Amazon Managed Streaming for Apache Kafka

Developer Guide

Amazon Managed Streaming for Apache Kafka: Developer Guide

The AWS Documentation website is getting a new look!

Try it now and let us know what you think. Switch to the new look >>

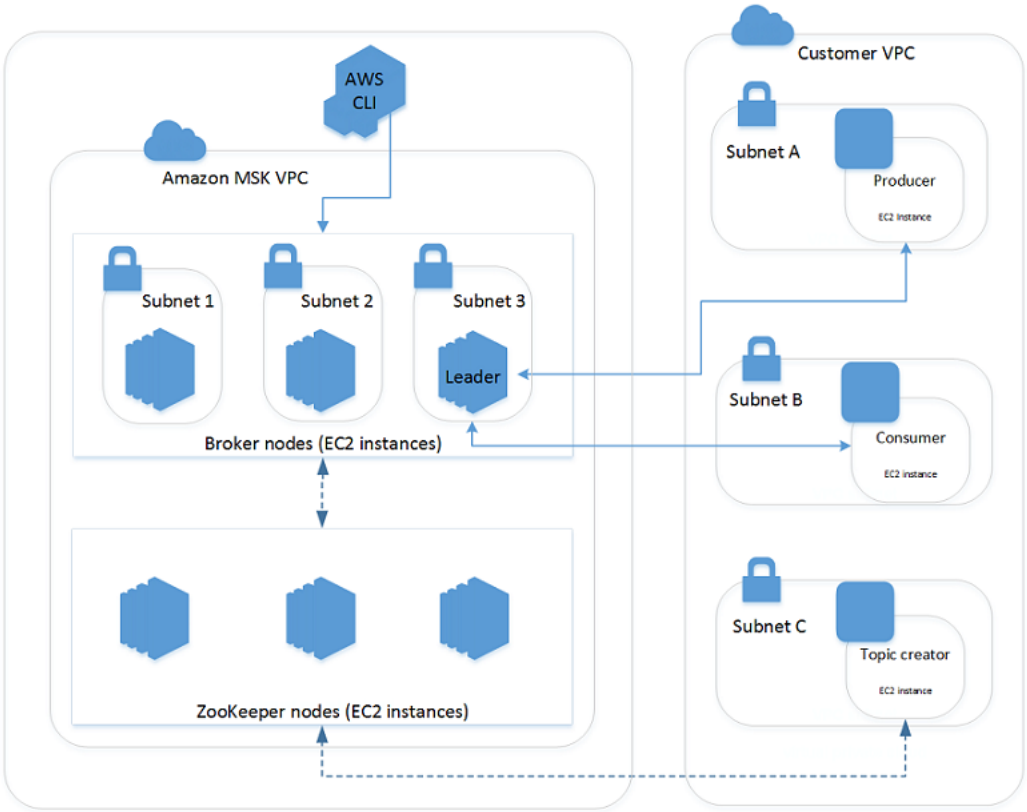
You can return to the original look by selecting English in the language selector above.

Table of Contents

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 and 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 Apache Kafka versions 1.1.1 and 2.2.1.

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

Amazon MSK detects and automatically recovers from the most common failure scenarios for Multi-AZ clusters so that your producer and consumer applications can continue their write and read operations with minimal impact. Amazon MSK automatically detects the following failure scenarios:

- Loss of network connectivity to a broker
- Compute unit failure for a broker

When Amazon MSK detects one of these failures, it replaces the unhealthy or unreachable broker with a new broker. 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. 4).

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* (p. 44).

Setting Up Amazon MSK

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

Tasks

Sign Up for AWS

When you sign up for Amazon Web Services (AWS), your AWS 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 AWS account

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

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

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 AWS 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.
- The Amazon Managed Streaming for Kafka API Reference documents the API operations that Amazon MSK supports.
- The AWS SDKs for Go, Java, JavaScript, .NET, Node.js, PHP, Python, and Ruby include Amazon MSK support and samples.

Getting Started Using Amazon MSK

The goal of this section is to help you get started quickly with Amazon MSK. This tutorial shows you how to set up an MSK cluster, create the cluster, produce and consume data, and finally, monitor the health of your cluster using metrics.

This is a step-by-step tutorial that uses the AWS Management Console and the AWS CLI. Alternatively, you can use AWS CloudFormation to set up an MSK cluster. For some example AWS CloudFormation templates, see Amazon MSK CloudFormation Examples.

Prerequisites

Before you start, ensure that you have an AWS 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. 3).

