Amazon Managed Streaming for Apache Kafka
Developer Guide
## Table of Contents

What Is Amazon MSK? ................................................................................................................................. 1  
Setting Up .......................................................................................................................................................... 4  
  Sign Up for AWS .......................................................................................................................................... 4  
  Download Libraries and Tools .................................................................................................................... 4  
Getting Started .................................................................................................................................................. 5  
  Step 1: Create a VPC ................................................................................................................................. 5  
  Step 2: Enable High Availability and Fault Tolerance .............................................................................. 8  
  Step 3: Create a Cluster ............................................................................................................................ 9  
  Step 4: Create a Client Machine ............................................................................................................. 11  
  Step 5: Create a Topic ............................................................................................................................. 11  
  Step 6: Produce and Consume Data ...................................................................................................... 13  
  Step 7: View Metrics ............................................................................................................................... 14  
  Step 8: Delete the Cluster ....................................................................................................................... 15  
How It Works .................................................................................................................................................... 16  
Creating a Cluster ............................................................................................................................................ 16  
  Broker types ............................................................................................................................................... 16  
  Creating a cluster using the AWS Management Console ...................................................................... 17  
  Creating a cluster using the AWS CLI ...................................................................................................... 18  
  Creating a cluster with a custom MSK configuration using the AWS CLI ........................................ 19  
  Creating a cluster using the API ............................................................................................................. 19  
Deleting a Cluster ........................................................................................................................................ 19  
  Deleting a cluster using the AWS Management Console .................................................................. 19  
  Deleting a cluster using the AWS CLI .................................................................................................... 19  
  Deleting a cluster using the API ............................................................................................................. 20  
Getting the Apache ZooKeeper Connection String .............................................................................. 20  
  Getting the Apache ZooKeeper connection string using the AWS Management Console ............ 20  
  Getting the Apache ZooKeeper connection string using the AWS CLI ............................................. 20  
  Getting the Apache ZooKeeper connection string using the API ..................................................... 21  
Getting the Bootstrap Brokers ............................................................................................................... 21  
  Getting the bootstrap brokers using the AWS Management Console ........................................... 21  
  Getting the bootstrap brokers using the AWS CLI ............................................................................ 22  
  Getting the bootstrap brokers using the API ....................................................................................... 22  
Listing Clusters ............................................................................................................................................ 22  
  Listing clusters using the AWS Management Console ................................................................... 22  
  Listing clusters using the AWS CLI ...................................................................................................... 22  
  Listing clusters using the API .............................................................................................................. 22  
Scaling Up Broker Storage ...................................................................................................................... 22  
  Automatic scaling ..................................................................................................................................... 23  
  Manual scaling .......................................................................................................................................... 24  
Updating the Broker Type ......................................................................................................................... 25  
  Updating the broker type using the AWS Management Console .................................................... 25  
  Updating the broker type using the AWS CLI ...................................................................................... 26  
  Updating the broker type using the API ............................................................................................... 27  
Updating the Configuration of a Cluster ............................................................................................... 27  
  Updating the configuration of a cluster using the AWS CLI .............................................................. 27  
  Updating the configuration of a cluster using the API ...................................................................... 28  
Expanding a Cluster .................................................................................................................................... 29  
  Expanding a cluster using the AWS Management Console ............................................................ 29  
  Expanding a cluster using the AWS CLI ............................................................................................... 29  
  Expanding a cluster using the API ....................................................................................................... 30  
Updating Security ......................................................................................................................................... 30  
  Updating a cluster's security settings using the AWS Management Console .................................. 31  
  Updating a cluster's security settings using the AWS CLI ................................................................. 31  
  Updating a cluster's security settings using the API ........................................................................... 32
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. 134).

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.

To get started using Amazon MSK, see *Getting Started* (p. 5).

To see the control-plane operations available through Amazon MSK, see the Amazon MSK API Reference.

After you create a cluster, you can use Amazon CloudWatch to monitor it. For more information about monitoring your cluster using metrics, see *Monitoring a Cluster* (p. 114).
Setting Up Amazon MSK

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

Tasks
- Sign Up for AWS (p. 4)
- Download Libraries and Tools (p. 4)

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
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.

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 Amazon MSK. 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 section 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.

This is a step-by-step tutorial that uses the AWS Management Console and the AWS CLI. If you prefer to follow a different approach, you can perform the types of AWS operations described in this tutorial using only the AWS Management Console, only the AWS CLI, only the AWS API, or a combination of the three environments. Alternatively, you can use AWS CloudFormation to set up an MSK cluster. For some example AWS CloudFormation templates, see Amazon MSK CloudFormation Examples. For more on the use of Amazon MSK with AWS CloudFormation templates, see AWS Streaming Data Solution for Amazon MSK: The AWS Streaming Data Solution for Amazon MSK provides AWS CloudFormation templates where data flows through producers, streaming storage, consumers, and destinations.

Prerequisites

Before you start, ensure that you have an Amazon Web Services account and that you have the AWS Command Line Interface (AWS CLI) installed on your computer. For more information about these prerequisites, see Setting Up (p. 4).

Important

Make sure that you have the latest version of the AWS CLI. Earlier versions might not have the full functionality of Amazon MSK.

Topics

- Step 1: Create a VPC for Your MSK Cluster (p. 5)
- Step 2: Enable High Availability and Fault Tolerance (p. 8)
- Step 3: Create an Amazon MSK Cluster (p. 9)
- Step 4: Create a Client Machine (p. 11)
- Step 5: Create a Topic (p. 11)
- Step 6: Produce and Consume Data (p. 13)
- Step 7: Use Amazon CloudWatch to View Amazon MSK Metrics (p. 14)
- Step 8: Delete the Amazon MSK Cluster (p. 15)

Step 1: Create a VPC for Your MSK Cluster

In the first step of Getting Started Using Amazon MSK (p. 5), you use the Amazon Virtual Private Cloud Console to create an Amazon Virtual Private Cloud (Amazon VPC). You create an MSK cluster in this VPC in a later step.

To create a VPC

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose Launch VPC Wizard.
3. Choose Select to accept the default Amazon VPC configuration named VPC with a Single Public Subnet.
4. For **VPC name**, enter `AWSKafkaTutorialVPC`. If you copy this name and paste it into the console, delete any white space that gets into the text field before the name.

5. For **Availability Zone**, choose `us-east-1a`.

6. For **Subnet name**, enter `AWSKafkaTutorialSubnet-1`. If you copy this name and paste it into the console, delete any white space that gets into the text field before the name.

7. Choose **Create VPC**, and then choose **OK**.

8. In the list of VPCs, find `AWSKafkaTutorialVPC` and copy its ID from the **VPC ID** column. Save this ID somewhere because you need it in some of the following steps.

**Next Step**

**Step 2: Enable High Availability and Fault Tolerance (p. 8)**
Step 2: Enable High Availability and Fault Tolerance

In this step of Getting Started Using Amazon MSK (p. 5), you enable high availability and fault tolerance. To do so, you add two subnets to the VPC that you created with one subnet in the previous step. After you complete this step, you will have three subnets in three different Availability Zones.

To add subnets to your VPC

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose Subnets.
3. In the list of subnets, find AWSKafkaTutorialSubnet-1, and then find the column named Route table. Copy the value associated with AWSKafkaTutorialSubnet-1 in that column and save it for later.
4. Choose Create subnet.
5. For the Name tag, enter AWSKafkaTutorialSubnet-2. If you copy this name and paste it into the console, delete any white space that gets into the text field before the name.
6. For VPC, choose AWSKafkaTutorialVPC.
7. For Availability Zone, choose us-east-1b.
8. For IPv4 CIDR block, enter 10.0.1.0/24.
9. Choose Create, and then choose Close.
10. Choose AWSKafkaTutorialSubnet-2 from the list of subnets by selecting the check box next to it. Ensure that no other check boxes in the list are selected.
11. In the subnet view near the bottom of the page, choose the Route Table tab, and then choose Edit route table association.
12. In the Route Table ID list, choose the route table whose value you copied earlier in this procedure.
13. Choose Save, and then choose Close.
14. Repeat this procedure to create another subnet with the name AWSKafkaTutorialSubnet-3, in the us-east-1c Availability Zone, and with the IPv4 CIDR block set to 10.0.2.0/24. If you copy the name of the subnet and paste it into the console, delete any white space that gets into the text field before the name.
15. Edit the route table for AWSKafkaTutorialSubnet-3 to ensure that it has the same route table used for AWSKafkaTutorialSubnet-1 and AWSKafkaTutorialSubnet-2.

Next Step

Step 3: Create an Amazon MSK Cluster (p. 9)

Step 3: Create an Amazon MSK Cluster

In this step of Getting Started Using Amazon MSK (p. 5), you create an Amazon MSK cluster in the VPC.

To create an Amazon MSK cluster using the AWS CLI

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose Subnets, and then copy the subnet IDs of the three subnets you created previously. Save these IDs because you need them later in this procedure.
3. In the navigation pane, choose Security Groups. Then in the table of security groups, find the group for which the VPC ID column has the ID you saved for AWSKafkaTutorialVPC. Copy the ID of this security group and save it because you need it later in this procedure.
4. Copy the following JSON and save it to a file. Name the file clusterinfo.json.
In the `clusterinfo.json` file, replace the placeholders for the three subnet IDs and the security group ID with the values that you saved in previous steps.

5. In `clusterinfo.json`, replace `your-CMK` with a customer managed CMK. You can also remove `EncryptionAtRest` and let Amazon MSK create a CMK and use it on your behalf. Setting `InCluster` to `true` means that you want Amazon MSK to encrypt your data as it travels between brokers within the cluster. For `ClientBroker` you can choose one of the following settings: `TLS`, `TLS.PLAINTEXT`, or `PLAINTEXT`. In this exercise, we use `TLS` to indicate that we want data to be encrypted as it travels between clients and brokers. For more information about encryption settings, see the section called "Encryption" (p. 75).

6. Upgrade your AWS CLI to the latest version to ensure that it has support for Amazon MSK. For detailed instructions on how to upgrade the AWS CLI, see Installing the AWS Command Line Interface.

7. Run the following AWS CLI command in the directory where you saved the `clusterinfo.json` file.

```bash
aws kafka create-cluster --cli-input-json fileb://clusterinfo.json
```

The output of the command looks like the following JSON:

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

8. Save the value of the `ClusterArn` key because you need it later.

9. Ensure that you saved `ClusterArn` before you proceed.

Next Step
Step 4: 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, we’ll put this client machine in the same VPC as the Amazon MSK cluster. But a client machine doesn’t have to be in the same VPC as the cluster.

To create a client machine

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose Launch Instance.
3. Choose Select to create an instance of Amazon Linux 2 AMI (HVM), SSD Volume Type.
4. Choose the t2.xlarge instance type by selecting the check box next to it.
5. Choose Next: Configure Instance Details.
6. In the Network list, choose AWSKafkaTutorialVPC.
7. In the Auto-assign Public IP list, choose Enable.
8. In the menu near the top, choose 5. Add Tags.
9. Choose Add Tag.
10. Enter Name for the Key and AWSKafkaTutorialClient for the Value.
11. Choose Review and Launch, and then choose Launch.
12. 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 if you prefer.
13. Read the acknowledgement, select the check box next to it, and choose Launch Instances.
14. Choose View Instances. Then, in the Security Groups column, choose the security group that is associated with the AWSKafkaTutorialClient instance.
15. Copy the value of Group ID (and not the group name) that is associated with the security group, and save it for later.
17. In the navigation pane, choose Security Groups. In the VPC ID column of the security groups, find the row that contains the ID you saved for AWSKafkaTutorialVPC, and where the Description column has the value default VPC security group. Choose this row by selecting the check box in the first column.
20. In the new rule, choose All traffic in the Type column. In the second field in the Source column, enter the ID of the security group of the client machine. This is the group ID that you saved earlier.
21. Choose Save rules.

Next Step

Step 5: 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, and then choose AWSKafkaTutorialClient by selecting the check box next to it.
3. Choose Actions, and then choose Connect. Follow the instructions to connect to the client machine AWSKafkaTutorialClient.
4. Install Java on the client machine by running the following command:

   ```sh
   sudo yum install java-1.8.0
   ```

5. Run the following command to download Apache Kafka.

   ```sh
   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.

   ```sh
   tar -xzf kafka_2.12-2.2.1.tgz
   ```

7. Go to the kafka_2.12-2.2.1 directory.

8. Cluster creation can take a few minutes. To find out whether the cluster you created is ready, run the following command, replacing `ClusterArn` with the Amazon Resource Name (ARN) that you obtained at the end of the section called “Step 3: Create a Cluster” (p. 9).

   ```sh
   aws kafka describe-cluster --region us-east-1 --cluster-arn "ClusterArn"
   ```

The result of running this command looks like the following JSON:

```json
{
   "ClusterInfo": {
      "BrokerAZDistribution": "DEFAULT",
      "ClientSubnets": [
         "subnet-0d44a1567c2ce409a",
         "subnet-051201cac6561565",
         "subnet-08b4ecf2bd3b8c2"
      ],
      "InstanceType": "kafka.m5.large",
      "SecurityGroups": [
         "sg-041e78b0a68ba7f834"
      ],
      "StorageInfo": {
         "EbsStorageInfo": {
            "VolumeSize": 1000
         }
      },
      "ClusterArn": "...",
      "ClusterName": "AWSKafkaTutorialCluster",
      "CreationTime": "2018-11-06T01:36:57.451Z",
      "CurrentBrokerSoftwareInfo": {
         "KafkaVersion": "2.2.1"
      },
      "CurrentVersion": "K3UN6WX5RRO2AG",
   }
}```
"EncryptionInfo": {
    "EncryptionAtRest": {
        "DataVolumeKMSKeyId": "arn:aws:kms:us-east-1:012345678901:key/a7de6539-7d2e-4e71-a279-aaa5555878"
    },
    "EnhancedMonitoring": "DEFAULT",
    "NumberOfBrokerNodes": 3,
    "State": "CREATING"
}
}

If the output of the command shows that the state of the cluster is still CREATING, wait a few minutes, and then run the command again. Keep running this command every few minutes until the state turns to ACTIVE. When the state is ACTIVE, the output of this describe-cluster command includes an additional key named ZookeeperConnectString. Copy the entire value associated with this key because you need it to create an Apache Kafka topic in the following command.

