request a quota adjustment, create a support case.
Maximum number of client connections	1000
Maximum connection attempts	100 per second
Maximum message size	8 MB
Maximum request size	100 MB

Dimension	Quota
Maximum request rate	15,000 per second
Maximum fetch bytes per request	55 MB
Maximum number of consumer groups	500
Maximum number of partitions	120. To request a quota adjustment, create a support case.
Maximum rate of partition creation and deletion	120 in 5 minutes
Maximum ingress throughput per partition	5 MBps
Maximum egress throughput per partition	10 MBps
Maximum partition size	250 GB
Maximum number of client VPCs per serverless cluster	5
Maximum number of serverless clusters per account	3

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, create a support case.

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” (p. 122), a resource is an entity to which you can allow or deny access, as defined in the the section called “Resources” (p. 128) 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.

Topics

Supported Apache Kafka versions

Amazon Managed Streaming for Apache Kafka (Amazon MSK) supports the following Apache Kafka and Amazon MSK versions.

Topics

Apache Kafka version 3.3.1

For information about Apache Kafka version 3.3.1, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 3.2.0

For information about Apache Kafka version 3.2.0, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 3.1.1

For information about Apache Kafka version 3.1.1, see its 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 (p. 13).

Apache Kafka version 2.8.1

For information about Apache Kafka version 2.8.1, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 2.8.0

For information about Apache Kafka version 2.8.0, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 2.7.2

For information about Apache Kafka version 2.7.2, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 2.7.1

For information about Apache Kafka version 2.7.1, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 2.6.3

For information about Apache Kafka version 2.6.3, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 2.6.2 [recommended]

For information about Apache Kafka version 2.6.2, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 2.7.0

For information about Apache Kafka version 2.7.0, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 2.6.1

For information about Apache Kafka version 2.6.1, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 2.6.0

For information about Apache Kafka version 2.6.0, see its release notes on the Apache Kafka downloads site.

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 (p. 140).

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 Updating the Apache Kafka version (p. 179).

To work around this issue without upgrading the cluster to version 2.4.1.1, see the Consumer group stuck in `PreparingRebalance` state (p. 182) section of the Troubleshooting your Amazon MSK cluster (p. 182) 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 (p. 178) 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” (p. 164).

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 *Configuration* (p. 39).

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.

Apache Kafka version 2.3.1

For information about Apache Kafka version 2.3.1, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 2.2.1

For information about Apache Kafka version 2.2.1, see its release notes on the Apache Kafka downloads site.

Apache Kafka version 1.1.1 (for existing clusters only)

You can no longer create a new MSK cluster with Apache Kafka version 1.1.1. You can continue to use existing clusters that are configured with Apache Kafka version 1.1.1. For information about Apache Kafka version 1.1.1, see its release notes on the Apache Kafka downloads site.

Updating the Apache Kafka version

You can update an existing MSK cluster to a newer version of Apache Kafka. You can't update it to an older version. When you update 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. Amazon MSK only updates the server software. It doesn't update your clients.

For information about how to make a cluster highly available during an update, see the section called “Build highly available clusters” (p. 189).

> **Important**
> You can't update the Apache Kafka version for an MSK cluster that exceeds the limits described in the section called “ Right-size your cluster: Number of partitions per broker” (p. 188).

Updating the Apache Kafka version using the AWS Management Console

1. Open the Amazon MSK console at https://console.aws.amazon.com/msk/.

2. Choose the MSK cluster on which you want to update the Apache Kafka version.
3. On the **Properties** tab choose **Upgrade** in the **Apache Kafka version** section.

Updating 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 “Listing clusters” (p. 22).

```
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 update 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 “Listing clusters” (p. 22).

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 KTVPDKIKX0DER.

```
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.

Updating 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 update the cluster.
2. Invoke the UpdateClusterKafkaVersion operation to update the cluster to one of the compatible Apache Kafka versions.

Troubleshooting 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.

Topics

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 (p. 178), 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 Updating the Apache Kafka version (p. 179).

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 (p. 182) , or to Identify and reboot (p. 183) 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 “Step 2: Create a client machine” (p. 5).

- 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 *Migration* (p. 152).

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 *How it works* (p. 10).

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” (p. 184).

Disk space is running low

See the following best practices for managing disk space: the section called “Monitor disk space” (p. 190) and the section called “Adjust data retention parameters” (p. 190).