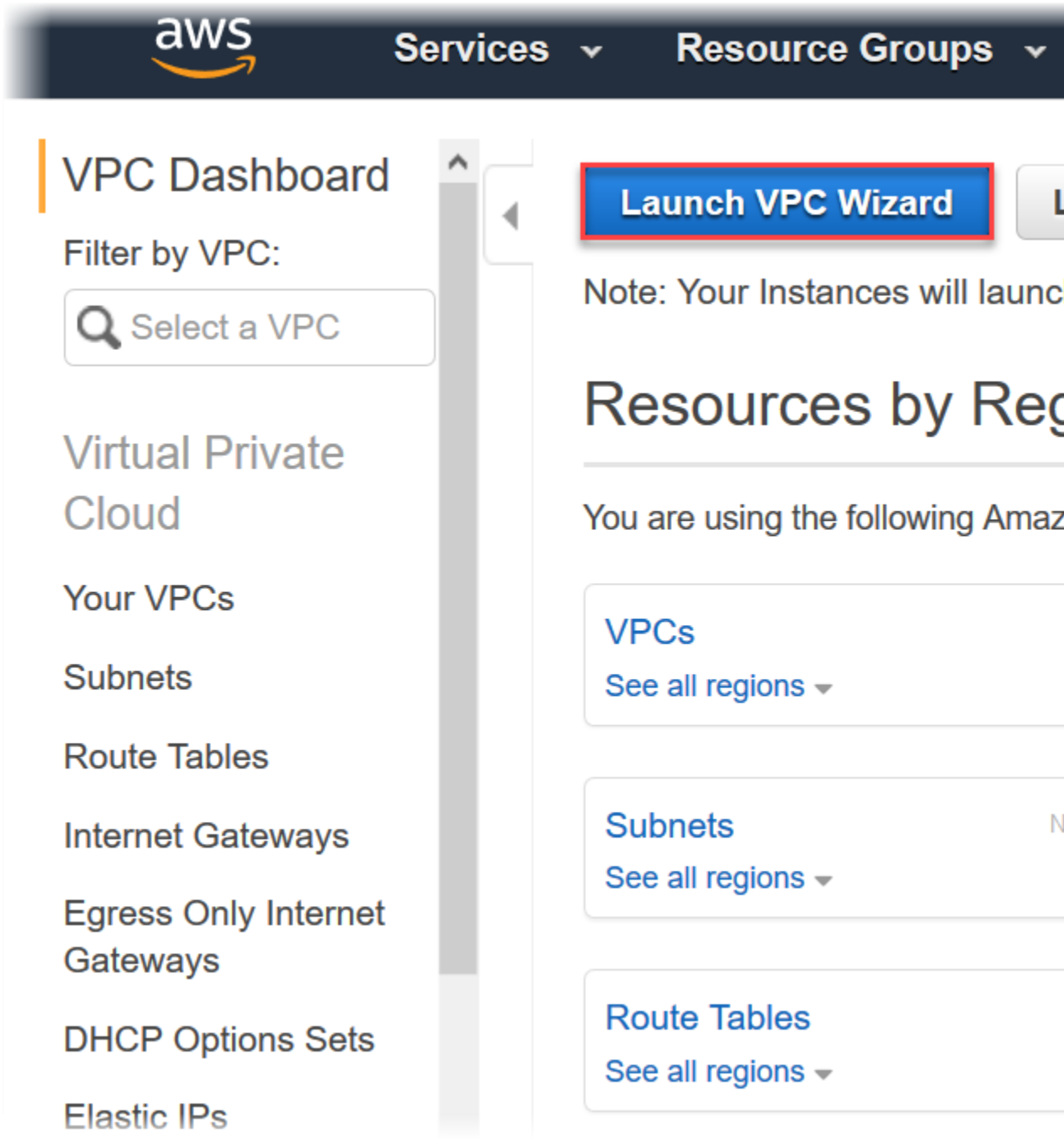
Topics

Step 1: Create a VPC for Your MSK Cluster

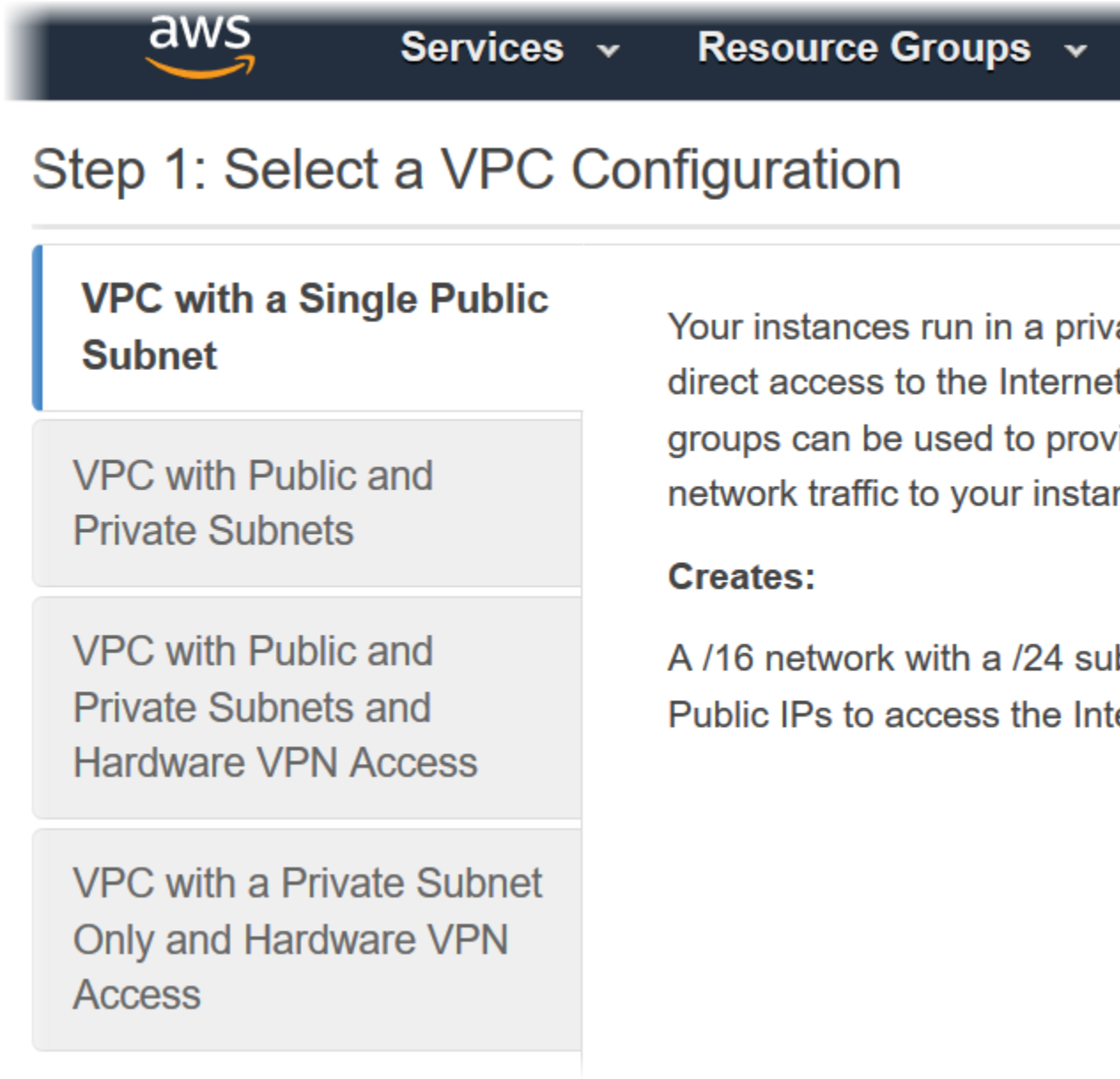
In the first step of Getting Started Using Amazon MSK (p. 4), you create an Amazon Virtual Private Cloud (Amazon VPC), where you create your Apache Kafka cluster in a later step.

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`**.
5. For **Availability Zone**, choose **us-east-1a**.
6. For **Subnet name**, enter **`AWSKafkaTutorialSubnet-1`**.
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

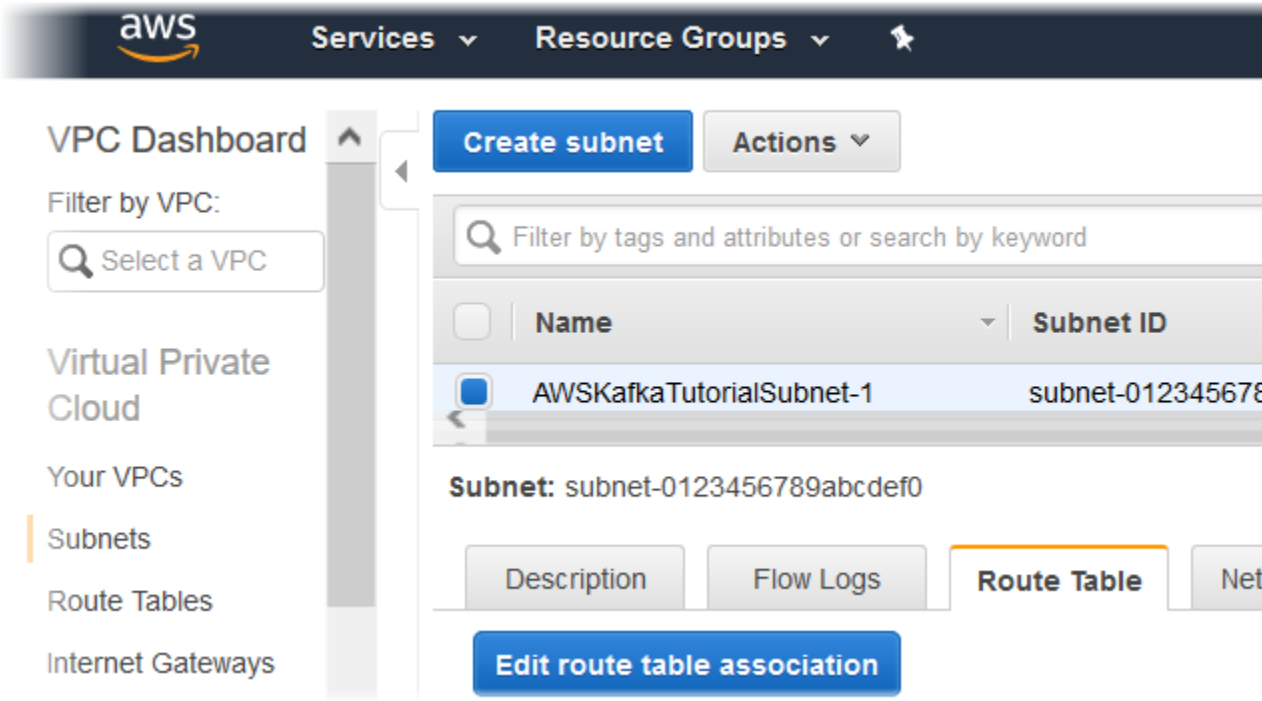
Step 2: Enable High Availability and Fault Tolerance (p. 7)