9. Run the following command, replacing ZookeeperConnectString with the value that you saved after you ran the describe-cluster command.

```
bin/kafka-topics.sh --create --zookeeper ZookeeperConnectString --replication-factor 3 --partitions 1 --topic AWSKafkaTutorialTopic
```

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

**Next Step**

**Step 6: Produce and Consume Data (p. 13)**

**Step 6: 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. 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, replacing JDKFolder with the name of your JDK folder. For example, the name of the JDK folder on your instance might be 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
   ```

2. 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
   ```

3. Run the following command, replacing ClusterArn with the Amazon Resource Name (ARN) that you obtained at the end of the section called “Step 3: Create a Cluster” (p. 9).

   ```
   aws kafka get-bootstrap-brokers --region us-east-1 --cluster-arn ClusterArn
   ```
From the JSON result of the command, save the value associated with the string named "BootstrapBrokerStringTls" because you need it in the following commands.

4. Run the following command in the bin folder, replacing $BootstrapBrokerStringTls$ with the value that you obtained when you ran the previous command.

```
./kafka-console-producer.sh --broker-list $BootstrapBrokerStringTls$ --producer.config client.properties --topic AWSKafkaTutorialTopic
```

5. 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.

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

7. In the following command, replace $BootstrapBrokerStringTls$ with the value that you saved earlier. Then, go to the bin folder and run the command using your second connection to the client machine.

```
./kafka-console-consumer.sh --bootstrap-server $BootstrapBrokerStringTls$ --consumer.config client.properties --topic AWSKafkaTutorialTopic --from-beginning
```

You start seeing the messages you entered earlier when you used the console producer command. These messages are TLS encrypted in transit.

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

Next Step

Step 7: Use Amazon CloudWatch to View Amazon MSK Metrics (p. 14)

**Step 7: 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**

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 8: Delete the Amazon MSK Cluster (p. 15)
Step 8: Delete the Amazon MSK Cluster

In the final step of Getting Started Using Amazon MSK (p. 5), you delete the MSK cluster that you created in Step 3: Create an Amazon MSK Cluster (p. 9).

To delete the Amazon MSK cluster using the AWS CLI

1. Run the following command on the computer where you have the AWS CLI installed.

   ```bash
   aws kafka list-clusters --region us-east-1
   ```

2. In the output of the `list-clusters` command, look for the cluster Amazon Resource Name (ARN) that corresponds to the cluster that you want to delete. Copy that ARN.

3. Run the following command, replacing `ClusterArn` with the ARN that you obtained when you ran the previous command.

   ```bash
   aws kafka delete-cluster --region us-east-1 --cluster-arn ClusterArn
   ```
Creating an Amazon MSK Cluster

Before you can create an Amazon MSK cluster you need to have a 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. For an example of how to set up a VPC and subnets for an MSK cluster, see ??? (p. 5) and ??? (p. 8).

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.m5.2xlarge, 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
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 MSK configurations, see **Configuration** (p. 36).
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. 34).
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, 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. 75).
12. Choose the kind of CMK that you want to use for encrypting data at rest. For more information, see the section called **Encryption at Rest** (p. 75).
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. 93).
14. Choose the monitoring level you want. This determines the set of metrics you get. For more information, see **Monitoring a Cluster** (p. 114).
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 EC2 Security Groups Associated With a Specific VPC, Programmatically and in the Console.
16. Choose **Create cluster**.
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 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. 114). 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 fileb://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.
Creating a cluster with a custom MSK configuration using the AWS CLI

For information about custom MSK configurations and how to create them, see Configuration (p. 36).

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.

   ```json
   {
     "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.

   ```bash
   aws kafka create-cluster --cluster-name ExampleClusterName --broker-node-group-info fileb://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.

   ```json
   {
     "ClusterName": "CustomConfigExampleCluster",
     "State": "CREATING"
   }
   ```

Creating a cluster using the API

To create a cluster using the API, see CreateCluster.

Deleting an Amazon MSK Cluster

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).
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.

```json
{
    "ClusterInfo": {
        "BrokerNodeGroupInfo": {
            "BrokerAZDistribution": "DEFAULT",
            "ClientSubnets": [
                "subnet-0123456789abcdef0",
                "subnet-2468013579abcdef1",
                "subnet-1357902468abcdef2"
            ],
            "InstanceType": "kafka.m5.large",
            "StorageInfo": {
                "EbsStorageInfo": {
                    "VolumeSize": 1000
                }
            }
        }
    },
    "OpenNodes": 0
}
```
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](https://docs.aws.amazon.com/AmazonMSK/latest/APIReference/API_DescribeCluster.html).

### 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/](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. 85), 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 number of bootstrap brokers in the string is equal to the number of zones in which your MSK cluster is deployed.

**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.

**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
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.

Topics
- Automatic scaling (p. 23)
- Manual scaling (p. 24)

Automatic scaling
You can configure Amazon Managed Streaming for Apache Kafka to automatically expand your cluster's storage in response to increased usage using Application Auto-Scaling policies. Your auto-scaling policy sets the target disk utilization and the maximum scaling capacity.

Note
The Amazon MSK service 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. 111).

MSK Storage auto-expansion details
Your 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**: This setting is the maximum scaling limit that Amazon MSK can set your broker storage. You can set the maximum storage capacity up to 16 TiB per broker. For more information, see Amazon MSK Quota (p. 132).

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

Important
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. To repeatedly expand your storage capacity more quickly, set the storage capacity manually using the UpdateBrokerStorage operation. For information about right-sizing your storage, see the section called "Right-size your cluster" (p. 145).

You can check to determine if auto scaling operations occurred using the ListClusterOperations operation.

Setting up auto-expansion for your Amazon MSK cluster
You can use the Amazon MSK console or the Amazon MSK API to implement auto-expanding storage.
**Note**
You can't implement auto-expansion when you create a cluster. You must first create the cluster, and then create and enable an auto-expansion policy for it. However, you can create the policy while the Amazon MSK service creates your cluster.

**Setting up auto-expansion 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 the cluster's Details page.
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 auto-expansion 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 auto-expansion 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.

```bash
```

### 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. 145). 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 "Number of partitions per broker" (p. 145).

### Updating the broker type using the AWS Management Console

1. Open the Amazon MSK console at [https://console.aws.amazon.com/msk/](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. 16).

```bash
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:

```bash
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.

```json
{
    "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.

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

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

```json
{
    "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. 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](#).

**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.

   ```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).

   ```bash
   ```
Updating the configuration of a cluster using the API

To create a cluster using the API, see `UpdateClusterConfiguration`. 

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.

The output of the `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/abcdefab-1234-abcd-123e8e8e-1",
        "Revision": 1
      }
    }
  }
}
```
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. 148).

**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 KTVPDKIXXODER.

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.

```bash
```

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

```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"
}
```
Expanding a cluster using the API

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.

```bash
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",
        "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 a cluster's security settings

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.

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Client-Broker Encryption Options</th>
<th>Broker-Broker Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unauthenticated</td>
<td>TLS, PLAINTEXT, TLS_PLAINTEXT</td>
<td>Can be on or off.</td>
</tr>
</tbody>
</table>
Amazon Managed Streaming for Apache Kafka Developer Guide

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

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Client-Broker Encryption Options</th>
<th>Broker-Broker Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTLS</td>
<td>TLS, TLS_PLAINTEXT</td>
<td>Must be on.</td>
</tr>
<tr>
<td>SASL/SCRAM</td>
<td>TLS</td>
<td>Must be on.</td>
</tr>
<tr>
<td>SASL/IAM</td>
<td>TLS</td>
<td>Must be on.</td>
</tr>
</tbody>
</table>

For more information about security settings, see Security (p. 74).

**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.

   ```json
   { "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.

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

3. Run the following AWS CLI command:

   ```bash
   ```

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

   ```json
   {
     "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/abcdefab-1234-abcd-5678-cdef0133ab01-2",
     "ClusterOperationArn": "arn:aws:kafka:us-east-1:012345678012:cluster-operation/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2/0123abcd-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.

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

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

```json
{
   "ClusterOperationInfo": {
      "ClientRequestId": "c0b7af47-8591-45b5-9c0c-909a1a2c99ea",
      "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2",
      "CreationTime": "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/.
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.

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

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

   ```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.

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

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

   ```json
   {
     "ClusterOperationInfo": {
       "ClientRequestId": "c0b7af47-8591-45b5-9c0c-909a1a2c99ea",
       "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/abcdefab-1234-abcd-5678-cdef0123ab01-2",
       "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 (p. 34)
- Tracking Costs Using Tagging (p. 34)
- Tag Restrictions (p. 34)
- Tagging Resources Using the Amazon MSK API (p. 35)

**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 and Cost Management 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 MSK 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 (p. 36)
- The Default Amazon MSK Configuration (p. 41)
- Amazon MSK Configuration Operations (p. 43)

Custom MSK Configurations

Amazon MSK enables you 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. 41). For more information about configuration properties, see Apache Kafka Configuration.

**Apache Kafka Configuration Properties That You Can Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto.create.topics.enable</td>
<td>Enables topic autocreation on the server.</td>
</tr>
<tr>
<td>compression.type</td>
<td>The final compression type for a given topic. You can set this property to</td>
</tr>
<tr>
<td></td>
<td>the standard compression codecs (gzip, snappy, lz4, and zstd). It additionally</td>
</tr>
<tr>
<td></td>
<td>accepts uncompressed, which is equivalent to no compression; and producer,</td>
</tr>
<tr>
<td></td>
<td>which means retain the original compression codec set by the producer.</td>
</tr>
<tr>
<td>default.replication.factor</td>
<td>The default replication factor for automatically created topics.</td>
</tr>
<tr>
<td>delete.topic.enable</td>
<td>Enables the delete topic operation. If this config is turned off, you can</td>
</tr>
<tr>
<td></td>
<td>'t delete a topic through the admin tool.</td>
</tr>
<tr>
<td>group.initial.rebalance.delay.ms</td>
<td>Amount of time the group coordinator waits for more consumers to join a</td>
</tr>
<tr>
<td></td>
<td>new group before performing the first rebalance. A longer delay means</td>
</tr>
<tr>
<td></td>
<td>potentially fewer rebalances, but increases the time until processing</td>
</tr>
<tr>
<td></td>
<td>begins.</td>
</tr>
<tr>
<td>group.max.session.timeout.ms</td>
<td>Maximum session timeout for registered consumers. Longer timeouts give</td>
</tr>
<tr>
<td></td>
<td>consumers more time to process messages in between heartbeats at the cost</td>
</tr>
<tr>
<td></td>
<td>of a longer time to detect failures.</td>
</tr>
<tr>
<td>group.min.session.timeout.ms</td>
<td>Minimum session timeout for registered consumers. Shorter timeouts result</td>
</tr>
<tr>
<td></td>
<td>in quicker failure detection at the cost of more frequent consumer</td>
</tr>
<tr>
<td></td>
<td>heartbeating, which can overwhelm broker resources.</td>
</tr>
</tbody>
</table>