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 *Best practices* (p. 188).
- 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” (p. 151). 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 "Getting the bootstrap brokers using the AWS Management Console" (p. 21).

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 "Getting the bootstrap brokers" (p. 21) 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
   ```
3. Repeat the previous command for all the bootstrap brokers.
4. Use any of the methods described in the section called "Getting the Apache ZooKeeper connection string" (p. 19) to get the addresses of the cluster's Apache ZooKeeper nodes.
5. On the client machine run the following command, replacing *Apache-ZooKeeper-node* with the address of one of the Apache ZooKeeper nodes that you obtained in the previous step. The number 2181 is the port number. Repeat for all the Apache ZooKeeper nodes.

   ```
   telnet Apache-ZooKeeper-node 2181
   ```

If the client machine is able to access the brokers and the Apache ZooKeeper nodes, 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 "Getting the bootstrap brokers" (p. 21). 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 and the Apache ZooKeeper nodes, 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 *Getting started* (p. 5).

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-xxxxxxxx)`.
- 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 (p. 173) page.

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 (p. 118).

Best practices

This topic outlines some best practices to follow when using Amazon MSK.

Right-size your cluster: Number of partitions per broker

The following table shows the maximum number of partitions (including leader and follower replicas) that you can have per broker.

Broker type	**Maximum number of partitions (including leader and follower replicas) per broker**
`kafka.t3.small`	300
`kafka.m5.large` or `kafka.m5.xlarge`	1000
`kafka.m5.2xlarge`	2000
`kafka.m5.4xlarge`, `kafka.m5.8xlarge`, `kafka.m5.12xlarge`, `kafka.m5.16xlarge`, or `kafka.m5.24xlarge`	4000

If the number of partitions per broker exceeds the maximum value specified in the previous table, you cannot perform any of the following operations on the cluster:

- Update the cluster configuration
- Update the Apache Kafka version for the cluster
- Update the cluster to a smaller broker type
- Associate an AWS Secrets Manager secret with a cluster that has SASL/SCRAM authentication

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 type for your brokers. For more information about the different broker types, see the section called “Broker types” (p. 10).

Right-size your cluster: Number of brokers per cluster

To determine the right number of brokers for your MSK cluster and understand costs, see the MSK Sizing and Pricing spreadsheet. This spreadsheet provides an estimate for sizing an MSK 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 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 on how to continuously verify the size of your production clusters.

Build highly available clusters

Use the following recommendations so that your MSK cluster can be highly available during an update (such as when you're updating the broker type 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.
- 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 the section called “Getting the bootstrap brokers” (p. 21).

Monitor CPU usage

Amazon MSK strongly recommends 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. One example of when this is necessary is when Amazon MSK detects and recovers from a broker fault; in this case, Amazon MSK performs automatic maintenance, like patching. Another example is when a user requests a broker-type change or version upgrade; in these two cases, Amazon MSK deploys 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 can ensure you have enough CPU headroom in your cluster to tolerate operational events like these.

You can use Amazon CloudWatch metric math 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 (recommended): Update your broker type to the next larger type. For example, if the current type is `kafka.m5.large`, update the cluster to use `kafka.m5.xlarge`. Keep in mind that when you update the broker type 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 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. Amazon MSK recommends 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.

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” (p. 25). You can also manually increase broker storage as described in the section called “Manual scaling” (p. 27).
- Reduce the message retention period or log size. For information on how to do that, see the section called “Adjust data retention parameters” (p. 190).
- 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 *Monitoring a cluster* (p. 155).

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 configurations” (p. 39).

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 --zookeeper ZooKeeperConnectionString --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 --zookeeper ZooKeeperConnectionString --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.

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

If you use Apache ZooKeeper commands to add brokers, these brokers don't get added to your MSK cluster, and your Apache ZooKeeper will contain incorrect information about the cluster. This might result in data loss. For supported cluster operations, see *How it works* (p. 10).

Enable in-transit encryption

For information about encryption in transit and how to enable it, see the section called "Encryption in transit" (p. 107).

Reassign partitions

To move partitions to different brokers on the same cluster, you can use the partition reassignment tool named `kafka-reassign-partitions.sh`. For example, after you add new brokers to expand a cluster, you can rebalance that cluster by reassigning partitions to the new brokers. For information about how to add brokers to a cluster, see the section called "Expanding a cluster" (p. 31). For information about the partition reassignment tool, see Expanding your cluster in the Apache Kafka documentation.

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.