Step 2: Enable High Availability and Fault Tolerance

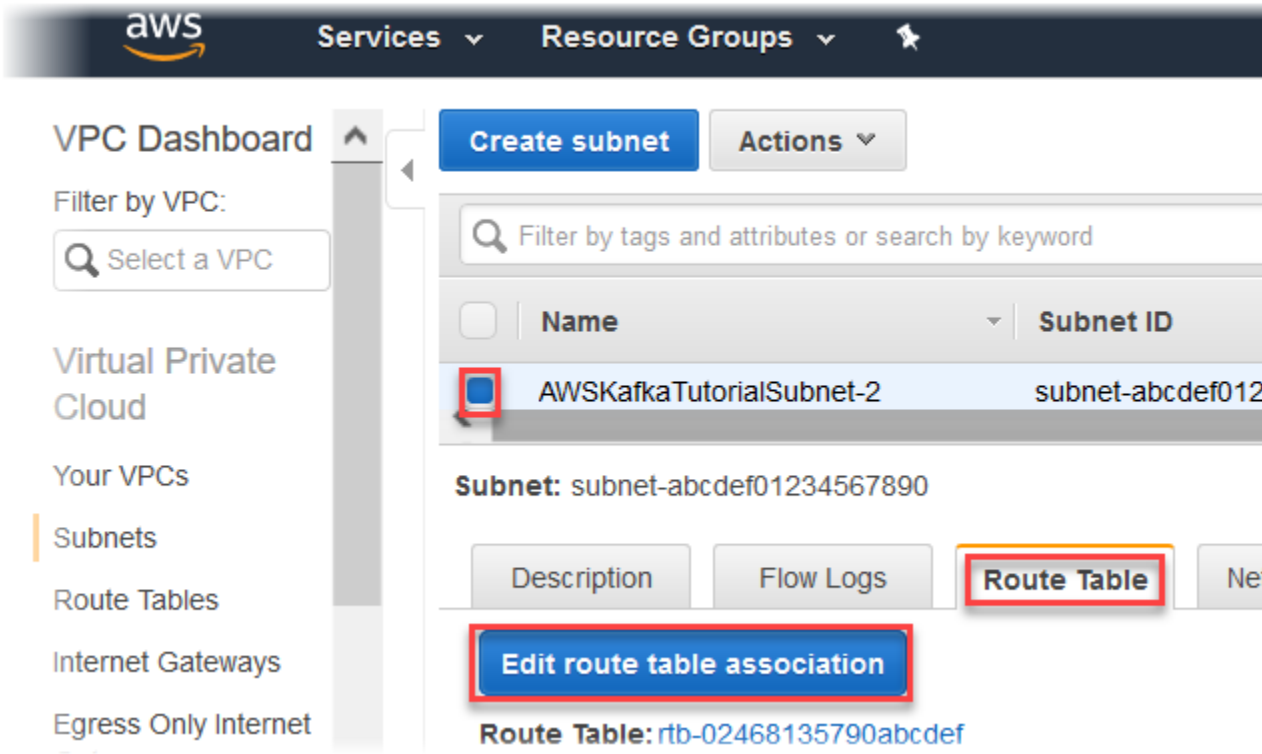
In this step of Getting Started Using Amazon MSK (p. 4), you enable high availability and fault tolerance. To do so, you add two subnets to the virtual private cloud (VPC) that you created with one subnet in the previous step. After you complete this step, you will have three subnets in three different 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`**.
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`**.
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. 8)

Step 3: Create an Amazon MSK Cluster

In this step of Getting Started Using Amazon MSK (p. 4), you create an Amazon MSK cluster in the virtual private cloud (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.
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 because you need it in the next step.
4. Copy the following JSON and save it to a file. Name the file `brokernodegroupinfo.json`. Replace the three subnet IDs and the security group ID in the JSON with the values that you saved

in previous steps. Then save the updated JSON file on the computer where you have the AWS CLI installed.

```
{
  "InstanceType": "kafka.m5.large",
  "ClientSubnets": [
    "AWSKafkaTutorialSubnet-1 Subnet ID",
    "AWSKafkaTutorialSubnet-2 Subnet ID",
    "AWSKafkaTutorialSubnet-3 Subnet ID"
  ],
  "SecurityGroups": [
    "AWSKafkaTutorialVPC Security Group ID"
  ]
}
```

5. Copy the following JSON and save it to a file. Name the file `encryptioninfo.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. 35).

```
{
   "EncryptionAtRest": {
       "DataVolumeKMSKeyId": "your-CMK"
    },
   "EncryptionInTransit": {
        "InCluster": true,
        "ClientBroker": "TLS"
    }
}
```

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 `brokernodegroupinfo.json` and `encryptioninfo.json` files.

```
aws kafka create-cluster --cluster-name "AWSKafkaTutorialCluster" --broker-node-group-
info file://brokernodegroupinfo.json --encryption-info file://encryptioninfo.json
 --kafka-version "2.2.1" --number-of-broker-nodes 3 --enhanced-monitoring
 PER_TOPIC_PER_BROKER --region us-east-1
```

The output of the command looks like the following JSON:

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

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

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

Next Step

Step 4: Create a Client Machine (p. 10)

Step 4: Create a Client Machine

In this step of Getting Started Using Amazon MSK (p. 4), you create a client machine. You use this client machine to create a topic that produces and consumes data.

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.
16. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
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 the **Description** column has the value **default VPC security group**. Choose this row by selecting the check box in the first column.
18. In the **Inbound Rules** tab, choose **Edit rules**.
19. Choose **Add Rule**.
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**.
22. Repeat these steps to add an inbound rule in the security group that corresponds to your client machine to allow it to receive traffic from the **AWSKafkaTutorialVPC** security group. This enables your client machine to communicate back and forth with your MSK cluster.

Next Step

Step 5: Create a Topic (p. 10)

Step 5: Create a Topic

In this step of Getting Started Using Amazon MSK (p. 4), 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:

```
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. 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. 8).

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

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

```
{
    "ClusterInfo": {
        "BrokerNodeGroupInfo": {
            "BrokerAZDistribution": "DEFAULT",
            "ClientSubnets": [
                "subnet-0d44a1567c2ce409a",
                "subnet-051201cac65561565",
                "subnet-08b4eceb2bd3bd8c2"
            ],
            "InstanceType": "kafka.m5.large",
            "SecurityGroups": [
                "sg-041e78b0a8ba7f834"
            ],
            "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-aaaa5555878"
            }
        },
        "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 result of this `describe-cluster` command includes an additional key named `ZooKeeperConnectionString`. 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 *`ZooKeeperConnectionString`* with the value that you saved after you ran the `describe-cluster` command.

```
bin/kafka-topics.sh --create --zookeeper ZooKeeperConnectionString --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. 12)

Step 6: Produce and Consume Data

In this step of Getting Started Using Amazon MSK (p. 4), 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.

   Note
   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
```

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. 8).

```
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 `"BootstrapBrokerString"` because you need it in the following commands.

4. Run the following command, replacing *BootstrapBrokerString* with the value that you obtained when you ran the previous command.

```
bin/kafka-console-producer.sh --broker-list BootstrapBrokerString --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 *BootstrapBrokerString* with the value that you saved earlier. Then run the command using your second connection to the client machine.

```
bin/kafka-console-consumer.sh --bootstrap-server BootstrapBrokerString --
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. 13)

Step 7: Use Amazon CloudWatch to View Amazon MSK Metrics

In this step of Getting Started Using Amazon MSK (p. 4), you look at the Amazon MSK metrics in Amazon CloudWatch.

To view Amazon MSK metrics in CloudWatch

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

Next Step

Step 8: Delete the Amazon MSK Cluster (p. 14)

Step 8: Delete the Amazon MSK Cluster

In the final step of Getting Started Using Amazon MSK (p. 4), you delete the MSK cluster that you created in Step 3: Create an Amazon MSK Cluster (p. 8).

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.

```
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.

```
aws kafka delete-cluster --region us-east-1 --cluster-arn ClusterArn
```

Amazon MSK Cluster Operations

The topics in this section show you how to perform Amazon MSK operations using the AWS Management Console, the AWS CLI, and the Amazon MSK API. 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.

Topics

Create an Amazon MSK Cluster

Before you can create an Amazon MSK cluster you need to specify a VPC and set up subnets within that VPC. You need two subnets in two different Availability Zones in the following Regions: South America (São Paulo), Canada (Central), and US West (N. California). 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. 4) and ??? (p. 7).

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. 26).