36
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>leader.imbalance.per.broker.percentage</td>
<td>The ratio of leader imbalance allowed per broker. The controller triggers a leader balance if it goes above this value per broker. This value is specified in percentage.</td>
</tr>
<tr>
<td>log.cleaner.delete.retention.ms</td>
<td>Amount of time that you want Apache Kafka to retain deleted records. The minimum value is 0.</td>
</tr>
<tr>
<td>log.cleaner.min.cleanable.ratio</td>
<td>This configuration property can have values between 0 and 1. It determines how frequently the log compactor attempts to clean the log (assuming log compaction is enabled). By default, Apache Kafka avoids cleaning a log where more than 50% of the log has been compacted. This ratio bounds the maximum space wasted in the log by duplicates (at 50%, which 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.</td>
</tr>
<tr>
<td>log.cleanup.policy</td>
<td>The default cleanup policy for segments beyond the retention window. A comma-separated list of valid policies. Valid policies are delete and compact.</td>
</tr>
<tr>
<td>log.flush.interval.messages</td>
<td>Number of messages accumulated on a log partition before messages are flushed to disk.</td>
</tr>
<tr>
<td>log.flush.interval.ms</td>
<td>Maximum time in ms that a message in any topic is kept in memory before flushed to disk. If not set, the value in log.flush.scheduler.interval.ms is used. The minimum value is 0.</td>
</tr>
<tr>
<td>log.message.timestamp.difference.max.ms</td>
<td>The maximum difference allowed 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.</td>
</tr>
<tr>
<td>log.message.timestamp.type</td>
<td>Specifies whether the timestamp in the message is the message creation time or the log append time. The allowed values are CreateTime and LogAppendTime.</td>
</tr>
<tr>
<td>log.retention.bytes</td>
<td>Maximum size of the log before deleting it.</td>
</tr>
<tr>
<td>log.retention.hours</td>
<td>Number of hours to keep a log file before deleting it, tertiary to the log.retention.ms property.</td>
</tr>
<tr>
<td>log.retention.minutes</td>
<td>Number of minutes to keep a log file before deleting it, secondary to log.retention.ms property. If not set, the value in log.retention.hours is used.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>log.retention.ms</td>
<td>Number of milliseconds to keep a log file before deleting it (in milliseconds), If not set, the value in log.retention.minutes is used.</td>
</tr>
<tr>
<td>log.roll.ms</td>
<td>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.</td>
</tr>
<tr>
<td>log.segment.bytes</td>
<td>Maximum size of a single log file.</td>
</tr>
<tr>
<td>max.incremental.fetch.session.cache.slots</td>
<td>Maximum number of incremental fetch sessions that are maintained.</td>
</tr>
<tr>
<td>message.max.bytes</td>
<td>Largest record batch size allowed by Kafka. If this is increased and there are consumers older than 0.10.2, the consumers’ fetch size must also be increased so that the they can fetch record batches this large.</td>
</tr>
<tr>
<td></td>
<td>In the latest message format version, records are always grouped into batches for efficiency. In previous message format versions, uncompressed records are not grouped into batches and this limit only applies to a single record in that case.</td>
</tr>
<tr>
<td></td>
<td>This can be set per topic with the topic level max.message.bytes config.</td>
</tr>
<tr>
<td>min.insync.replicas</td>
<td>When a producer sets acks to &quot;all&quot; (or &quot;-1&quot;), 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).</td>
</tr>
<tr>
<td></td>
<td>When used together, min.insync.replicas and acks enable you to enforce greater durability guarantees. A typical scenario would be to create a topic with a replication factor of 3, set min.insync.replicas to 2, and produce with acks of &quot;all&quot;. This ensures that the producer raises an exception if a majority of replicas don’t receive a write.</td>
</tr>
<tr>
<td>num.io.threads</td>
<td>The number of threads that the server uses for processing requests, which may include disk I/O.</td>
</tr>
<tr>
<td>num.network.threads</td>
<td>The number of threads that the server uses for receiving requests from the network and sending responses to it.</td>
</tr>
<tr>
<td>num.partitions</td>
<td>Default number of log partitions per topic.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>num.recovery.threads.per.data.dir</td>
<td>The number of threads per data directory to be used for log recovery at startup and for flushing at shutdown.</td>
</tr>
<tr>
<td>num.replica.fetchers</td>
<td>The number of fetcher threads used to replicate messages from a source broker. Increasing this value can increase the degree of I/O parallelism in the follower broker.</td>
</tr>
<tr>
<td>offsets.retention.minutes</td>
<td>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, using manual assignment), offsets are expired after the time of the last commit plus this retention period.</td>
</tr>
<tr>
<td>offsets.topic.replication.factor</td>
<td>The replication factor for the offsets topic (set higher to ensure availability). Internal topic creation fails until the cluster size meets this replication factor requirement.</td>
</tr>
<tr>
<td>replica.fetch.max.bytes</td>
<td>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 that progress can be made. The maximum record batch size accepted by the broker is defined via message.max.bytes (broker config) or max.message.bytes (topic config).</td>
</tr>
<tr>
<td>replica.fetch.response.max.bytes</td>
<td>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 that progress can be made. 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.</td>
</tr>
<tr>
<td>replica.lag.time.max.ms</td>
<td>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.</td>
</tr>
</tbody>
</table>
| MinValue: 10000                                | MaxValue (inclusive) = 30000


<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replica.selector.class</td>
<td>The fully-qualified class name that implements ReplicaSelector. This is used by the broker to find the preferred read replica. If you are using Apache Kafka version 2.4.1 or higher, and want to allow consumers to fetch from the closest replica, set this property to <code>org.apache.kafka.common.replica.RackAwareReplicaSelector</code>. For more information, see the section called “Apache Kafka version 2.4.1 (use 2.4.1.1 instead)” (p. 135).</td>
</tr>
<tr>
<td>replica.socket.receive.buffer.bytes</td>
<td>The socket receive buffer for network requests.</td>
</tr>
<tr>
<td>socket.receive.buffer.bytes</td>
<td>The SO_RCVBUF buffer of the socket server sockets. The minimum value to which you can set this property is -1. If the value is -1, Amazon MSK uses the OS default.</td>
</tr>
<tr>
<td>socket.request.max.bytes</td>
<td>The maximum number of bytes in a socket request.</td>
</tr>
<tr>
<td>socket.send.buffer.bytes</td>
<td>The SO_SNDBUF buffer of the socket server sockets. The minimum value to which you can set this property is -1. If the value is -1, Amazon MSK uses the OS default.</td>
</tr>
<tr>
<td>transaction.max.timeout.ms</td>
<td>Maximum timeout for transactions. If a client's requested transaction time exceed this, the broker returns an error in InitProducerIdRequest. This prevents a client from too large of a timeout, which can stall consumers reading from topics included in the transaction.</td>
</tr>
<tr>
<td>transaction.state.log.min.isr</td>
<td>Overridden min.insync.replicas config for the transaction topic.</td>
</tr>
<tr>
<td>transaction.state.log.replication.factor</td>
<td>The replication factor for the transaction topic. Set it to a higher value to increase availability. Internal topic creation fails until the cluster size meets this replication factor requirement.</td>
</tr>
<tr>
<td>transactional.id.expiration.ms</td>
<td>The time in milliseconds that the transaction coordinator waits without receiving any transaction status updates for the current transaction before expiring its transactional ID. This setting also influences producer ID expiration: producer IDs are expired 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 due to the topic's retention settings. The minimum value for this property is 1 millisecond.</td>
</tr>
<tr>
<td>unclean.leader.election.enable</td>
<td>Indicates whether to enable replicas not in the ISR set to be elected as leader as a last resort, even though doing so may result in data loss.</td>
</tr>
</tbody>
</table>
Dynamic Configuration

In addition to the configuration properties that Amazon MSK provides, you can dynamically set cluster- and broker-level configuration properties that don’t require a broker restart. You can dynamically set configuration properties that aren’t marked as read-only in the table under Broker Configs in the Apache Kafka documentation. For information about 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 about topic-level configuration properties and examples on how to set them, see Topic-Level Configs in the Apache Kafka documentation.

**Configuration States**

Amazon MSK configurations can be in 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 without specifying 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow.everyone.if.no.acl.found</td>
<td>If no resource patterns match a specific resource, the resource has no associated ACLs. In this case, if this property is set to true, everyone is allowed to access the resource, not just the super users.</td>
<td>true</td>
</tr>
<tr>
<td>auto.create.topics.enable</td>
<td>Enables autocreation of a topic on the server.</td>
<td>false</td>
</tr>
<tr>
<td>auto.leader.rebalance.enable</td>
<td>Enables auto leader balancing. A background thread checks and triggers leader balance if required at regular intervals.</td>
<td>true</td>
</tr>
<tr>
<td>default.replication.factor</td>
<td>Default replication factors for automatically created topics.</td>
<td>3 for 3-AZ clusters, 2 for 2-AZ clusters</td>
</tr>
<tr>
<td>min.insync.replicas</td>
<td>When a producer sets acks to &quot;all&quot; (or &quot;-1&quot;), min.insync.replicas specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful. If this minimum can't be met, the producer raises an exception (either NotEnoughReplicas or NotEnoughReplicasAfterAppend).</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>When used together, min.insync.replicas and acks enable you to enforce greater durability guarantees. A typical scenario would be to create a topic with a replication factor of 3, set min.insync.replicas to 2, and produce with acks of &quot;all&quot;. This ensures that the producer raises an exception if a majority of replicas do not receive a write.</td>
<td></td>
</tr>
<tr>
<td>num.io.threads</td>
<td>Number of threads that the server uses for processing requests, which may include disk I/O.</td>
<td>8</td>
</tr>
<tr>
<td>num.network.threads</td>
<td>Number of threads that the server uses for receiving requests from the network</td>
<td>5</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>num.partitions</td>
<td>Default number of log partitions per topic.</td>
<td>1</td>
</tr>
<tr>
<td>num.replica.fetchers</td>
<td>Number of fetcher threads used to replicate messages from a source broker. Increasing this value can increase the degree of I/O parallelism in the follower broker.</td>
<td>2</td>
</tr>
<tr>
<td>replica.lag.time.max.ms</td>
<td>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.</td>
<td>30000</td>
</tr>
<tr>
<td>socket.receive.buffer.bytes</td>
<td>SO_RCVBUF buffer of the socket server sockets. If the value is -1, the OS default is used.</td>
<td>102400</td>
</tr>
<tr>
<td>socket.request.max.bytes</td>
<td>Maximum number of bytes in a socket request.</td>
<td>104857600</td>
</tr>
<tr>
<td>socket.send.buffer.bytes</td>
<td>SO_SNDBUF buffer of the socket server sockets. If the value is -1, the OS default is used.</td>
<td>102400</td>
</tr>
<tr>
<td>unclean.leader.election.enable</td>
<td>Indicates whether to enable replicas not in the ISR set to be elected as leader as a last resort, even though doing so may result in data loss.</td>
<td>true</td>
</tr>
<tr>
<td>zookeeper.session.timeout.ms</td>
<td>The Apache ZooKeeper session timeout in milliseconds.</td>
<td>18000</td>
</tr>
<tr>
<td>zookeeper.set.acl</td>
<td>Set client to use secure ACLs.</td>
<td>false</td>
</tr>
</tbody>
</table>

For information about how to specify custom configuration values, see the section called “Custom Configurations” (p. 36).

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. 16).

This topic contains the following sections:
- To create an MSK configuration (p. 44)
Create Configuration

- To update an MSK configuration (p. 44)
- To delete an MSK configuration (p. 45)
- To describe an MSK configuration (p. 46)
- To describe an MSK configuration revision (p. 46)
- To list all MSK configurations in your account for the current Region (p. 47)

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.

```plaintext
auto.create.topics.enable = true
zookeeper.connection.timeout.ms = 1000
log.roll.ms = 604800000
```

2. Run the following AWS CLI command, replacing `config-file-path` with the path to the file where you saved your configuration in the previous step.

   ```bash
   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 running this command.

   ```json
   {
   "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/abcdabcd-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 the newly created configuration. Save this ARN because you need it to refer to this configuration in other commands. If you lose your configuration ARN, you can find it again by listing all the configurations in your account.

To update an MSK configuration

First, ensure that the number of partitions per broker on your MSK cluster is under the limits described in the section called “Number of partitions per broker” (p. 145). You can't update the configuration of a cluster that exceeds these limits.

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.
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 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, and find 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 delete-configuration --arn configuration-arn
   ```

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

   ```
   {
   "arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/abcdabcd-1234-abcd123e8e8e-1",
   "state": "DELETING"
   }
   ```
To describe an MSK configuration

1. This 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 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, and find 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 --arn configuration-arn
```

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

```
{
  "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

Describing an MSK configuration using the `describe-configuration-revision` command, gives you the metadata of the configuration. To see a description of the configuration, use this command, `describe-configuration-revision`, instead.

- Run the following command, replacing `configuration-arn` with the ARN 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, and find 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 running this command.

```
{
  "CreationTime": "2019-05-21T00:54:23.591Z",
  "Description": "Example configuration description.",
  "Revision": 1,
  "ServerProperties": "YXV0by5jcmVhdGUudG9waWNzImluYjVyc29uZy5jcmVhdGUuY2F0aGluZ3M="
}
```
The value of `ServerProperties` is encoded using Base64. If you use a Base64 decoder (for example, https://www.base64decode.org/) to manually decode it, you get the contents of the original configuration file that you used to create the custom configuration. In this case, you get the following:

```text
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 running this command.

```json
{
  "Configurations": [
    {
      "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/abcdabcd-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-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"
    }
  ]
}
```
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, 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. 48).

To learn about the AWS resources that you can create with MSK Connect, see the section called “Connectors” (p. 54), the section called “Plugins” (p. 57), and the section called “Workers” (p. 57).

For information about the MSK Connect API, see the Amazon MSK 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 (p. 48)
- Step 2: Create custom plugin (p. 51)
- Step 3: Create client machine and Apache Kafka topic (p. 52)
- Step 4: Create connector (p. 53)
- Step 5: Send data (p. 54)

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. Enter mkc-tutorial-destination-bucket for the name of the bucket.
4. Scroll down and choose Create bucket.
5. In the list of bucket, choose the newly created mkc-tutorial-destination-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 Networking, choose an Amazon VPC, then set the Number of Zones to 2, then select the Availability Zones and subnets that you want to use. Remember the IDs of the Amazon VPC and subnets you selected because we need them later in this tutorial.
7. Under Access control methods ensure that only Unauthenticated access is selected.
8. Under Encryption ensure that only Plaintext is selected.
9. Scroll down and choose Create cluster. This takes you to the cluster's details page. On that page, look for the security group ID under Security groups applied. Remember that ID because we 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, then replace the text in the window with the following policy.

```json
{ "Version": "2012-10-17", "Statement": [ ...
```
Step 1: Set up required resources

```
{
  "Effect":"Allow",
  "Action":[
    "s3:ListAllMyBuckets"
  ],
  "Resource":"arn:aws:s3:::*"
},
{
  "Effect":"Allow",
  "Action":[
    "s3:ListBucket",
    "s3:GetBucketLocation"
  ],
  "Resource":"arn:aws:s3:::mkc-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.
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.
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**

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 Confluent 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. Go to the **Confluent Amazon S3 Sink Connector** page.
2. Find the section titled **Download installation**, then choose the **Download** button that is under that section.
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. 52)
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 instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose Launch instance.
3. Choose Select to create an instance of Amazon Linux 2 AMI (HVM), SSD Volume Type.
4. Choose the t2.xlarge instance type by selecting the check box next to it.
5. Choose Next: Configure Instance Details.
6. In the Network list, choose the same VPC whose name you saved when you created the cluster in the section called “Step 1: Set up required resources” (p. 48).
7. In the Auto-assign Public IP list, choose Enable.
8. In the menu near the top, choose 5. Add Tags.
9. Choose Add Tag.
10. Enter Name for the Key and mkc-tutorial-client for the Value.
11. Choose Review and Launch, and then choose Launch.
12. In the first list, choose the option to 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 if you prefer.
13. Choose Launch Instances.
14. In the bottom right part of the screen, choose View Instances.
15. In the list of instances, find mkc-tutorial-client. Choose it by selecting the box to its left. Make sure no other instances are also selected.
16. In the bottom half of the screen, choose the Security tab.
17. Under Security groups copy the ID of the security group. We use it in the following procedure.

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. 48). 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.
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 instance. This is the security group ID that you saved after you created the client in the previous procedure.
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 Apache ZooKeeper connection string that’s under the label Plaintext. Also copy the bootstrap servers string. You need both of these strings in the following steps.
12. Choose Done.
13. Run the following command on the client instance (mkc-tutorial-client), replacing
   `ZookeeperConnectString` with the value that you saved when you viewed the cluster’s client information.

   `bin/kafka-topics.sh --create --zookeeper ZookeeperConnectString --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. 53)**

**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.confluent.connect.s3.S3SinkConnector
tasks.max=2
topics=mkc-tutorial-topic
s3.region=us-east-1
s3.bucket.name=mkc-tutorial-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. 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**

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 in the **bin** folder, replacing **BootstrapBrokerString** with the value that you obtained when you ran the previous command.