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 five brokers per Availability Zone.
8. (Optional) Assign tags to your cluster. Tags are optional. For more information, see *Tagging* (p. 57).
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. 35).
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. 35).
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 "Authentication" (p. 38).
14. Choose the monitoring level you want. This determines the set of metrics you get. For more information, see *Monitoring* (p. 44).
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. 44). The `enhanced-monitoring` parameter is optional. If you don't specify it in the `create-cluster` command, you get the `DEFAULT` level of monitoring.

```
aws kafka create-cluster --cluster-name "Your-Cluster-Name" --broker-node-group-info
 file://brokernodegroupinfo.json --kafka-version "2.2.1" --number-of-broker-nodes 3 --
enhanced-monitoring "Monitoring-Level"
```

The output of the command looks like the following JSON:

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

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

3. Save the value of the `ClusterArn` key because you need it to perform other actions on your cluster.

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. 26).

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

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

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

```
aws kafka create-cluster --cluster-name ExampleClusterName --broker-node-group-info
 file://brokernodegroupinfo.json --kafka-version "1.1.1" --number-of-broker-nodes 3 --
enhanced-monitoring PER_TOPIC_PER_BROKER --configuration-info file://configuration.json
```

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

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

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

Creating a Cluster Using the API

- To create a cluster using the API, see CreateCluster.

Delete an Amazon MSK Cluster

Use the AWS Management Console to delete a cluster

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.

Use the AWS CLI to delete a cluster

- Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see the section called “List Clusters” (p. 20).

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

Use the API to delete a cluster

- To delete a cluster using the API, see DeleteCluster.

Get the ZooKeeper Connection String for an Amazon MSK Cluster

Use the AWS Management Console to get the ZooKeeper connection string

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 servers, as well as the Apache ZooKeeper connection string.

Use the AWS CLI to get the ZooKeeper connection string

1. If you don't know the Amazon Resource Name (ARN) of your cluster, you can find it by listing all the clusters in your account. For more information, see the section called “List Clusters” (p. 20).
2. To get the ZooKeeper connection string, along with other information about your cluster, run the following command, replacing *ClusterArn* with the ARN of your cluster.

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

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

```
{
    "ClusterInfo": {
        "BrokerNodeGroupInfo": {
            "BrokerAZDistribution": "DEFAULT",
            "ClientSubnets": [
                "subnet-0123456789abcdef0",
                "subnet-2468013579abcdef1",
                "subnet-1357902468abcdef2"
            ],
            "InstanceType": "kafka.m5.large",
            "StorageInfo": {
                "EbsStorageInfo": {
                    "VolumeSize": 1000
                }
            }
        },
        "ClusterArn": "arn:aws:kafka:us-east-1:111122223333:cluster/
testcluster/12345678-abcd-4567-2345-abcdef123456-2",
        "ClusterName": "testcluster",
        "CreationTime": "2018-12-02T17:38:36.75Z",
        "CurrentBrokerSoftwareInfo": {
            "KafkaVersion": "2.2.1"
        },
        "CurrentVersion": "K13V1IB3VIYZZH",
        "EncryptionInfo": {
            "EncryptionAtRest": {
                "DataVolumeKMSKeyId": "arn:aws:kms:us-east-1:555555555555:key/12345678-
abcd-2345-ef01-abcdef123456"
            }
        },
        "EnhancedMonitoring": "DEFAULT",
        "NumberOfBrokerNodes": 3,
        "State": "ACTIVE",
        "ZookeeperConnectString": "10.0.1.101:2018,10.0.2.101:2018,10.0.3.101:2018"
    }
}
```

The previous JSON example shows the `ZookeeperConnectString` key in the output of the `describe-cluster` command. Copy the value corresponding to this key and save it for when you need to create a topic on your cluster.

Important
Your Amazon MSK cluster must be in the `ACTIVE` state for you to be able to obtain the 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.

Use the API to get the ZooKeeper connection string

- To get the ZooKeeper connection string using the API, see DescribeCluster.

Get the Bootstrap Brokers for an Amazon MSK Cluster

Use the AWS Management Console to get the bootstrap brokers

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 servers, as well as the Apache ZooKeeper connection string.

Use the AWS CLI to get the bootstrap brokers

- Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see the section called "List Clusters" (p. 20).

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

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

```
{
    "BootstrapBrokerStringTls": "b-3.exampleClusterName.abcde.c2.kafka.us-
east-1.amazonaws.com:9094,b-1.exampleClusterName.abcde.c2.kafka.us-
east-1.amazonaws.com:9094,b-2.exampleClusterName.abcde.c2.kafka.us-
east-1.amazonaws.com:9094"
}
```

Use the API to get the bootstrap brokers

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

List All Amazon MSK Clusters in an Account

Use the AWS Management Console to list clusters

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.

Use the AWS CLI to list clusters

- Run the following command.

```
aws kafka list-clusters
```

Use the API to list clusters

- To list clusters using the API, see ListClusters.

Update the EBS Storage for Brokers

You can use this operation to increase the amount of storage per broker. You can't decrease the storage.

Use the AWS Management Console to update broker storage

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**.

Use the AWS CLI to update broker storage

- Run the following command, replacing *ClusterArn* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see the section called "List Clusters" (p. 20).

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

  > **Important**
  > Cluster versions aren't simple integers. You can obtain the current version by describing the cluster. An example version is `KTVPDKIKX0DER`.

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

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

Use the API to update broker storage

- To update a broker storage using the API, see UpdateBrokerStorage.

Update the Configuration of an Amazon MSK Cluster

For more information about MSK configuration, including how to create a custom configuration, which properties you can update, and what happens when you update the configuration of an existing cluster, see *Configuration* (p. 26).

Use the AWS CLI to update the configuration of a cluster

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

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

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

2. Run the following command, replacing *ClusterArn* with the ARN that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see the section called "List Clusters" (p. 20).

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

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

   Important
   Cluster versions aren't simple integers. You can obtain the current version by describing the cluster. An example version is `KTVPDKIKX0DER`.

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

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

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

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

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

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

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

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

```
{
    "ClusterOperationInfo": {
        "ClientRequestId": "982168a3-939f-11e9-8a62-538df00285db",
```

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

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

Use the API to update the configuration of a cluster

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

Update the Number of Brokers in an Amazon MSK Cluster

Use this Amazon MSK operation when you want to change the number of brokers in your MSK cluster.

Important
Make sure to use this Amazon MSK operation when you want to update the number of broker nodes. Don't try to add or remove brokers from an MSK cluster without using this operation.

For information about how to rebalance the data load after you add brokers to a cluster, see the section called “Balance Your Cluster” (p. 66).

Use the AWS Management Console to update the number of brokers

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 update.
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**.

Use the AWS CLI to update the number of brokers

1. Run the following command, replacing *`ClusterArn`* with the Amazon Resource Name (ARN) that you obtained when you created your cluster. If you don't have the ARN for your cluster, you can find it by listing all clusters. For more information, see the section called “List Clusters” (p. 20).

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

Important
Cluster versions aren't simple integers. You can obtain the current version by describing the cluster. An example version is `KTVPDKIKX0DER`.

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

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

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

```
{

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

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

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

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

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

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

Use the API to update the number of brokers

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

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

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. 29). For more information about configuration properties, see Apache Kafka Configuration.

Apache Kafka Configuration Properties That You Can Set

Name	Description
auto.create.topics.enable	Enables topic autocreation on the server.
delete.topic.enable	Enables the delete topic operation. If this config is turned off, you can't delete a topic through the admin tool.
log.cleaner.delete.retention.ms	Amount of time that you want Apache Kafka to retain deleted records.
group.initial.rebalance.delay.ms	Amount of time the group coordinator waits for more consumers to join a new group before performing the first rebalance. A longer delay means potentially fewer rebalances, but increases the time until processing begins.
group.max.session.timeout.ms	Maximum session timeout for registered consumers. Longer timeouts give consumers more time to process messages in between heartbeats at the cost of a longer time to detect failures.
group.min.session.timeout.ms	Minimum session timeout for registered consumers. Shorter timeouts result in quicker failure detection at the cost of more frequent consumer heartbeating, which can overwhelm broker resources.
log.flush.interval.messages	Number of messages accumulated on a log partition before messages are flushed to disk.
log.flush.interval.ms	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.

Name	Description
log.retention.bytes	Maximum size of the log before deleting it.
log.retention.hours	Number of hours to keep a log file before deleting it, tertiary to the log.retention.ms property.
log.retention.minutes	Number of minutes to keep a log file before deleting it, secondary to log.retention.ms property. If not set, the value in log.retention.hours is used.
log.retention.ms	Number of milliseconds to keep a log file before deleting it (in milliseconds), If not set, the value in log.retention.minutes is used.
num.partitions	Default number of log partitions per topic.
max.incremental.fetch.session.cache.slots	Maximum number of incremental fetch sessions that are maintained.
log.cleaner.min.cleanable.ratio	Minimum ratio of dirty log to total log for a log to be eligible for cleaning. A higher ratio means fewer, more efficient cleanings, but it also means more wasted space in the log.
offsets.retention.minutes	After a consumer group loses all its consumers (that is, it becomes empty) its offsets are kept for this retention period before getting discarded. For standalone consumers (that is, using manual assignment), offsets are expired after the time of the last commit plus this retention period.
zookeeper.connection.timeout.ms	Maximum time that the client waits to establish a connection to ZooKeeper. If not set, the value in zookeeper.session.timeout.ms is used.
unclean.leader.election.enable	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.
min.insync.replicas	When a producer sets acks to `"all"` (or `"-1"`), 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). 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 `"all"`. This ensures that the producer raises an exception if a majority of replicas don't receive a write.

Name	Description
message.max.bytes	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. In the latest message format version, records are always grouped into batches for efficiency. In previous message format versions, uncompressed records are not grouped into batches and this limit only applies to a single record in that case. This can be set per topic with the topic level max.message.bytes config.
log.segment.bytes	Maximum size of a single log file.
log.roll.ms	Maximum time before a new log segment is rolled out (in milliseconds). If not set, the value in log.roll.hours is used.
transaction.max.timeout.ms	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.
replica.fetch.max.bytes	Number of bytes of messages to attempt to fetch for each partition. This is not an absolute maximum. If the first record batch in the first non-empty partition of the fetch is larger than this value, the record batch is returned to ensure 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).

To learn how you can create a custom MSK configuration, list all configurations, or describe them, see the section called "Configuration Operations" (p. 30). To create an MSK cluster using a custom MSK configuration or to update a cluster with a new custom configuration, see *Cluster Operations* (p. 15).

When you update your existing MSK cluster with a custom MSK configuration, Amazon MSK does rolling restarts when necessary, using best practices to minimize customer downtime. For example, after Amazon MSK restarts each broker, it tries to let the broker catch up on data that the broker might have missed during the configuration update before it moves to the next broker.

Dynamic Configuration

In addition to the configuration properties that Amazon MSK provides, you can dynamically set cluster- 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.

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.

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

Name	Description	Default Value
num.network.threads	Number of threads that the server uses for receiving requests from the network and sending responses to the network.	5
num.io.threads	Number of threads that the server uses for processing requests, which may include disk I/O.	8
num.replica.fetchers	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.	2
socket.send.buffer.bytes	SO_SNDBUF buffer of the socket sever sockets. If the value is -1, the OS default is used.	102400
socket.receive.buffer.bytes	SO_RCVBUF buffer of the socket sever sockets. If the value is -1, the OS default is used.	102400
socket.request.max.bytes	Maximum number of bytes in a socket request.	104857600
num.partitions	Default number of log partitions per topic.	1
auto.create.topics.enable	Enables autocreation of a topic on the server.	false
default.replication.factor	Default replication factors for automatically created topics.	3
min.insync.replicas	When a producer sets acks to `"all"` (or `"-1"`),	2

Name	Description	Default Value
	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). 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 `"all"`. This ensures that the producer raises an exception if a majority of replicas do not receive a write.	
unclean.leader.election.enable	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.	true
auto.leader.rebalance.enable	Enables auto leader balancing. A background thread checks and triggers leader balance if required at regular intervals.	true
allow.everyone.if.no.acl.found	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.	true
zookeeper.set.acl	Set client to use secure ACLs.	false

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 *Cluster Operations* (p. 15).

To create an MSK configuration

1. Create a file where you specify the configuration properties that you want to set and the values that you want to assign to them. The following are the contents of an example configuration file.

```
auto.create.topics.enable = true
```

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

log.roll.ms = 604800000
```

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

   Note
   The name that you choose for your configuration must match the following regex: "^[0-9A-Za-z-]+$".

```
aws kafka create-configuration --name "ExampleConfigurationName" --description
 "Example configuration description." --kafka-versions "1.1.1" --server-properties
 file://config-file-path
```

   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-
abcd-1234-abcd123e8e8e-1",
    "CreationTime": "2019-05-21T19:37:40.626Z",
    "LatestRevision": {
        "CreationTime": "2019-05-21T19:37:40.626Z",
        "Description": "Example configuration description.",
        "Revision": 1
    },
    "Name": "ExampleConfigurationName"
}
```

3. The previous command returns an Amazon Resource Name (ARN) for 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 describe an MSK configuration

- 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
```

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

```
{
    "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/abcdabcd-
abcd-1234-abcd-abcd123e8e8e-1",
    "CreationTime": "2019-05-21T00:54:23.591Z",
    "Description": "Example configuration description.",
    "KafkaVersions": [
        "1.1.1"
    ],
    "LatestRevision": {
```

```
        "CreationTime": "2019-05-21T00:54:23.591Z",
        "Description": "Example configuration description.",
        "Revision": 1
    },
    "Name": "SomeTest"
}
```

To describe an MSK configuration revision

Describing an MSK configuration using the `describe-configuration` 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.

```
{
    "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/abcdabcd-
abcd-1234-abcd-abcd123e8e8e-1",
    "CreationTime": "2019-05-21T00:54:23.591Z",
    "Description": "Example configuration description.",
    "Revision": 1,
    "ServerProperties":
 "YXV0by5jcmVhdGUudG9waWNzLmVuYWJsZSA9IHRydWUKCgp6b29rZWVwZXIuY29ubmVjdGlvbi50aW1lb3V0Lm1zID0gMTAwM
}
```

The value of `ServerProperties` is encoded 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:

```
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.

```
{
    "Configurations": [
        {
```

```
            "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/
abcdabcd-abcd-1234-abcd-abcd123e8e8e-1",
            "CreationTime": "2019-05-21T00:54:23.591Z",
            "Description": "Example configuration description.",
            "KafkaVersions": [
                "1.1.1"
            ],
            "LatestRevision": {
                "CreationTime": "2019-05-21T00:54:23.591Z",
                "Description": "Example configuration description.",
                "Revision": 1
            },
            "Name": "SomeTest"
        },
        {
            "Arn": "arn:aws:kafka:us-east-1:123456789012:configuration/SomeTest/
abcdabcd-1234-abcd-1234-abcd123e8e8e-1",
            "CreationTime": "2019-05-03T23:08:29.446Z",
            "Description": "Example configuration description.",
            "KafkaVersions": [
                "1.1.1"
            ],
            "LatestRevision": {
                "CreationTime": "2019-05-03T23:08:29.446Z",
                "Description": "Example configuration description.",
                "Revision": 1
            },
            "Name": "ExampleConfigurationName"
        }
    ]
}
```

Security in Amazon Managed Streaming for 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 Kafka, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your 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 AWS services that help you to monitor and secure your Amazon MSK resources.

Topics

Accessing an Amazon MSK Cluster

You can access your Amazon MSK cluster from an Amazon VPC or from an EC2-Classic instance.

Accessing your Amazon MSK cluster from an Amazon VPC

To access your MSK cluster from an Amazon EC2 instance that is in an Amazon VPC, follow the steps in ??? (p. 10).

Accessing your Amazon MSK cluster from an EC2-Classic instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Running Instances**.

3. Choose your EC2-Classic instance from the list of instances by selecting the check box next to it.
4. In the **Actions** menu choose **ClassicLink**, then choose **Link to VPC**.
5. In the **Select a VPC** list, choose the VPC that you want to link your EC2-Classic instance to. If your VPC is associated with more than one security group, choose the security group you want to associate with your EC2-Classic instance, then choose **Link to VPC**.
6. In the **Description** tab in the lower part of the page, look for **Private IPs** and copy the private IP associated with your EC2-Classic instance.
7. Using the AWS CLI, run the following command, replacing *`ClusterArn`* with the Amazon Resource Name (ARN) for your MSK cluster.

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

8. In the output of the `describe-cluster` command, look for `SecurityGroups` and save the ID of the security group for your MSK cluster.
9. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
10. In the left pane, choose **Security Groups**.
11. 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.
12. In the lower half of the page, choose **Inbound Rules**.
13. Choose **Edit rules**, then choose **Add Rule**.
14. For the **Type** field, choose **All traffic** in the drop-down list.
15. Leave the **Source** set to **Custom** and enter the private IP of your EC2-Classic instance, followed immediately by **`/32`** with no intervening spaces.
16. 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 producers and consumers in plaintext, brokers use port 9092.
- To communicate with producers and consumers in TLS, brokers use port 9094.
- Apache ZooKeeper nodes use port 2181.

Amazon MSK Encryption

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.

Working 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. 10).
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

Important
TLS authentication isn't available for Amazon MSK in the South America (São Paulo) Region.

To create a cluster that supports client authentication

1. Make sure that you have a certificate authority (CA) available to sign your ACM private CA certificate. This can be a commercial or an on-premises CA. It can be a root or an intermediate CA. You can use OpenSSL to create a test certificate authority. We recommend, however, that you use a production CA rather than a test CA. For more information, see AWS Certificate Manager Private Certificate Authority.
2. Create a file named `clientauthinfo.json` with the following contents. Replace *Private-CA-ARN* with the ARN of your PCA.