```
./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 Architecture**

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.

- **Auto scaled**: Choose this mode if the capacity requirements for your connector are variable or if you don't know them in advance. You specify three sets of values:
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. 57). 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 Lenses Amazon S3 Sink Connector.

```
connector.class=io.lenses.streamreactor.connect.aws.s3.sink.S3SinkConnector
tasks.max=2
topics=connect-test-lenses
aws.region=us-east-1
connect.s3.kcql=INSERT INTO mkc-tutorial-destination-bucket:tutorial SELECT * FROM mkc-tutorial-topic
flush.size=1
key.converter.schemas.enable=false
value.converter=org.apache.kafka.connect.storage.StringConverter
key.converter=org.apache.kafka.connect.storage.StringConverter
schema.enable=false
errors.log.enable=true
```

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. 55).
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. 57).
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. 59).
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. 66).

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 Guide in the Apache Kafka documentation.

**Creating a custom plugin using the AWS Management Console**

1. Open the Amazon MSK console at https://console.aws.amazon.com/msk/.
2. In the left pane, under **MSK Connect**, choose **Custom plugins**.
3. Choose **Create custom plugin**.
4. Choose **Browse S3**.
5. In the list of S3 buckets, choose the bucket that has the JAR or ZIP file for the plugin.
6. In the list of object, select the box to the left of the JAR or ZIP file for the plugin, then choose **Choose**.
7. Choose **Create custom plugin**.

To use the MSK Connect API to create a custom plugin, see **CreateCustomPlugin**.

---

**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 and 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. 55).

**Default worker configuration**

MSK Connect provides the following default worker configuration:
Amazon Managed Streaming for Apache Kafka Developer Guide
Custom worker configurations

key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter

Custom worker configurations

MSK Connect provides a default worker configuration. Alternatively, you can create a custom worker configuration to use with your connectors. The following restrictions apply:

• A configuration property that starts with the `producer.` prefix is allowed only if it starts with one of the following prefixes:

```plaintext
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.request.timeout.ms
producer.retry.backoff.ms
producer.value.serializer
```

• A configuration property that starts with the `consumer.` prefix is allowed only if it starts with one of the following prefixes:

```plaintext
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.request.timeout.ms
consumer.retry.backoff.ms
consumer.session.timeout.ms
consumer.value.deserializer
```

• All configuration properties that don't start with the `producer.` or `consumer.` prefixes are allowed, except for the following properties:

```plaintext
access.control.
admin.
admin.listeners.http.
client.
connect.
inter.worker.
internal.
listeners.http.
metrics.
metrics.context.
reconnect.
rest.
sasl.
security.
```
For information about worker configuration properties and what they represent, see Kafka Connect Configs in the Apache Kafka documentation.

**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.

**IAM roles and policies for MSK Connect**

**Topics**
- Service execution role (p. 59)
- Examples of IAM policies for MSK Connect (p. 61)
- AWS managed policies for MSK Connect (p. 62)
- Using Service-Linked Roles for MSK Connect (p. 65)

**Service execution role**

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. The role must have the following trust policy so that MSK Connect can assume it.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "kafkaconnect.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

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. 91).
{  
  "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.

```json
{
   "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": "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 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.

```json
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "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"
  }
  ]
}
```
AWS managed policies

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.
AWS managed policies

```json
{
   "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",
         "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. For automatic alerts about changes to this page, subscribe to the RSS feed on the Document History (p. 149) page.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK Connect started tracking changes</td>
<td>MSK Connect started tracking changes for its AWS managed policies.</td>
<td>SEPTEMBER 14, 2021</td>
</tr>
</tbody>
</table>

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. 63).

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.

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.

MSK Connect emits the following types of log events:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>Runtime events of interest at startup and shutdown.</td>
</tr>
<tr>
<td>WARN</td>
<td>Runtime situations that aren't errors but are undesirable or unexpected.</td>
</tr>
<tr>
<td>FATAL</td>
<td>Severe errors that cause premature termination.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Unexpected conditions and runtime errors that aren't fatal.</td>
</tr>
</tbody>
</table>

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-
est=1.amazonaws.com/INTERNAL_IP) could not be established. Broker may not be available.
(org.apache.kafka.clients.NetworkClient:782)
```
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

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ByetsInPerSec</td>
<td>The total number of bytes received by the connector.</td>
</tr>
<tr>
<td>BytesOutPerSec</td>
<td>The total number of bytes delivered by the connector.</td>
</tr>
<tr>
<td>CpuUtilization</td>
<td>The percentage of CPU consumption by system and user.</td>
</tr>
<tr>
<td>ErroredTaskCount</td>
<td>The number of tasks that have errored out.</td>
</tr>
<tr>
<td>MemoryUtilization</td>
<td>The percentage of memory consumption for the connector.</td>
</tr>
<tr>
<td>RebalanceCompletedTotal</td>
<td>The total number of rebalances completed by this connector.</td>
</tr>
<tr>
<td>RebalanceTimeAvg</td>
<td>The average time in milliseconds spent by the connector on rebalancing.</td>
</tr>
<tr>
<td>RebalanceTimeMax</td>
<td>The maximum time in milliseconds spent by the connector on rebalancing.</td>
</tr>
<tr>
<td>RebalanceTimeSinceLast</td>
<td>The time in milliseconds since this connector completed the most recent rebalance.</td>
</tr>
<tr>
<td>RunningTaskCount</td>
<td>The running number of tasks in the connector.</td>
</tr>
<tr>
<td>SinkRecordReadRate</td>
<td>The average per-second number of records read from the Apache Kafka or Amazon MSK cluster.</td>
</tr>
<tr>
<td>SinkRecordSendRate</td>
<td>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.</td>
</tr>
<tr>
<td>SourceRecordPollRate</td>
<td>The average per-second number of records produced or polled.</td>
</tr>
</tbody>
</table>
## Examples

### 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. 59).

```json
{
    "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": "your cluster's bootstrap servers string",
            "Vpc": {
                "Subnets": [
                    "cluster's-subnet-1",
                    "cluster's-subnet-2",
                    "cluster's-subnet-3"
                ],
                "SecurityGroups": ["cluster's security group ID"]
            }
        }
    },
    "Capacity": {
        "ProvisionedCapacity": {
```
Amazon Managed Streaming for Apache Kafka Developer Guide
Amazon Redshift sink connector

```
"McuCount": 2,
"WorkerCount": 4
},
"KafkaConnectVersion": "2.7.1",
"ServiceExecutionRoleArn": "ARN of a role that MSK Connect can assume",
"Plugins": [
{
"CustomPlugin": {
"CustomPluginArn": "ARN of the plugin that contains the code for the
connector",
"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 fileb://connector-info.json
```

The following is an example of the output 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

This example shows how to use the Confluent Amazon Redshift sink connector and the AWS CLI to create an Amazon Redshift sink connector in MSK Connect.

1. Go to the Amazon Redshift Sink Connector page on the Confluent site.
2. Under the title Download installation choose Download.
3. Go to Use previous JDBC driver version 1.0 driver versions in certain cases, and download the latest JDBC-compatible driver.
4. Move the driver you downloaded in the previous step to the confluentinc-kafka-connect-aws-redshift-1.1.0/lib folder.
5. Use ZIP to compress the confluentinc-kafka-connect-aws-redshift-1.1.0 folder. Name the ZIP file confluentinc-kafka-connect-aws-redshift-1.1.0-withRedshiftJDBC.zip.
6. Upload the ZIP file from the previous step to an S3 bucket. In the following steps we use this ZIP file to create a custom plugin.
7. Copy the following JSON and paste it in a file, replacing your-S3-bucket-ARN with the ARN of the S3 bucket where you uploaded the ZIP file, and my-example-custom-plugin with the name that you want the plugin to have.

```
{
"Name": "my-example-custom-plugin",
"ContentType": "ZIP",
"Location": {
}
```
8. Give the file where you saved the JSON the name `redshift-sink-custom-plugin.json`.

9. Run the following AWS CLI command from the folder where you saved the JSON file.

```
aws kafkaconnect create-custom-plugin --cli-input-json file://redshift-sink-custom-plugin.json
```

The following JSON is an example of what the output of the command looks like.

```
{
    "CustomPluginArn": "arn:aws:kafkaconnect:us-east-1:012345678901:custom-plugin/my-example-custom-plugin/abcd1234-a0b0-1234-c1-12345678abcd-1",
    "CustomPluginState": "CREATING",
    "Name": "redshift-sink-example",
    "Revision": 1
}
```

10. To ensure that the custom plugin is active, run the following command, replacing the ARN placeholder with the ARN that you got in the output of the previous command.

```
aws kafkaconnect describe-custom-plugin --custom-plugin-arn "CustomPluginArn"
```

If the output of the `describe-custom-plugin` command, check the `CustomPluginState` value.
If it isn't `ACTIVE`, wait a few minutes, then run the command again. Keep checking until the status of the plugin is `ACTIVE`, then proceed to the next step to create the MSK connector.

11. Save the following JSON to a file and name the file `redshift-sink-connector.json`. Replace the placeholders with values that correspond to your scenario and resources.

```
{
    "ConnectorName": "my-example-connector",
    "KafkaConnectVersion": "2.7.1",
    "ServiceExecutionRoleArn": "arn:aws:iam::123456789000:role/my-example-role",
    "Plugins": [
        {
            "CustomPlugin": {
                "CustomPluginArn": "arn:aws:kafkaconnect:us-east-1:643570005906:custom-plugin/moe-redshift/fcf4f299-52b9-4ddd-83f2-74ab71445b60-2",
                "Revision": 1
            }
        }
    ],
    "Capacity": {
        "ProvisionedCapacity": {
            "WorkerCount": 2,
            "McuCount": 1
        }
    },
    "KafkaCluster": {
        "ApacheKafkaCluster": {
```
Debezium source connector

This example shows how to use the Debezium MySQL connector with a MySQL-compatible Amazon Aurora database as the source.

1. Go to the Debezium site and download the MySQL connector plugin.
2. Open the downloaded archive file and ZIP the `debezium-connector-mysql` folder.
3. Use the ZIP file from the previous step to create an MSK Connect plugin. For information on how to create plugins, see the section called “Plugins” (p. 57).
4. 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. 59).

```json

"Vpc": {
  "Subnets": [
    "subnet-abcd1234",
    "subnet-cdef0123",
    "subnet-abcd0101"
  ]
},
"KafkaClusterClientAuthentication": {"AuthenticationType": "NONE"},
"KafkaClusterEncryptionInTransit": {"EncryptionType": "PLAINTEXT"},
"LogDelivery": {
  "WorkerLogDelivery": {
    "CloudWatchLogs": {
      "LogGroup": "my-example-log-group",
      "Enabled": true
    }
  }
},
"ConnectorConfiguration": {
  "confluent.topic.replication.factor": "1",
  "connector.class": "io.confluent.connect.aws.redshift.RedshiftSinkConnector",
  "tasks.max": "2",
  "topics": "my-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": "my-redshift-user",
  "aws.redshift.password": "my-redshift-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"
}

12. To create the connector, run the following command.

```
aws kafkaconnect create-connector --cli-input-json file://redshift-sink-connector.json
```

Debezium source connector

This example shows how to use the Debezium MySQL connector with a MySQL-compatible Amazon Aurora database as the source.

1. Go to the Debezium site and download the MySQL connector plugin.
2. Open the downloaded archive file and ZIP the `debezium-connector-mysql` folder.
3. Use the ZIP file from the previous step to create an MSK Connect plugin. For information on how to create plugins, see the section called “Plugins” (p. 57).
4. 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. 59).
Debezium source connector

```json
{
  "ConnectorConfiguration": {
    "connector.class": "io.debezium.connector.mysql.MySqlConnector",
    "tasks.max": "1",
    "database.hostname": "aurora-database-writer-instance-endpoint",
    "database.port": "3306",
    "database.user": "your-database-user",
    "database.password": "your-secret-password",
    "database.server.id": "123456",
    "database.server.name": "logical name of the database server",
    "database.include.list": "the list of databases hosted by the specified server",
    "database.history.kafka.topic": "Apache Kafka topic used internally by Debezium to track database schema changes",
    "database.history.kafka.bootstrap.servers": "your cluster's 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": "your cluster's bootstrap servers string",
      "Vpc": {
        "Subnets": [
          "cluster's-subnet-1",
          "cluster's-subnet-2",
          "cluster's-subnet-3"
        ],
        "SecurityGroups": ["cluster's 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 the MSK Connect plugin that contains the code for the connector",
          "Revision": 1
        }
      }
    ],
    "KafkaClusterEncryptionInTransit": {"EncryptionType": "TLS"},
    "KafkaClusterClientAuthentication": {"AuthenticationType": "IAM"}
}
```
5. 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 fileb://connector-info.json
```

The following is an example of the output you get when you run the command successfully.

```
{
    "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](#).
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 (p. 74)
- Authentication and Authorization for Amazon MSK APIs (p. 78)
- Authentication and Authorization for Apache Kafka APIs (p. 85)
- Controlling Access to Apache ZooKeeper (p. 100)
- Logging (p. 102)
- Compliance Validation for Amazon Managed Streaming for Apache Kafka (p. 107)
- Resilience in Amazon Managed Streaming for Apache Kafka (p. 107)
- Infrastructure Security in Amazon Managed Streaming for Apache Kafka (p. 107)

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 (p. 75)
• How Do I Get Started with Encryption? (p. 76)

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 customer master key (CMK) that you want Amazon MSK to use to encrypt your data at rest. If you don't specify a CMK, Amazon MSK creates an AWS managed CMK for you and uses it on your behalf. For more information about CMKs, see Customer Master Keys (CMKs) 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.
Enabling encryption reduces performance by approximately 30%. However, the exact percentage depends on the configuration of your cluster and clients.

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 CMK or the AWS managed CMK 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 CMK. To determine which CMK 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 4: Create a Client Machine” (p. 11).
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. Go to the `bin` folder of the Apache Kafka installation on the client machine. To create a topic, run the following command, replacing `ZookeeperConnectString` with the value you obtained for `ZookeeperConnectString` in the previous step.

```
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. In the `bin` folder of the Apache Kafka installation on the client machine, run the following, replacing `BootstrapBrokerStringTls` with the value you obtained in the previous step. Leave this producer command running.

```
kafka-console-producer.sh --broker-list BootstrapBrokerStringTls --producer.config client.properties --topic TLSTestTopic
```

9. Open a new command window on the same client machine, go to the `bin` folder of the Apache Kafka installation, and run the following command to create a consumer.

```
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. 85).

Topics
• How Amazon MSK Works with IAM (p. 78)
• Amazon MSK Identity-Based Policy Examples (p. 81)
• Using Service-Linked Roles for Amazon MSK (p. 83)
• Troubleshooting Amazon MSK Identity and Access (p. 85)

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 (p. 78)
• Amazon MSK Resource-Based Policies (p. 80)
• AWS Managed Policies (p. 80)
• Authorization Based on Amazon MSK Tags (p. 80)
• Amazon MSK IAM Roles (p. 81)

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-dcbe-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 (*).
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. 81).

**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. 34).
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. 83).

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. 83).

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 (p. 81)
• Allow Users to View Their Own Permissions (p. 82)
• Accessing One Amazon MSK Cluster (p. 82)
• Accessing Amazon MSK Clusters Based on Tags (p. 83)

Policy Best Practices

Identity-based policies are very powerful. They 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 using AWS managed policies – To start using Amazon MSK quickly, use AWS managed
policies to give your employees the permissions they need. These policies are already available in
your account and are maintained and updated by AWS. For more information, see Get started using
permissions with AWS managed policies in the IAM User Guide.

• Grant least privilege – When you create custom policies, grant only the permissions required
to perform a task. Start with a minimum set of permissions and grant additional permissions as
necessary. Doing so is more secure than starting with permissions that are too lenient and then trying
to tighten them later. For more information, see Grant least privilege in the IAM User Guide.
• **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

• **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition 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.

```json
{
    "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.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "UpdateCluster",
            "Effect": "Allow",
            "Action": [
                "kafka:DescribeCluster",
                "kafka:GetBootstrapBrokers",
                "kafka:GetBootstrapBorkerNodes",
                "kafka:DescribeClusterAttributes",
                "kafka:UpdateCluster"
            ],
        }
    ]
}
```
"Effect":"Allow",
"Action": [  "kafka:Describe*",  "kafka:Get*",  "kafka:List*",  "kafka:Update*"]},

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": {  "kafka: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.
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 *`

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.

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 (p. 85)

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. 78).

Topics
- IAM access control (p. 85)
- Mutual TLS Authentication (p. 93)
- Username and password authentication with AWS Secrets Manager (p. 96)
- Apache Kafka ACLs (p. 99)

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. 104).

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. 100).

**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. 86)
- the section called “Configure clients for IAM access control” (p. 86)
- the section called “Create authorization policies” (p. 87)
- the section called “Get the bootstrap brokers for IAM access control” (p. 88)

### Create a cluster that uses IAM access control

You can use the AWS Management Console, the API, or the AWS CLI to create a cluster that uses IAM access control. You can’t enable IAM access control for an existing cluster.

#### Use the AWS Management Console to create a cluster that uses IAM access control

1. Open the Amazon MSK console at https://console.aws.amazon.com/msk/.
2. Choose **Create cluster**.
3. Choose **Create cluster with custom settings**.
4. In the **Authentication** section, choose **IAM access control**.
5. Complete the rest of the workflow for creating a cluster.

#### Use the API or the AWS CLI to create a cluster that uses IAM access control

- To create a cluster with IAM access control enabled, use the `CreateCluster` API or the `create-cluster` CLI command, and pass the following JSON for the `ClientAuthentication` parameter:
  ```json
  "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.
ssl.truststore.location=<PATH_TO_TRUST_STORE_FILE>
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:

```xml
<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. 86) 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. 88).

**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.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
    },
    {
      "Effect": "Allow",
```
IAM Access Control

"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. 91).

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. 87).

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.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Required actions</th>
<th>Required resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>kafka-cluster:Connect</td>
<td>Grants permission to connect and authenticate to the cluster.</td>
<td>None</td>
<td>cluster</td>
</tr>
<tr>
<td>kafka-cluster:DescribeCluster</td>
<td>Grants permission to describe various aspects of the cluster, equivalent to Apache Kafka's DESCRIBE CLUSTER ACL.</td>
<td>kafka-cluster:Connect</td>
<td>cluster</td>
</tr>
<tr>
<td>kafka-cluster:AlterCluster</td>
<td>Grants permission to alter various aspects of the cluster, equivalent to Apache Kafka's ALTER CLUSTER ACL.</td>
<td>kafka-cluster:Connect kafka-cluster:DescribeCluster</td>
<td>cluster</td>
</tr>
<tr>
<td>Action</td>
<td>Description</td>
<td>Required actions</td>
<td>Required resources</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-----------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>kafka-cluster:DescribeClusterDynamicConfiguration</td>
<td>Grants permission to describe the dynamic configuration of a cluster, equivalent to Apache Kafka's DESCRIBE_CONFIGS CLUSTER ACL.</td>
<td>kafka-cluster:Connect</td>
<td>cluster</td>
</tr>
<tr>
<td>kafka-cluster:AlterClusterDynamicConfiguration</td>
<td>Grants permission to alter the dynamic configuration of a cluster, equivalent to Apache Kafka's ALTER_CONFIGS CLUSTER ACL.</td>
<td>kafka-cluster:Connect</td>
<td>kafka-cluster:DescribeClusterDynamicConfiguration</td>
</tr>
<tr>
<td>kafka-cluster:WriteDataIdempotently</td>
<td>Grants permission to write data idempotently on a cluster, equivalent to Apache Kafka's IDEMPOTENT_WRITE CLUSTER ACL.</td>
<td>kafka-cluster:Connect</td>
<td>kafka-cluster:WriteData</td>
</tr>
<tr>
<td>kafka-cluster:CreateTopic</td>
<td>Grants permission to create topics on a cluster, equivalent to Apache Kafka's CREATE CLUSTER/TOPIC ACL.</td>
<td>kafka-cluster:Connect</td>
<td>topic</td>
</tr>
<tr>
<td>kafka-cluster:DescribeTopic</td>
<td>Grants permission to describe topics on a cluster, equivalent to Apache Kafka's DESCRIBE TOPIC ACL.</td>
<td>kafka-cluster:Connect</td>
<td>topic</td>
</tr>
<tr>
<td>kafka-cluster:AlterTopic</td>
<td>Grants permission to alter topics on a cluster, equivalent to Apache Kafka's ALTER TOPIC ACL.</td>
<td>kafka-cluster:Connect</td>
<td>kafka-cluster:DescribeTopic</td>
</tr>
<tr>
<td>kafka-cluster:DeleteTopic</td>
<td>Grants permission to delete topics on a cluster, equivalent to Apache Kafka's DELETE TOPIC ACL.</td>
<td>kafka-cluster:Connect</td>
<td>kafka-cluster:DescribeTopic</td>
</tr>
<tr>
<td>kafka-cluster:DescribeTopicDynamicConfiguration</td>
<td>Grants permission to describe the dynamic configuration of topics on a cluster, equivalent to Apache Kafka's DESCRIBE_CONFIGS TOPIC ACL.</td>
<td>kafka-cluster:Connect</td>
<td>topic</td>
</tr>
<tr>
<td>Action</td>
<td>Description</td>
<td>Required actions</td>
<td>Required resources</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>kafka-cluster:AlterTopicDynamicConfiguration</td>
<td>Grants permission to alter the dynamic configuration of topics on a cluster, equivalent to Apache Kafka's ALTER_CONFIGS_TOPIC ACL.</td>
<td>kafka-cluster:Connect</td>
<td>topic</td>
</tr>
<tr>
<td>kafka-cluster:ReadData</td>
<td>Grants permission to read data from topics on a cluster, equivalent to Apache Kafka's READ_TOPIC ACL.</td>
<td>kafka-cluster:Connect</td>
<td>topic</td>
</tr>
<tr>
<td>kafka-cluster:DescribeTopic</td>
<td>Grants permission to describe topics on a cluster, equivalent to Apache Kafka's DESCRIBE_TOPIC ACL.</td>
<td>kafka-cluster:Connect</td>
<td>topic</td>
</tr>
<tr>
<td>kafka-cluster:WriteData</td>
<td>Grants permission to write data to topics on a cluster, equivalent to Apache Kafka's WRITE_TOPIC ACL.</td>
<td>kafka-cluster:Connect</td>
<td>topic</td>
</tr>
<tr>
<td>kafka-cluster:DescribeGroup</td>
<td>Grants permission to describe groups on a cluster, equivalent to Apache Kafka's DESCRIBE_GROUP_ACL.</td>
<td>kafka-cluster:Connect</td>
<td>group</td>
</tr>
<tr>
<td>kafka-cluster:AlterGroup</td>
<td>Grants permission to alter groups on a cluster, equivalent to Apache Kafka's ALTER_GROUP_ACL.</td>
<td>kafka-cluster:Connect</td>
<td>group</td>
</tr>
<tr>
<td>kafka-cluster:DeleteGroup</td>
<td>Grants permission to delete groups on a cluster, equivalent to Apache Kafka's DELETE_GROUP_ACL.</td>
<td>kafka-cluster:Connect</td>
<td>group</td>
</tr>
<tr>
<td>kafka-cluster:DescribeTransactionalId</td>
<td>Grants permission to describe transactional IDs on a cluster, equivalent to Apache Kafka's DESCRIBE.Transactional_ID_ACL.</td>
<td>kafka-cluster:Connect</td>
<td>transactional-id</td>
</tr>
<tr>
<td>kafka-cluster:AlterTransactionalId</td>
<td>Grants permission to alter transactional IDs on a cluster, equivalent to Apache Kafka's WRITE_TRANSACTIONAL_ID_ACL.</td>
<td>kafka-cluster:Connect</td>
<td>transactional-id</td>
</tr>
</tbody>
</table>
You can use the asterisk (*) wildcard any number of times in an action after the colon. The following are examples.

- `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 transaction ID ARNs. To specify a resource in an authorization policy, use that resource's ARN.

<table>
<thead>
<tr>
<th>Resource</th>
<th>ARN format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td><code>arn:aws:kafka:region:account-id:cluster/cluster-name/cluster-uuid</code></td>
</tr>
<tr>
<td>Topic</td>
<td><code>arn:aws:kafka:region:account-id:topic/cluster-name/cluster-uuid/topic-name</code></td>
</tr>
<tr>
<td>Group</td>
<td><code>arn:aws:kafka:region:account-id:group/cluster-name/cluster-uuid/group-name</code></td>
</tr>
</tbody>
</table>

You can use the asterisk (*) wildcard any number of times anywhere in the part of the ARN that comes after `:cluster/`, `:topic/`, `:group/`, and `:transaction-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. 88).
**Note**
Actions are denied by default. You must explicitly allow every action that you want to authorize the client to perform.

<table>
<thead>
<tr>
<th>Use case</th>
<th>Required actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>kafka-cluster:*</td>
</tr>
<tr>
<td>Create a topic</td>
<td>kafka-cluster:Connect</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:CreateTopic</td>
</tr>
<tr>
<td>Produce data</td>
<td>kafka-cluster:Connect</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:DescribeTopic</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:WriteData</td>
</tr>
<tr>
<td>Consume data</td>
<td>kafka-cluster:Connect</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:DescribeTopic</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:DescribeGroup</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:AlterGroup</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:ReadData</td>
</tr>
<tr>
<td>Produce data idempotently</td>
<td>kafka-cluster:Connect</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:DescribeTopic</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:WriteData</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:WriteDataIdempotently</td>
</tr>
<tr>
<td>Produce data transactionally</td>
<td>kafka-cluster:Connect</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:DescribeTopic</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:WriteData</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:DescribeTransactionalId</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:AlterTransactionalId</td>
</tr>
<tr>
<td>Describe the configuration of a</td>
<td>kafka-cluster:Connect</td>
</tr>
<tr>
<td>cluster</td>
<td>kafka-cluster:DescribeClusterDynamicConfiguration</td>
</tr>
<tr>
<td>Update the configuration of a</td>
<td>kafka-cluster:Connect</td>
</tr>
<tr>
<td>cluster</td>
<td>kafka-cluster:DescribeClusterDynamicConfiguration</td>
</tr>
<tr>
<td></td>
<td>kafka-cluster:AlterClusterDynamicConfiguration</td>
</tr>
<tr>
<td>Describe the configuration of a</td>
<td>kafka-cluster:Connect</td>
</tr>
<tr>
<td>topic</td>
<td></td>
</tr>
</tbody>
</table>
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 ACM Private CA. For information about private CAs, see Creating and Managing a 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. 99).

This topic contains the following sections:

- To create a cluster that supports client authentication (p. 93)
- To set up a client to use authentication (p. 94)
- To produce and consume messages using authentication (p. 95)

To create a cluster that supports client authentication

This procedure shows you how to enable client authentication using a CA that is hosted by ACM.

1. Create a file named `clientauthinfo.json` with the following contents. Replace `Private-CA-ARN` with the ARN of your PCA.

   ```json
   {
     "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. 18).