```
{
   "Tls": {
       "CertificateAuthorityArnList": ["Private-CA-ARN"]
    }
}
```

3. Create a file named `brokernodegroupinfo.json` as described in the section called “Step 3: Create a Cluster” (p. 8).
4. Client authentication requires that you also enable encryption in transit between clients and brokers. Create a file named `encryptioninfo.json` with the following contents. Replace *KMS-Key-ARN* with the ARN of your KMS key. You can set `ClientBroker` to `TLS` or `TLS_PLAINTEXT`.

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

   For more information about encryption, see the section called “Encryption” (p. 35).
5. On a machine where you have the AWS CLI installed, run the following command to create a cluster with authentication and in-transit encryption enabled. Save the cluster ARN provided in the response.

```
aws kafka create-cluster --cluster-name "AuthenticationTest" --broker-node-group-info
 file://brokernodegroupinfo.json --encryption-info file://encryptioninfo.json --client-
```

```
authentication file://clientauthinfo.json --kafka-version "2.2.1" --number-of-broker-
nodes 3
```

To set up a client to use authentication

1. Create an Amazon EC2 instance to use as a client machine. For simplicity, create this instance in the same VPC you used for the cluster. See the section called "Step 4: Create a Client Machine" (p. 10) for instructions on how to create such a client machine.
2. Create a topic. See the instructions under the section called "Step 5: Create a Topic" (p. 10).
3. On a machine where you have the AWS CLI installed, run the following command to get the bootstrap brokers of the cluster. Replace *Cluster-ARN* with the ARN of your cluster.

   ```
   aws kafka get-bootstrap-brokers --cluster-arn Cluster-ARN
   ```

   Save the string associated with `BootstrapBrokerStringTls` in the response.
4. On your client machine, run the following command to use the JVM trust store to create your client trust store. If your JVM path is different, adjust the command accordingly.

   ```
   cp /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.201.b09-0.amzn2.x86_64/jre/lib/security/
   cacerts kafka.client.truststore.jks
   ```
5. On your client machine, run the following command to create a private key for your client. Replace *Distinguished-Name*, *Example-Alias*, *Your-Store-Pass*, and *Your-Key-Pass* with strings of your choice.

   ```
   keytool -genkey -keystore kafka.client.keystore.jks -validity 300 -storepass Your-
   Store-Pass -keypass Your-Key-Pass -dname "CN=Distinguished-Name" -alias Example-Alias -
   storetype pkcs12
   ```
6. On your client machine, run the following command to create a certificate request with the private key you created in the previous step.

   ```
   keytool -keystore kafka.client.keystore.jks -certreq -file client-cert-sign-request -
   alias Example-Alias -storepass Your-Store-Pass -keypass Your-Key-Pass
   ```
7. Open the `client-cert-sign-request` file and ensure that it starts with `-----BEGIN CERTIFICATE REQUEST-----` and ends with `-----END CERTIFICATE REQUEST-----`. If it starts with `-----BEGIN NEW CERTIFICATE REQUEST-----`, delete the word `NEW` (and the single space that follows it) from the beginning and the end of the file.
8. On a machine where you have the AWS CLI installed, run the following command to sign your certificate request. Replace *Private-CA-ARN* with the ARN of your PCA. You can change the validity value if you want. Here we use 300 as an example.

   ```
   aws acm-pca issue-certificate --certificate-authority-arn Private-CA-ARN --csr
    file://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.

   ```
   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 to add this certificate to your keystore so you can present it when you talk to the MSK brokers.

```
keytool -keystore kafka.client.keystore.jks -import -file signed-certificate-from-acm -
alias Example-Alias -storepass Your-Store-Pass -keypass Your-Key-Pass
```

12. Create a file named `client.properties` with the following contents. Adjust the truststore and keystore locations to the paths where you saved `kafka.client.truststore.jks`.

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

To produce and consume messages using authentication

1. Run the following command to create a topic.

```
bin/kafka-topics.sh --create --zookeeper ZooKeeper-Connection-String --replication-
factor 3 --partitions 1 --topic ExampleTopic
```

2. Run the following command to start a console producer. The file named `client.properties` is the one you created in the previous procedure.

```
bin/kafka-console-producer.sh --broker-list BootstrapBroker-String --topic ExampleTopic
 --producer.config client.properties
```

3. In a new command window on your client machine, run the following command to start a console consumer.

```
bin/kafka-console-consumer.sh --bootstrap-server BootstrapBroker-String --topic
 ExampleTopic --consumer.config client.properties
```

4. Type messages in the producer window and watch them appear in the consumer window.

To add or remove read and write access to a topic

1. 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`.

2. 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`.

For more information about adding, removing, and listing ACLs, see Kafka Authorization Command Line Interface.

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.

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 "Get the ZooKeeper Connection String" (p. 18). Save the IP addresses of the Apache ZooKeeper nodes because you will need them later in this procedure.
2. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
3. In the left pane, under **NETWORK & SECURITY**, choose **Network Interfaces**.
4. 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.
5. Select the check box at the beginning of the row that corresponds to the first network interface in the list.
6. 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.
7. 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.
8. 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.
9. 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 Service-Linked Roles for Amazon MSK

Amazon MSK uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon MSK. Service-linked roles are predefined by Amazon MSK and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon MSK easier because you do not have to manually add the necessary permissions. Amazon MSK defines the permissions of its service-linked roles. Unless defined otherwise, only Amazon MSK can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see 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.

Topics

Service-Linked Role Permissions for Amazon MSK

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

The AWSServiceRoleForKafka service-linked role trusts the following services to assume the role:

- `kafka.amazonaws.com`

The role permissions policy allows Amazon MSK to complete the following actions on the specified resources:

- Action: `ec2:CreateNetworkInterface` on `*`

  Action: `ec2:DescribeNetworkInterfaces` on `*`

  Action: `ec2:CreateNetworkInterfacePermission` on `*`

  Action: `ec2:AttachNetworkInterface` on `*`

  Action: `ec2:DeleteNetworkInterface` on `*`

  Action: `ec2:DetachNetworkInterface` on `*`

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 AWS Regions where the service is available. For more information, see AWS Regions and Endpoints.

Monitoring Amazon MSK with Amazon CloudWatch

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

Topics

Amazon MSK Monitoring Levels

When creating an Amazon MSK cluster, you can set the enhancedMonitoring property to one of three monitoring levels: `DEFAULT`, `PER_BROKER`, or `PER_TOPIC_PER_BROKER`. The tables in the following section show all the metrics that Amazon MSK makes available starting at each monitoring level.

Amazon MSK Metrics

Amazon MSK integrates with Amazon CloudWatch metrics 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. The following three tables show the metrics that become available at each of the three monitoring levels.

`DEFAULT` Level Monitoring

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

Metrics available at the `DEFAULT` monitoring level

Name	When Visible	Dimensio	Description
`ZooKeeperRequestLatencyMsMean`	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	Mean latency in milliseconds for ZooKeeper requests from broker.
`ZooKeeperSessionState`	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	Connection status of broker's ZooKeeper session which may be one of the following: NOT_CONNECTED: '0.0', ASSOCIATING: '0.1', CONNECTING: '0.5', CONNECTEDREADONLY: '0.8', CONNECTED: '1.0', CLOSED: '5.0', AUTH_FAILED: '10.0'.
`ActiveControllerCount`	After the cluster gets to the ACTIVE state.	Cluster Name	Only one controller per cluster should be active at any given time.

Name	When Visible	Dimensio	Description
GlobalPartitionCount	After the cluster gets to the ACTIVE state.	Cluster Name	Total number of partitions across all brokers in the cluster.
GlobalTopicCount	After the cluster gets to the ACTIVE state.	Cluster Name	Total number of topics across all brokers in the cluster.
OfflinePartitionsCount	After the cluster gets to the ACTIVE state.	Cluster Name	Total number of partitions that are offline in the cluster.
SwapUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of swap memory that is in use for the broker.
SwapFree	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of swap memory that is available for the broker.
MemoryUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of memory that is in use for the broker.
MemoryBuffered	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of buffered memory for the broker.
MemoryFree	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of memory that is free and available for the broker.
MemoryCached	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The size in bytes of cached memory for the broker.
CpuUser	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU in user space.
CpuSystem	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU in kernel space.
CpuIdle	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of CPU idle time.

Name	When Visible	Dimensio	Description
RootDiskUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of the root disk used by the broker.
KafkaAppLogsDiskUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of disk space used for application logs.
KafkaDataLogsDiskUsed	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The percentage of disk space used for data logs.
NetworkRxErrors	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of network receive errors for the broker.
NetworkTxErrors	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of network transmit errors for the broker.
NetworkRxDropped	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of dropped receive packages.
NetworkTxDropped	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of dropped transmit packages.
NetworkRxPackets	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of packets received by the broker.
NetworkTxPackets	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of packets transmitted by the broker.

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.

Additional metrics that are available starting at the PER_BROKER monitoring level

Name	When Visible	Dimensi	Description
MessagesInPerSec	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of incoming messages per second for the broker.
NetworkProcessorAvgIdlePercent	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The average percentage of the time the network processors are idle.
RequestHandlerAvgIdlePercent	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The average percentage of the time the request handler threads are idle.
LeaderCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of leader replicas.
PartitionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of partitions for the broker.
ProduceLocalTimeMsMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean time in milliseconds for the follower to send a response.
ProduceMessageConversionsTimeMsMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean time in milliseconds spent on message format conversions.
ProduceRequestQueueTimeMsMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean time in milliseconds that request messages spend in the queue.
ProduceResponseQueueTimeMsMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean time in milliseconds that response messages spend in the queue.
ProduceResponseSendTimeMsMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean time in milliseconds spent on sending response messages.
ProduceTotalTimeMsMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean produce time in milliseconds.

Name	When Visible	Dimensi	Description
RequestBytesMean	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The mean number of request bytes for the broker.
UnderMinIsrPartitionCount	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of under minIsr partitions for the broker.
UnderReplicatedPartitions	After the cluster gets to the ACTIVE state.	Cluster Name, Broker ID	The number of under-replicated partitions for the broker.
BytesInPerSec	After you create a topic.	Cluster Name, Broker ID	The number of bytes per second received from clients.
BytesOutPerSec	After you create a topic.	Cluster Name, Broker ID	The number of bytes per second sent to clients.
MessagesInPerSec	After you create a topic.	Cluster Name, Broker ID	The number of messages received from clients per second.
FetchMessageConversionsPerSec	After you create a topic.	Cluster Name, Broker ID	The number of fetch message conversions per second for the broker.
ProduceMessageConversionsPerSec	After you create a topic.	Cluster Name, Broker ID	The number of produce message conversions per second for the broker.
FetchConsumerTotalTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean total time in milliseconds that consumers spend on fetching data from the broker.
FetchFollowerTotalTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean total time in milliseconds that followers spend on fetching data from the broker.
FetchConsumerRequestQueueTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean time in milliseconds that the consumer request waits in the request queue.

Name	When Visible	Dimensi	Description
FetchFollowerRequestQueueTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean time in milliseconds that the follower request waits in the request queue.
FetchConsumerLocalTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean time in milliseconds that the consumer request is processed at the leader.
FetchFollowerLocalTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean time in milliseconds that the follower request is processed at the leader.
FetchConsumerResponseQueueTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean time in milliseconds that the consumer request waits in the response queue.
FetchFollowerResponseQueueTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean time in milliseconds that the follower request waits in the response queue.
FetchConsumerResponseSendTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean time in milliseconds for the consumer to send a response.
FetchFollowerResponseSendTimeMsMean	After there's a producer/consumer.	Cluster Name, Broker ID	The mean time in milliseconds for the follower to send a response.
ProduceThrottleTime	After bandwidth throttling is applied.	Cluster Name, Broker ID	The average produce throttle time in milliseconds.
ProduceThrottleByteRate	After bandwidth throttling is applied.	Cluster Name, Broker ID	The number of throttled bytes per second.
ProduceThrottleQueueSize	After bandwidth throttling is applied.	Cluster Name, Broker ID	The number of messages in the throttle queue.
FetchThrottleTime	After bandwidth throttling is applied.	Cluster Name, Broker ID	The average fetch throttle time in milliseconds.

Name	When Visible	Dimensi	Description
FetchThrottleByteRate	After bandwidth throttling is applied.	Cluster Name, Broker ID	The number of throttled bytes per second.
FetchThrottleQueueSize	After bandwidth throttling is applied.	Cluster Name, Broker ID	The number of messages in the throttle queue.
RequestThrottleTime	After request throttling is applied.	Cluster Name, Broker ID	The average request throttle time in milliseconds.
RequestTime	After request throttling is applied.	Cluster Name, Broker ID	The average time spent in broker network and I/O threads to process requests.
RequestThrottleQueueSize	After request throttling is applied.	Cluster Name, Broker ID	The number of messages in the throttle queue.
RequestExemptFromThrottleTime	After request throttling is applied.	Cluster Name, Broker ID	The average time spent in broker network and I/O threads to process requests that are exempt from throttling.

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.

Additional metrics that are available starting at the PER_TOPIC_PER_BROKER monitoring level

Name	When Visible	Dimensic	Description
BytesInPerSec	After you create a topic.	Cluster Name, Broker ID, Topic	The number of bytes received per second.
BytesOutPerSec	After you create a topic.	Cluster Name, Broker ID, Topic	The number of bytes sent per second.
MessagesInPerSec	After you create a topic.	Cluster Name,	The number of messages received per second.

Name	When Visible	Dimensic	Description
		Broker ID, Topic	
`FetchMessageConversionsPerSec`	After you create a topic.	Cluster Name, Broker ID, Topic	The number of fetched messages converted per second.
`ProduceMessageConversionsPerSec`	After you create a topic.	Cluster Name, Broker ID, Topic	The number of conversions per second for produced messages.

Viewing Amazon MSK Metrics Using Amazon 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 Checking 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.

To set up and use Burrow with Amazon MSK

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. 4).
2. Run the following command on the EC2 instance that serves as your client machine.

   ```
   sudo yum install go
   ```
3. Run the following command on the client machine to get the Burrow project t.

   ```
   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.

   ```
   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.

   ```
   /home/ec2-user/go/bin/dep ensure
   ```
6. Run the following command in the same folder.

   ```
   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 `ZookeeperConnectString`. See the section called "Get the ZooKeeper Connection String" (p. 18).

   To get your bootstrap brokers, see the section called "Get the Bootstrap Brokers" (p. 20).