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.
Amazon Managed Streaming for Apache Kafka Developer Guide
Mutual TLS Authentication

```json
{
   "EncryptionAtRest": {
      "DataVolumeKMSKeyId": "KMS-Key-ARN"
   },
   "EncryptionInTransit": {
      "InCluster": true,
      "ClientBroker": "TLS"
   }
}
```

For more information about encryption, see the section called “Encryption” (p. 75).

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.

```bash
```

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 4: Create a Client Machine” (p. 11) 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 5: Create a Topic” (p. 11).

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.

```bash
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.

```bash
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.

```bash
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.

```bash
```
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.

```bash
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.

9. Run the following command to get the certificate that ACM signed for you. Replace `Certificate-ARN` with the ARN you obtained from the response to the previous command.

```bash
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.

```bash
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`.

```properties
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.
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 your users’ credentials in Secrets Manager reduces the overhead of cluster authentication, including auditing, updating, and rotating credentials. Using Secrets Manager also lets you share user credentials across clusters.

This topic contains the following sections:
- How it Works (p. 96)
- Setting up SASL/SCRAM authentication for an Amazon MSK Cluster (p. 96)
- Working with Users (p. 98)
- Limitations (p. 99)

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 the 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 have 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:

```json
{
    "username": "alice",
    "password": "alice-secret"
}
```

• Record the ARN (Amazon Resource Name) value for your secret.

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.

The following example JSON input for the `BatchAssociateScramSecret` operation associates a secret with a cluster:

```json
{
    "secretArnList": [
    ]
}
```

**Connecting to your cluster with a username and password**

After you create your secret and associate it with your cluster, do the following to connect your client to the cluster:

1. Run the following command, replacing `ClusterArn` with the Amazon Resource Name (ARN) of your cluster:

   ```bash
   aws kafka describe-cluster --region us-west-2 --cluster-arn "ClusterArn"
   ```

   From the JSON result of the command, save the value associated with the string named "ZookeeperConnectString".

2. Create a topic in your cluster by running the following command in the `bin` directory in your client machine, replacing `ZookeeperConnectString` with the string you recorded in the previous command. You can specify a different name for the topic, but if you do, remember to use the name you chose in the rest of this procedure.

   ```bash
   ./kafka-topics.sh --create --zookeeper ZookeeperConnectString --replication-factor 3 -- partitions 1 --topic ExampleTopicName
   ```

3. On your client machine, create a JAAS configuration file with 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. Export this JAAS config file as a KAFKA_OPTS environment parameter with the following command:

```bash
export KAFKA_OPTS=-Djava.security.auth.login.config=<path-to-jaas-file>/users_jaas.conf
```

5. Copy the JDK key store file from your JVM to a ./tmp directory. For details about this step, see Step 6: Produce and Consume Data (p. 13) in the Getting Started (p. 5) tutorial.

6. Create a client properties file called `client_sasl.properties` with the following contents. This file defines the SASL mechanism and protocol.

```properties
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-512
ssl.truststore.location=<path-to-keystore-file>/kafka.client.truststore.jks
```

7. Run the following command, replacing `ClusterArn` with the Amazon Resource Name (ARN) of your cluster:

```bash
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
```

From the JSON result of the command, save the value associated with the string named "BootstrapBrokerStringSaslScram".

8. To produce to the topic you created, run the following command in the `bin` directory in your client machine, replacing `BootstrapBrokerStringSaslScram` with the value that you obtained when you ran the previous command.

```bash
./kafka-console-producer.sh --broker-list BootstrapBrokerStringSaslScram --topic ExampleTopicName --producer.config client_sasl.properties
```

9. To consume from the topic you created, run the following command in the `bin` directory in your client machine, replacing `BootstrapBrokerStringSaslScram` with the value that you obtained previously.

```bash
./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. AWS KMS requires the username and password data to be in the following format:

```json
{
  "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. 99).

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. 100).

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 a customer master key (CMK) with your Secret. You cannot use a Secret that uses the default Secrets Manager encryption key with Amazon MSK. For information about creating a CMK, see Creating symmetric CMKs.
• You can't use an asymmetric CMK 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.

Note
To avoid blocking brokers, don't set ACLs on CLUSTER_ACTIONS.

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. 20).

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).

```
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. Use the same `Distinguished-Name` you used when you created the private key.

```
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. Use the same `Distinguished-Name` you used when you created the private key.

```
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`.

## 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 (p. 100)
- Using TLS security with Apache ZooKeeper (p. 101)

### 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. 20). 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. Save the IP addresses of the Apache ZooKeeper nodes because you need them later in this procedure.

4. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

5. In the left pane, under NETWORK & SECURITY, choose Network Interfaces.

6. 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.

7. Select the check box at the beginning of the row that corresponds to the first network interface in the list.

8. 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.

9. 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.

10. 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.

11. 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. 76).
- Retrieve the TLS Apache ZooKeeper endpoints using the DescribeCluster operation.
- Create an Apache ZooKeeper configuration file for use with the following CLI commands: Config, ACL, and ZooKeeper Shell. You use the Apache Zookeeper config file with these commands using the --zk-tls-config-file parameter.

The following example shows a typical Apache ZooKeeper configuration file:

```plaintext
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"

Once you have configured 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:

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

Required permissions

For Amazon MSK to deliver broker logs to the destinations that you configure, ensure that the AmazonMSKFullAccess policy is attached to your IAM role. To stream broker logs to an S3 bucket, you also need the s3:PutBucketPolicy permission attached to your IAM role. 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 CMK 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 Customer Master Key (CMK), add the following to the key policy for your CMK 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 **Details** tab. Scroll down to the **Monitoring** 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](https://kafka.apache.org/changes.html#KIP-412). 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

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 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](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudTrail.html).

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](https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/ViewingEvents.html).

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 operations as events in CloudTrail log files. In addition, it logs the Apache Kafka actions listed under the section called “Semantics of actions and resources” (p. 88).
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%2Fexample%2Fcluster%2F01234567-abcd-0123-abcd-abcd0123efa-2"
      },
      "responseElements": null,
      "requestID": "bd83f636-fdb5-0123-157e2f6b2be",
      "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%2Fexample%2Fcluster%2F01234567-abcd-0123-abcd-abcd0123efa-2"
      },
      "responseElements": null,
      "requestID": "bd83f636-fdb5-0123-157e2f6b2be",
      "eventID": "60052aba-0123-4511-bcde-3e18dbd42aa4",
      "readOnly": true,
      "eventType": "AwsApiCall",
      "recipientAccountId": "012345678901"
    }
  ]
}
The following example shows a CloudTrail log entry that demonstrates the kafka-cluster:CreateTopic action.

```json
{  
  "eventVersion": "1.08",  
  "userIdentity": {  
    "type": "IAMUser",  
    "principalId": "ABCDEFGH1IJKLMN2P34Q5",  
    "arn": "arn:aws:iam::111122223333:user/Admin",  
    "accountId": "111122223333",  
    "accessKeyId": "CDEFAB1C2UUUUU9AB4TT",  
    "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",  
  },  
  "responseElements": null,  
  "requestID": "e7c5e49f-6a6ac-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.

In addition to the AWS global infrastructure, Amazon MSK offers several features to help support your data resiliency and backup needs.

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

To connect to your Amazon MSK cluster from a client that's in the same Amazon 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 an MSK cluster from outside its 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

Amazon Direct Connect links your internal network to an AWS Direct Connect location 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.

   ```bash
   aws kafka describe-cluster --region us-east-1 --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.
- To communicate with brokers by using SASL/SCRAM, use port 9096.
- To communicate with brokers in a cluster that is set up to use the section called "IAM Access Control" (p. 85), use port 9098.
- 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:

   ```
   ./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.inflight.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 in-flight 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.
MirrorMaker 2.* Advantages

- Increases batch size.
- Tuning 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 MSK gathers Apache Kafka metrics and sends them to Amazon CloudWatch where you can view them. For more information about Apache Kafka metrics, including the ones that Amazon MSK surfaces, see Monitoring in the Apache Kafka documentation.

You can also monitor your MSK cluster with Prometheus, an open-source monitoring application. For information about Prometheus, see Overview in the Prometheus documentation. To learn how to monitor your cluster with Prometheus, see the section called "Open Monitoring with Prometheus" (p. 126).

Topics
- Amazon MSK Metrics for Monitoring with CloudWatch (p. 114)
- Viewing Amazon MSK Metrics Using CloudWatch (p. 122)
- Consumer-Lag Monitoring (p. 123)
- Open Monitoring with Prometheus (p. 126)