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

   ```
   [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"
kafka-version="0.10.0"

[cluster.MSK-cluster-name]
class-name="kafka"
servers=[ "bootstrap-broker-host-port-pair-1", "bootstrap-broker-host-port-pair-2",
 "bootstrap-broker-host-port-pair-3" ]
client-profile="test"
topic-refresh=120
offset-refresh=30

[consumer.MSK-cluster-name]
class-name="kafka"
cluster="MSK-cluster-name"
servers=[ "bootstrap-broker-host-port-pair-1", "bootstrap-broker-host-port-pair-2",
 "bootstrap-broker-host-port-pair-3" ]
client-profile="test"
group-blacklist="^(console-consumer-|python-kafka-consumer-|quick-).*$"
group-whitelist=""

[consumer.MSK-cluster-name_zk]
class-name="kafka_zk"
cluster="MSK-cluster-name"
servers=[ "ZooKeeper-host-port-pair-1", "ZooKeeper-host-port-pair-2", "ZooKeeper-host-
port-pair-3" ]
zookeeper-path="/kafka-cluster"
zookeeper-timeout=30
group-blacklist="^(console-consumer-|python-kafka-consumer-|quick-).*$"
group-whitelist=""
```

8. In the `go/bin` folder run the following command.

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

9. Check for errors in the `bin/log/burrow.log` file.
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.

Logging Amazon MSK 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 all API calls for Amazon MSK as events. The calls captured include calls from the Amazon MSK console and code calls to the Amazon MSK API operations.

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, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Amazon MSK Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon MSK, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for 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 AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

Amazon MSK supports logging the following actions as events in CloudTrail log files:

- CreateCluster
- ListClusters
- DescribeCluster
- DeleteCluster
- GetBootstrapBrokers
- ListNodes

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, so they don't appear in any specific order.

The following example shows CloudTrail log entries that demonstrate the `DescribeCluster` and `DeleteCluster` actions.


```
{
  "Records": [
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "ABCDEF0123456789ABCDE",
        "arn": "arn:aws:iam::012345678901:user/Joe",
        "accountId": "012345678901",
        "accessKeyId": "AIDACKCEVSQ6C2EXAMPLE",
        "userName": "Joe"
      },
      "eventTime": "2018-12-12T02:29:24Z",
      "eventSource": "kafka.amazonaws.com",
      "eventName": "DescribeCluster",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "aws-cli/1.14.67 Python/3.6.0 Windows/10 botocore/1.9.20",
      "requestParameters": {
        "clusterArn": "arn%3Aaws%3Akafka%3Aus-east-1%3A012345678901%3Acluster
%2Fexamplecluster%2F01234567-abcd-0123-abcd-abcd0123efa-2"
      },
      "responseElements": null,
      "requestID": "bd83f636-fdb5-abcd-0123-157e2fbf2bde",
      "eventID": "60052aba-0123-4511-bcde-3e18dbd42aa4",
      "readOnly": true,
      "eventType": "AwsApiCall",
      "recipientAccountId": "012345678901"
    },
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "ABCDEF0123456789ABCDE",
        "arn": "arn:aws:iam::012345678901:user/Joe",
        "accountId": "012345678901",
        "accessKeyId": "AIDACKCEVSQ6C2EXAMPLE",
        "userName": "Joe"
      },
      "eventTime": "2018-12-12T02:29:40Z",
      "eventSource": "kafka.amazonaws.com",
```

```
      "eventName": "DeleteCluster",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "aws-cli/1.14.67 Python/3.6.0 Windows/10 botocore/1.9.20",
      "requestParameters": {
        "clusterArn": "arn%3Aaws%3Akafka%3Aus-east-1%3A012345678901%3Acluster
%2Fexamplecluster%2F01234567-abcd-0123-abcd-abcd0123efa-2"
      },
      "responseElements": {
        "clusterArn": "arn:aws:kafka:us-east-1:012345678901:cluster/
examplecluster/01234567-abcd-0123-abcd-abcd0123efa-2",
        "state": "DELETING"
      },
      "requestID": "c6bfb3f7-abcd-0123-afa5-293519897703",
      "eventID": "8a7f1fcf-0123-abcd-9bdb-1ebf0663a75c",
      "readOnly": false,
      "eventType": "AwsApiCall",
      "recipientAccountId": "012345678901"
    }
  ]
}
```

Tagging Your Amazon MSK Resources

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

Topics

Tag Basics

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

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

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

The following are several examples of tags:

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

Tracking Costs Using Tagging

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

Migrating Clusters Using 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.

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.in.flight.requests.per.connection=1
  - retries=Int.Max_Value
  - acks=all
  - max.block.ms = Long.Max_Value
- For a high producer throughput:
  - max.in.flight.requests.per.connection = 1+ (warning: no ordering)
  - For high throughput for producer:
  - Buffer messages and fill message batches – tune buffer.memory, batch.size, linger.ms
  - Tune socket buffers - receive.buffer.bytes, send.buffer.bytes
- To avoid data loss, turn off auto commit at the source, so that MirrorMaker can control the commits, which it typically does after it receives the ack from the destination cluster. If the producer has acks=all and the destination cluster has min.insync.replicas set to more than 1, the messages are persisted on more than one broker at the destination before the MirrorMaker consumer commits the offset at the source.
- If order is important, you can set retries to 0. Alternatively, for a production environment, set max inflight connections to 1 to ensure that the batches sent out are not committed out of order if a batch fails in the middle. This way, each batch sent is retried until the next batch is sent out. If max.block.ms is not set to the maximum value, and if the producer buffer is full, there can be data loss (depending on some of the other settings). This can block and back-pressure the consumer.

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

Amazon MSK Limits

Amazon MSK has the following limits. If you require higher limits, you can create a support case.

- Up to 90 brokers per account and 15 brokers per cluster
- A minimum of 1 GiB of storage per broker
- A maximum of 16384 GiB per broker
- Up to 100 configurations per account

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

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. 10).

- 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 *Mirroring* (p. 59).

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 *Cluster Operations* (p. 15).

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. 64).

Disk Space is Running Low

See the following best practices for managing disk space: the section called "Monitor Disk Space" (p. 65) and the section called "Adjust the Data Retention Parameters" (p. 65).

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.

Best Practices

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

Right-size Your Cluster

To determine the right size 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.

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:

- Increase broker storage. For information on how to do this, see the section called “Update Broker Storage” (p. 21).
- Reduce the message retention period or log size. For information on how to do that, see the section called “Adjust the Data Retention Parameters” (p. 65).
- 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* (p. 44).

Adjust the 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. 26).

You can also specify retention parameters at the topic level:

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

```
kafka-configs.sh --zookeeper ZooKeeperConnectionString --alter --entity-type topics --
entity-name TopicName --add-config retention.ms=DesiredRetentionTimePeriod
```

- To specify a retention log size per topic, use the following command.

```
kafka-configs.sh --zookeeper ZooKeeperConnectionString --alter --entity-type topics --
entity-name TopicName --add-config retention.bytes=DesiredRetentionLogSize
```

The retention parameters that you specify at the topic level take precedence over cluster-level parameters.

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 *Cluster Operations* (p. 15).

Enable In-Transit Encryption

For information about encryption in transit and how to enable it, see the section called “Encryption in Transit” (p. 35).

Balance Your Cluster

To balance the data load across a cluster, you can use the partition reassignment tool named `kafka-reassign-partitions.sh` to move partitions to different brokers on the same cluster. 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 “Update the Number of Brokers” (p. 23). For information about the partition reassignment tool and process, see Expanding your cluster in the Apache Kafka documentation.

Document History for Amazon MSK Developer Guide

- **Latest documentation update:** May 29, 2019
- **Current API version:** May 29, 2019

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

Change	Description	Date
Support for Apache Kafka 2.2.1.	Amazon MSK now supports Apache Kafka version 2.2.1.	July 31, 2019
General Availability	New features include tagging support, authentication, TLS encryption, configurations, and the ability to update broker storage.	May 30, 2019
Support for Apache Kafka 2.1.0.	Amazon MSK now supports Apache Kafka version 2.1.0.	February 5, 2019
Initial release of Amazon MSK.	This is the first release of the *Amazon MSK Developer Guide*.	November 29, 2018

AWS Glossary

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