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>When Visible</th>
<th>Dimensions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveControllerCount</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name</td>
<td>Only one controller per cluster should be active at any given time.</td>
</tr>
<tr>
<td>BurstBalance</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The remaining balance of input-output burst credits for EBS volumes in the cluster. Use it to investigate latency or decreased throughput.</td>
</tr>
<tr>
<td>BytesInPerSec</td>
<td>After you create a topic.</td>
<td>Cluster Name, Broker ID, Topic</td>
<td>The number of bytes per second received from clients. This metric is available per broker and also per topic.</td>
</tr>
<tr>
<td>Name</td>
<td>When Visible</td>
<td>Dimensions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BytesOutPerSec</td>
<td>After you create a topic.</td>
<td>Cluster Name, Broker ID, Topic</td>
<td>The number of bytes per second sent to clients. This metric is available per broker and also per topic.</td>
</tr>
<tr>
<td>ConnectionCount</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number active Apache Kafka connections.</td>
</tr>
<tr>
<td>CPUCreditBalance</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>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.</td>
</tr>
<tr>
<td>CpuIdle</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The percentage of CPU idle time.</td>
</tr>
<tr>
<td>CpuSystem</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The percentage of CPU in kernel space.</td>
</tr>
<tr>
<td>CpuUser</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The percentage of CPU in user space.</td>
</tr>
<tr>
<td>GlobalPartitionCount</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name</td>
<td>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.</td>
</tr>
<tr>
<td>GlobalTopicCount</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name</td>
<td>Total number of topics across all brokers in the cluster.</td>
</tr>
<tr>
<td>EstimatedMaxTimeLag</td>
<td>After consumer group consumes from a topic.</td>
<td>Consumer Group, Topic</td>
<td>Time estimate (in seconds) to drain MaxOffsetLag.</td>
</tr>
<tr>
<td>Name</td>
<td>When Visible</td>
<td>Dimension</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-------------------------------------</td>
<td>-------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KafkaAppLogsDiskUsed</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The percentage of disk space used for application logs.</td>
</tr>
<tr>
<td>KafkaDataLogsDiskUsed (Cluster Name, Broker ID dimension)</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The percentage of disk space used for data logs.</td>
</tr>
<tr>
<td>KafkaDataLogsDiskUsed (Cluster Name dimension)</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name</td>
<td>The percentage of disk space used for data logs.</td>
</tr>
<tr>
<td>LeaderCount</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The total number of leaders of partitions per broker, not including replicas.</td>
</tr>
<tr>
<td>MaxOffsetLag</td>
<td>After consumer group consumes from a topic.</td>
<td>Consumer Group, Topic</td>
<td>The maximum offset lag across all partitions in a topic.</td>
</tr>
<tr>
<td>MemoryBuffered</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The size in bytes of buffered memory for the broker.</td>
</tr>
<tr>
<td>MemoryCached</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The size in bytes of cached memory for the broker.</td>
</tr>
<tr>
<td>MemoryFree</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The size in bytes of memory that is free and available for the broker.</td>
</tr>
<tr>
<td>MemoryHeapAfterGC</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The percentage of total heap memory available after garbage collection.</td>
</tr>
<tr>
<td>MemoryUsed</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The size in bytes of memory that is in use for the broker.</td>
</tr>
<tr>
<td>MessagesInPerSec</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number of incoming messages per second for the broker.</td>
</tr>
<tr>
<td>NetworkRxDropped</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number of dropped receive packages.</td>
</tr>
<tr>
<td>Name</td>
<td>When Visible</td>
<td>Dimension</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NetworkRxErrors</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number of network receive errors for the broker.</td>
</tr>
<tr>
<td>NetworkRxPackets</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number of packets received by the broker.</td>
</tr>
<tr>
<td>NetworkTxDropped</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number of dropped transmit packages.</td>
</tr>
<tr>
<td>NetworkTxErrors</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number of network transmit errors for the broker.</td>
</tr>
<tr>
<td>NetworkTxPackets</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number of packets transmitted by the broker.</td>
</tr>
<tr>
<td>OfflinePartitionsCount</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name</td>
<td>Total number of partitions that are offline in the cluster.</td>
</tr>
<tr>
<td>PartitionCount</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The total number of topic partitions per broker, including replicas.</td>
</tr>
<tr>
<td>ProduceTotalTimeMsMean</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The mean produce time in milliseconds.</td>
</tr>
<tr>
<td>RequestBytesMean</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The mean number of request bytes for the broker.</td>
</tr>
<tr>
<td>RequestTime</td>
<td>After request throttling is applied.</td>
<td>Cluster Name, Broker ID</td>
<td>The average time in milliseconds spent in broker network and I/O threads to process requests.</td>
</tr>
<tr>
<td>RootDiskUsed</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The percentage of the root disk used by the broker.</td>
</tr>
<tr>
<td>SumOffsetLag</td>
<td>After consumer group consumes from a topic.</td>
<td>Consumer Group, Topic</td>
<td>The aggregated offset lag for all the partitions in a topic.</td>
</tr>
<tr>
<td>Name</td>
<td>When Visible</td>
<td>Dimensions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SwapFree</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The size in bytes of swap memory that is available for the broker.</td>
</tr>
<tr>
<td>SwapUsed</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The size in bytes of swap memory that is in use for the broker.</td>
</tr>
<tr>
<td>TrafficShaping</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>High-level metrics indicating the number of packets dropped due to exceeding network allocations. Finer detail is available with PER_BROKER metrics.</td>
</tr>
<tr>
<td>UnderMinIsrPartitionCount</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number of under minIsr partitions for the broker.</td>
</tr>
<tr>
<td>UnderReplicatedPartitions</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The number of under-replicated partitions for the broker.</td>
</tr>
<tr>
<td>ZooKeeperRequestLatencyMsMean</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>The mean latency in milliseconds for Apache ZooKeeper requests from broker.</td>
</tr>
<tr>
<td>ZooKeeperSessionState</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Cluster Name, Broker ID</td>
<td>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'.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>When Visible</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BwInAllowanceExceeded</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of packets shaped because the inbound aggregate bandwidth exceeded the maximum for the broker.</td>
</tr>
<tr>
<td>Name</td>
<td>When Visible</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BwOutAllowanceExceeded</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of packets shaped because the outbound aggregate bandwidth exceeded the maximum for the broker.</td>
</tr>
<tr>
<td>ConnTrackAllowanceExceeded</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>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.</td>
</tr>
<tr>
<td>ConnectionCloseRate</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of connections closed per second per listener. This number is aggregated per listener and filtered for the client listeners.</td>
</tr>
<tr>
<td>ConnectionCreationRate</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of new connections established per second per listener. This number is aggregated per listener and filtered for the client listeners.</td>
</tr>
<tr>
<td>CpuCreditUsage</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>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.</td>
</tr>
<tr>
<td>FetchConsumerLocalTimeMsMean</td>
<td>After there's a producer/consumer.</td>
<td>The mean time in milliseconds that the consumer request is processed at the leader.</td>
</tr>
<tr>
<td>FetchConsumerRequestQueueTimeMsMean</td>
<td>After there's a producer/consumer.</td>
<td>The mean time in milliseconds that the consumer request waits in the request queue.</td>
</tr>
<tr>
<td>FetchConsumerResponseQueueTimeMsMean</td>
<td>After there's a producer/consumer.</td>
<td>The mean time in milliseconds that the consumer request waits in the response queue.</td>
</tr>
<tr>
<td>FetchConsumerResponseSendTimeMsMean</td>
<td>After there's a producer/consumer.</td>
<td>The mean time in milliseconds for the consumer to send a response.</td>
</tr>
<tr>
<td>FetchConsumerTotalTimeMsMean</td>
<td>After there's a producer/consumer.</td>
<td>The mean total time in milliseconds that consumers spend on fetching data from the broker.</td>
</tr>
<tr>
<td>FetchFollowerLocalTimeMsMean</td>
<td>After there's a producer/consumer.</td>
<td>The mean time in milliseconds that the follower request is processed at the leader.</td>
</tr>
<tr>
<td>FetchFollowerRequestQueueTimeMsMean</td>
<td>After there's a producer/consumer.</td>
<td>The mean time in milliseconds that the follower request waits in the request queue.</td>
</tr>
<tr>
<td>Name</td>
<td>When Visible</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FetchFollowerResponseQueueTimeMsMean</td>
<td>After there’s a producer/consumer.</td>
<td>The mean time in milliseconds that the follower request waits in the response queue.</td>
</tr>
<tr>
<td>FetchFollowerResponseSendTimeMsMean</td>
<td>After there’s a producer/consumer.</td>
<td>The mean time in milliseconds for the follower to send a response.</td>
</tr>
<tr>
<td>FetchFollowerTotalTimeMsMean</td>
<td>After there’s a producer/consumer.</td>
<td>The mean total time in milliseconds that followers spend on fetching data from the broker.</td>
</tr>
<tr>
<td>FetchMessageConversionsPerSec</td>
<td>After you create a topic.</td>
<td>The number of fetch message conversions per second for the broker.</td>
</tr>
<tr>
<td>FetchThrottleByteRate</td>
<td>After bandwidth throttling is applied.</td>
<td>The number of throttled bytes per second.</td>
</tr>
<tr>
<td>FetchThrottleQueueSize</td>
<td>After bandwidth throttling is applied.</td>
<td>The number of messages in the throttle queue.</td>
</tr>
<tr>
<td>FetchThrottleTime</td>
<td>After bandwidth throttling is applied.</td>
<td>The average fetch throttle time in milliseconds.</td>
</tr>
<tr>
<td>NetworkProcessorAvgIdlePercent</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The average percentage of the time the network processors are idle.</td>
</tr>
<tr>
<td>PpsAllowanceExceeded</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of packets shaped because the bidirectional PPS exceeded the maximum for the broker.</td>
</tr>
<tr>
<td>ProduceLocalTimeMsMean</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The mean time in milliseconds for the follower to send a response.</td>
</tr>
<tr>
<td>ProduceMessageConversionsPerSec</td>
<td>After you create a topic.</td>
<td>The number of produce message conversions per second for the broker.</td>
</tr>
<tr>
<td>ProduceMessageConversionsTimeMsMean</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The mean time in milliseconds spent on message format conversions.</td>
</tr>
<tr>
<td>ProduceRequestQueueTimeMsMean</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The mean time in milliseconds that request messages spend in the queue.</td>
</tr>
<tr>
<td>ProduceResponseQueueTimeMsMean</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The mean time in milliseconds that response messages spend in the queue.</td>
</tr>
<tr>
<td>ProduceResponseSendTimeMsMean</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The mean time in milliseconds spent on sending response messages.</td>
</tr>
<tr>
<td>ProduceThrottleByteRate</td>
<td>After bandwidth throttling is applied.</td>
<td>The number of throttled bytes per second.</td>
</tr>
<tr>
<td>ProduceThrottleQueueSize</td>
<td>After bandwidth throttling is applied.</td>
<td>The number of messages in the throttle queue.</td>
</tr>
<tr>
<td>ProduceThrottleTime</td>
<td>After bandwidth throttling is applied.</td>
<td>The average produce throttle time in milliseconds.</td>
</tr>
<tr>
<td>Name</td>
<td>When Visible</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ProduceTotalTimeMsMean</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The mean produce time in milliseconds.</td>
</tr>
<tr>
<td>ReplicationBytesInPerSec</td>
<td>After you create a topic.</td>
<td>The number of bytes per second received from other brokers.</td>
</tr>
<tr>
<td>ReplicationBytesOutPerSec</td>
<td>After you create a topic.</td>
<td>The number of bytes per second sent to other brokers.</td>
</tr>
<tr>
<td>RequestExemptFromThrottleTime</td>
<td>After request throttling is applied.</td>
<td>The average time in milliseconds spent in broker network and I/O threads to process requests that are exempt from throttling.</td>
</tr>
<tr>
<td>RequestHandlerAvgIdlePercent</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The average percentage of the time the request handler threads are idle.</td>
</tr>
<tr>
<td>RequestThrottleQueueSize</td>
<td>After request throttling is applied.</td>
<td>The number of messages in the throttle queue.</td>
</tr>
<tr>
<td>RequestThrottleTime</td>
<td>After request throttling is applied.</td>
<td>The average request throttle time in milliseconds.</td>
</tr>
<tr>
<td>TcpConnections</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Shows number of incoming and outgoing TCP segments with the SYN flag set.</td>
</tr>
<tr>
<td>TrafficBytes</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>Shows network traffic in overall bytes between clients (producers and consumers) and brokers. Traffic between brokers isn't reported.</td>
</tr>
<tr>
<td>VolumeQueueLength</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of read and write operation requests waiting to be completed in a specified time period.</td>
</tr>
<tr>
<td>VolumeReadBytes</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of bytes read in a specified time period.</td>
</tr>
<tr>
<td>VolumeReadOps</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of read operations in a specified time period.</td>
</tr>
<tr>
<td>VolumeTotalReadTime</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The total number of seconds spent by all read operations that completed in a specified time period.</td>
</tr>
<tr>
<td>VolumeTotalWriteTime</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The total number of seconds spent by all write operations that completed in a specified time period.</td>
</tr>
<tr>
<td>VolumeWriteBytes</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of bytes written in a specified time period.</td>
</tr>
<tr>
<td>VolumeWriteOps</td>
<td>After the cluster gets to the ACTIVE state.</td>
<td>The number of write operations in a specified time period.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>When Visible</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FetchMessageConversionsPerSec</td>
<td>After you create a topic.</td>
<td>The number of fetched messages converted per second.</td>
</tr>
<tr>
<td>MessagesInPerSec</td>
<td>After you create a topic.</td>
<td>The number of messages received per second.</td>
</tr>
<tr>
<td>ProduceMessageConversionsPerSec</td>
<td>After you create a topic.</td>
<td>The number of conversions per second for produced messages.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>When Visible</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EstimatedTimeLag</td>
<td>After consumer group consumes from a topic.</td>
<td>Time estimate (in seconds) to drain the partition offset lag.</td>
</tr>
<tr>
<td>OffsetLag</td>
<td>After consumer group consumes from a topic.</td>
<td>Partition-level consumer lag in number of offsets.</td>
</tr>
</tbody>
</table>

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, open monitoring with Prometheus, or Burrow.

Consumer-Lag Metrics for CloudWatch and for 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. 114).

Consumer-Lag Monitoring with Burrow

Burrow is a monitoring companion for Apache Kafka that provides consumer-lag checking. Burrow has a modular design that includes the following subsystems:

- Clusters run an Apache Kafka client that periodically updates topic lists and the current HEAD offset (the most recent offset) for every partition.
- Consumers fetch information about consumer groups from a repository. This repository can be an Apache Kafka cluster (consuming the __consumer_offsets topic), ZooKeeper, or some other repository.
- The storage subsystem stores all of this information in Burrow.
- The evaluator subsystem retrieves information from the storage subsystem for a specific consumer group and calculates the status of that group. This follows the consumer lag evaluation rules.
- The notifier subsystem requests status on consumer groups according to a configured interval and sends out notifications (Email, HTTP, or some other method) for groups that meet the configured criteria.
The HTTP Server subsystem provides an API interface to Burrow for fetching information about clusters and consumers.

For more information about Burrow, see Burrow - Kafka Consumer Lag Checking.

**Important**
Make sure that Burrow is compatible with the version of Apache Kafka that you are using for your MSK cluster.

**To set up and use Burrow with Amazon MSK**

Follow this if you use plaintext communication. For TLS, see the next procedure, as well.

1. Create an MSK cluster and launch a client machine in the same VPC as the cluster. For example, you can follow the instructions at Getting Started (p. 5).
2. Run the following command on the EC2 instance that serves as your client machine.

   ```bash
   sudo yum install go
   ```

3. Run the following command on the client machine to get the Burrow project.

   ```bash
   go get github.com/linkedin/Burrow
   ```

4. Run the following command to install dep. It installs it in the `/home/ec2-user/go/bin/dep` folder.

   ```bash
   curl https://raw.githubusercontent.com/golang/dep/master/install.sh | sh
   ```

5. Go to the `/home/ec2-user/go/src/github.com/linkedin/Burrow` folder and run the following command.

   ```bash
   /home/ec2-user/go/bin/dep ensure
   ```

6. Run the following command in the same folder.

   ```bash
   go install
   ```

7. Open the `/home/ec2-user/go/src/github.com/linkedin/Burrow/config/burrow.toml` configuration file for editing. In the following sections of the configuration file, replace the placeholders with the name of your MSK cluster, the host:port pairs for your ZooKeeper servers, and your bootstrap brokers.

   To get your ZooKeeper host:port pairs, describe your MSK cluster and look for the value of ZookeeperConnectionString. See the section called “Getting the Apache ZooKeeper Connection String” (p. 20).

   To get your bootstrap brokers, see the section called “Getting the Bootstrap Brokers” (p. 21).

   Follow the formatting shown below when you edit the configuration file.

   ```toml
   [zookeeper]
   servers=["ZooKeeper-host-port-pair-1", "ZooKeeper-host-port-pair-2", "ZooKeeper-host-port-pair-3"]
   timeout=6
   root-path="/burrow"

   [client-profile.test]
   client-id="burrow-test"
   ```
8. In the `go/bin` folder run the following command.

```
./Burrow --config-dir /home/ec2-user/go/src/github.com/linkedin/Burrow/config
```


10. You can use the following command to test your setup.

```
curl -XGET 'HTTP://your-localhost-ip:8000/v3/kafka'
```

11. For all of the supported HTTP requests and links, see Burrow HTTP Endpoint.

### To use Burrow with TLS

In addition to the previous procedure, see the following steps if you are using TLS communication.

1. Run the following command.

```
sudo yum install java-1.8.0-openjdk-devel -y
```

2. Run the following command after you adjust the paths as necessary.

```
find /usr/lib/jvm/ -name "cacerts" -exec cp {} /tmp/kafka.client.truststore.jks \;
```

3. In the next step you use the `keytool` command, which asks for a password. The default password is `changeit`. We recommend that you run the following command to change the password before you proceed to the next step.

```
keytool -keystore /tmp/kafka.client.truststore.jks -storepass changeit -storepasswd -new Password
```

4. Run the following command.

```
keytool --list -rfc -keystore /tmp/kafka.client.truststore.jks >/tmp/truststore.pem
```

You need `truststore.pem` for the `burrow.toml` file that's described later in this procedure.

5. To generate the certfile and the keyfile, use the code at Managing Client Certificates for Mutual Authentication with Amazon MSK. You need the `pem` flag.

6. Set up your `burrow.toml` file like the following example. You can have multiple cluster and consumer sections to monitor multiple MSK clusters using one burrow cluster. You can also adjust
Open Monitoring with Prometheus

You can monitor your MSK cluster with Prometheus, an open-source monitoring system for time-series metric data. 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 the Details page for the cluster.
3. On the Details 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](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`.

   ```yaml
   # file: prometheus.yml
   # my global config
   global:
     scrape_interval: 10s
   
   # 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.

   ```json
   [
     {
       "labels": {
         "job": "jmx"
       },
       "targets": [
         "broker_dns_1:11001",
         "broker_dns_2:11001",
         ...
         "broker_dns_N:11001"
       ]
     },
     {
       "labels": {
         "job": "node"
       },
       "targets": [
   ```
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.
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. 126).
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
```

130
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.PrometheusMetricSampler

# 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.

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

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.

```bash
./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 has the following quota:

- Up to 90 brokers per account and 30 brokers per cluster. To request a higher limit, create a support case.
- A minimum of 1 GiB of storage per broker.
- A maximum of 16384 GiB of storage per broker.
- Up to 100 configurations per account. To request a higher quota, create a support case.
- 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 "Number of partitions per broker" (p. 145).
- A cluster that uses the section called "IAM Access Control" (p. 85) can have up to 3000 TCP connections per broker at any given time.
- A cluster that uses the section called "IAM Access Control" (p. 85) can have up to 20 TCP connections per broker per second for all broker types, except for brokers of type kafka.t3.small, which are limited to 1 TCP connection per second. If you want a client to retry connections after 1 second, set the configuration parameter reconnect.backoff.ms to 1000. You can also set this parameter to higher values. For more information, see reconnect.backoff.ms in the Apache Kafka documentation.

MSK Connect has the following 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.

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. 85), a resource is an entity to which you can allow or deny access, as defined in the the section called "Resources" (p. 91) 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 (p. 134)
- Updating the Apache Kafka version (p. 136)

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 2.8.0 (p. 134)
- Apache Kafka version 2.7.1 (p. 134)
- Apache Kafka version 2.6.2 [Recommended] (p. 134)
- Apache Kafka version 2.7.0 (p. 134)
- Apache Kafka version 2.6.1 (p. 135)
- Apache Kafka version 2.6.0 (p. 135)
- Apache Kafka version 2.5.1 (p. 135)
- Amazon MSK bug-fix version 2.4.1.1 (p. 135)
- Apache Kafka version 2.4.1 (use 2.4.1.1 instead) (p. 135)
- Apache Kafka version 2.3.1 (p. 136)
- Apache Kafka version 2.2.1 (p. 136)
- Apache Kafka version 1.1.1 (for existing clusters only) (p. 136)

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 2.7.1

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

Apache Kafka version 2.6.2 [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. 101).

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. 136).

To work around this issue without upgrading the cluster to version 2.4.1.1, see the Consumer group stuck in PreparingRebalance state (p. 139) section of the ??? (p. 139) 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. 135) 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. 122).

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. 36).

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. 145).

Important
You can't update the Apache Kafka version for an MSK cluster that exceeds the limits described in the section called “Number of partitions per broker” (p. 145).
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 Details tab choose Upgrade the Apache Kafka version.

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).

   ```bash
   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.

   ```json
   {
     "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 KTVPDKIXX0DER.

   ```bash
   ```

   The output of the previous command looks like the following JSON.

   ```json
   {
     "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/abcdefab-1234-abcd-5678-cdef013ab01-2",
   }
   ```
3. To get the result of the update-cluster-kafka-version operation, run the following command, replacing \texttt{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 \texttt{ClusterOperationArn}
```

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

```json
{
  "ClusterOperationInfo": {
    "ClientRequestId": "62cd41d2-1206-4ebf-85a8-dbb2ba0fe259",
    "ClusterArn": "arn:aws:kafka:us-east-1:012345678012:cluster/exampleClusterName/abcdefab-abcd-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-abcdef-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 \texttt{UPDATE\_IN\_PROGRESS}, wait a while, then run the describe-cluster-operation command again. When the operation is complete, the value of OperationState becomes \texttt{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 \texttt{GetCompatibleKafkaVersions} operation to get a list of the Apache Kafka versions to which you can update the cluster.
2. Invoke the \texttt{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 the Amazon MSK forum.

Topics
- Consumer group stuck in PreparingRebalance state (p. 139)
- Error delivering broker logs to Amazon CloudWatch Logs (p. 140)
- No default security group (p. 140)
- Cluster appears stuck in the CREATING state (p. 140)
- Cluster state goes from CREATING to FAILED (p. 141)
- Cluster state is ACTIVE but producers cannot send data or consumers cannot receive data (p. 141)
- AWS CLI doesn't recognize Amazon MSK (p. 141)
- Partitions go offline or replicas are out of sync (p. 141)
- Disk space is running low (p. 141)
- Memory is running low (p. 141)
- Producer Gets NotLeaderForPartitionException (p. 141)
- Under-Replicated Partitions (URP) greater than zero (p. 142)
- Cluster has topics called __amazon_msk_canary and __amazon_msk_canary_state (p. 142)
- Partition replication fails (p. 142)
- Networking issues (p. 142)

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. 135), 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. 136).

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. 139), or to Identify and Reboot (p. 140) 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.

```json
{"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 4: Create a Client Machine” (p. 11).

- 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. 111).

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. 16).

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. 141).

Disk space is running low

See the following best practices for managing disk space: the section called “Monitor disk space” (p. 147) and the section called “Adjust data retention parameters” (p. 147).

Memory is running low

If you see the MemoryUsed metric running high or MemoryFree running low, that doesn't mean there's a problem. Apache Kafka is designed to use as much memory as possible, and it manages it optimally.

Producer Gets NotLeaderForPartitionException

This is often a transient error. Set the producer's retries configuration parameter to a value that's higher than its current value.
Under-Replicated Partitions (URP) greater than zero

The UnderReplicatedPartitions metric is an important one to monitor. In a healthy MSK cluster, this metric has the value 0. If it's greater than zero, that might be for one of the following reasons.

- If UnderReplicatedPartitions is spiky, the issue might be that the cluster isn't provisioned at the right size to handle incoming and outgoing traffic. See the section called “Right-size your cluster” (p. 145).
- 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.

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. 20) 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.
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.

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

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.

**Note**
Clients must use a private connection to produce and consume data to and from an MSK cluster. Amazon MSK doesn’t support public endpoints.

### 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.
Best Practices

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

Right-size your cluster

When you create an MSK cluster, you specify the type and number of brokers.

Number of partitions per broker

The following table shows the recommended maximum number of partitions (including leader and follower replicas) per broker. However, the number of partitions per broker is affected by use case and configuration. 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. 16).

<table>
<thead>
<tr>
<th>Broker type</th>
<th>Maximum number of partitions (including leader and follower replicas) per broker</th>
</tr>
</thead>
<tbody>
<tr>
<td>kafka.t3.small</td>
<td>300</td>
</tr>
<tr>
<td>kafka.m5.large or kafka.m5.xlarge</td>
<td>1000</td>
</tr>
<tr>
<td>kafka.m5.2xlarge</td>
<td>2000</td>
</tr>
<tr>
<td>kafka.m5.4xlarge, kafka.m5.8xlarge, kafka.m5.12xlarge, kafka.m5.16xlarge, or kafka.m5.24xlarge</td>
<td>4000</td>
</tr>
</tbody>
</table>

For guidance on choosing the number of partitions, see Apache Kafka Supports 200K Partitions Per 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. This spreadsheet was the result of running a test workload with three producers and three consumers, and ensuring that P99 write latencies were below 100 ms. This might not reflect your workload or performance expectations. Therefore, we recommend that you test your workloads after provisioning the cluster.

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.
Monitor CPU usage

Amazon Managed Streaming for Apache Kafka Developer Guide

Monitor CPU usage

- Ensure that the replication factor (RF) is at least 2 for two-AZ clusters and at least 3 for three-AZ clusters. An RF of 1 can lead to offline partitions during a rolling update.
- 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 multiple brokers. 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).

Amazon MSK strongly recommends that you maintain the total CPU utilization for your brokers under 60%. Total CPU utilization is the sum of the CpuUser and CpuSystem metrics. 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 CPU metric that is the sum of CpuUser and CpuSystem. Set an alarm that gets triggered when the composite CPU metric reaches a P95 of 60%. When the 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 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. 23). You can also manually increase broker storage as described in the section called “Manual scaling” (p. 24).
- Reduce the message retention period or log size. For information on how to do that, see the section called “Adjust data retention parameters” (p. 147).
- 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. 114).

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. 36).

You can also specify retention parameters at the topic level:

- To specify a retention time period per topic, use the following command.

  ```bash
  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.

  ```bash
  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.

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. 16).
Enable in-transit encryption

For information about encryption in transit and how to enable it, see the section called “Encryption in Transit” (p. 75).

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. 29). For information about the partition reassignment tool, see Expanding your cluster in the Apache Kafka documentation.
## Document History for Amazon MSK Developer Guide

The following table describes the important changes to the Amazon MSK Developer Guide.

**Latest documentation update:** September 30, 2020

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK Connect</td>
<td>MSK Connect is a new feature that you can use to create and manage Apache Kafka connectors. For more information, see MSK Connect (p. 48).</td>
<td>2021-09-16</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.7.1</td>
<td>Amazon MSK now supports Apache Kafka version 2.7.1. For more information, see Supported Apache Kafka versions (p. 134).</td>
<td>2021-05-25</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.8.0</td>
<td>Amazon MSK now supports Apache Kafka version 2.8.0. For more information, see Supported Apache Kafka versions (p. 134).</td>
<td>2021-04-28</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.6.2</td>
<td>Amazon MSK now supports Apache Kafka version 2.6.2. For more information, see Supported Apache Kafka versions (p. 134).</td>
<td>2021-04-28</td>
</tr>
<tr>
<td>Support for Updating Broker Type</td>
<td>You can now change the broker type for an existing cluster. For more information, see Updating the broker type (p. 25).</td>
<td>2021-01-21</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.6.1</td>
<td>Amazon MSK now supports Apache Kafka version 2.6.1. For more information, see Supported Apache Kafka versions (p. 134).</td>
<td>2021-01-19</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.7.0</td>
<td>Amazon MSK now supports Apache Kafka version 2.7.0. For more information, see Supported Apache Kafka versions (p. 134).</td>
<td>2020-12-29</td>
</tr>
<tr>
<td>No New Clusters with Apache Kafka Version 1.1.1</td>
<td>You can no longer create a new Amazon MSK cluster with Apache Kafka version 1.1.1.</td>
<td>2020-11-24</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Change</td>
<td>However, if you have existing MSK clusters that are running Apache Kafka version 1.1.1, you can continue using all of the currently-supported features on those existing clusters. For more information, see Apache Kafka Versions (p. 134).</td>
<td></td>
</tr>
<tr>
<td>Consumer-Lag Metrics</td>
<td>Amazon MSK now provides metrics that you can use to monitor consumer lag. For more information, see Monitoring an Amazon MSK Cluster (p. 114).</td>
<td>2020-11-23</td>
</tr>
<tr>
<td>Support for Cruise Control</td>
<td>Amazon MSK now supports LinkedIn's Cruise Control. For more information, see Using LinkedIn's Cruise Control for Apache Kafka with Amazon MSK (p. 130).</td>
<td>2020-11-17</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.6.0</td>
<td>Amazon MSK now supports Apache Kafka version 2.6.0. For more information, see Supported Apache Kafka versions (p. 134).</td>
<td>2020-10-21</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.5.1</td>
<td>Amazon MSK now supports Apache Kafka version 2.5.1. With Apache Kafka version 2.5.1, Amazon MSK supports encryption in transit between clients and ZooKeeper endpoints. For more information, see Supported Apache Kafka versions (p. 134).</td>
<td>2020-09-30</td>
</tr>
<tr>
<td>Application Auto-Expansion</td>
<td>You can configure Amazon Managed Streaming for Apache Kafka to automatically expand your cluster's storage in response to increased usage. For more information, see Automatic scaling (p. 23).</td>
<td>2020-09-30</td>
</tr>
<tr>
<td>Support for Username and Password Security</td>
<td>Amazon MSK now supports logging into clusters using a username and password. Amazon MSK stores credentials in AWS Secrets Manager. For more information, see SASL/SCRAM Authentication (p. 96).</td>
<td>2020-09-17</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>Support for Upgrading the Apache Kafka Version of an Amazon MSK Cluster</td>
<td>You can now upgrade the Apache Kafka version of an existing MSK cluster.</td>
<td>2020-05-28</td>
</tr>
<tr>
<td>Support for T3.small Broker Nodes</td>
<td>Amazon MSK now supports creating clusters with brokers of Amazon EC2 type T3.small.</td>
<td>2020-04-08</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.4.1</td>
<td>Amazon MSK now supports Apache Kafka version 2.4.1.</td>
<td>2020-04-02</td>
</tr>
<tr>
<td>Support for Streaming Broker Logs</td>
<td>Amazon MSK can now stream broker logs to CloudWatch Logs, Amazon S3, and Amazon Kinesis Data Firehose. Kinesis Data Firehose can, in turn, deliver these logs to the destinations that it supports, such as OpenSearch Service.</td>
<td>2020-02-25</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.3.1</td>
<td>Amazon MSK now supports Apache Kafka version 2.3.1.</td>
<td>2019-12-19</td>
</tr>
<tr>
<td>Open Monitoring</td>
<td>Amazon MSK now supports open monitoring with Prometheus.</td>
<td>2019-12-04</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.2.1</td>
<td>Amazon MSK now supports Apache Kafka version 2.2.1.</td>
<td>2019-07-31</td>
</tr>
<tr>
<td>General Availability</td>
<td>New features include tagging support, authentication, TLS encryption, configurations, and the ability to update broker storage.</td>
<td>2019-05-30</td>
</tr>
<tr>
<td>Support for Apache Kafka 2.1.0</td>
<td>Amazon MSK now supports Apache Kafka version 2.1.0.</td>
<td>2019-02-05</td>
</tr>
</tbody>
</table>
AWS glossary

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