Distributed Message Service for Kafka

Best Practices

Issue	01
Date	2025-08-28

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Contents

1 Overview

This section summarizes best practices of Distributed Message Service (DMS) for Kafka in common scenarios. Each practice is given a description and procedure.

Table 1-1 Kafka best practices

Best Practice	Description
Improving Kafka Message Processing Efficiency	This document provides producers and consumers with message suggestions, improving the efficiency and reliability of message sending and consumption.
Interconnecting Logstash to Kafka to Produce and Consume Messages	Kafka instances are available as the input and output sources of Logstash. This document describes how to connect Logstash to Kafka instances for message production and consumption.
Using MirrorMaker to Synchronize Data Across Clusters	MirrorMaker can mirror data from a source cluster to a target cluster. This document describes how to use MirrorMaker to synchronize data between two Kafka instances unidirectionally or bidirectionally.
Handling Message Accumulation	This document describes the causes of message stacking and the handling measures.
Handling Service Overload	This document describes the causes of high CPU usage and full disk space and the handling measures.
Handling Uneven Service Data	This document describes the causes of unbalanced service data and the handling measures.
Setting the Kafka Topic Partition Quantity	This document provides suggestions on setting the partition quantity and typical cases.

Best Practice	Description
Configuring Message Accumulation Monitoring	This document describes how to generate an alarm when the number of stacked messages exceeds a specified threshold. In this way, you can be aware of the service running status in time by SMS or email.
Suggestions on Using DMS for Kafka Securely	This document describes security best practices of using Kafka. It aims to provide a standard guide for overall security capabilities.
Optimizing Consumer Polling	This document describes how to optimize consumer polling in scenarios where real-time message consumption is not required, saving resources when there are few or no messages.

2 Improving Kafka Message Processing Efficiency

During message sending and consumption, Distributed Message Service (DMS) for Kafka, producers, and consumers collaborate to ensure service reliability. In addition, efficiency and accuracy of message sending and consumption improves when developers make proper use of DMS for Kafka topics.

Best practices for message producers and consumers are as follows:

Acknowledging Message Production and Consumption

Message Production

The producer decides whether to re-send a message based on the DMS for Kafka response.

The producer waits for the sending result or asynchronous callback function to determine if the message is successfully sent. If an exception occurs when sending the message, the producer will not receive a success response and must decide whether to re-send the message. If a success response is received, it indicates that the message has been stored in DMS for Kafka.

Message consumption

The consumer acknowledges successful message consumption.

The produced messages are sequentially stored in DMS for Kafka. During consumption, messages stored in DMS for Kafka are obtained in sequence. Consumers obtain messages, consume them, and record the status (successful or failed). The status is then submitted to DMS for Kafka.

During this process, the message consumption status may not be successfully submitted due to an exception. In this case, the corresponding messages will be re-obtained by the consumer in the next message consumption request.

Idempotent Transferring of Message Production and Consumption

To guarantee lossless messaging, DMS for Kafka implements a series of reliability measures. For example, the message synchronization storage mechanism is used to prevent the system and server from being restarted or powered off. The ACK mechanism is used to deal with exceptions that occur during message transmission.

Considering extreme conditions such as network exceptions, you can use DMS for Kafka to design idempotent message transferring in addition to acknowledging message production and consumption.

- If message sending cannot be acknowledged, the producer needs to re-send the message.
- After obtaining a message that has been processed, the consumer needs to notify DMS for Kafka that consumption is successful and ensure that the message is not processed repeatedly.

Producing and Consuming Messages in Batches

It is recommended that messages be sent and consumed in batches to improve efficiency.

Figure 2-1 Messages being produced and consumed in batches

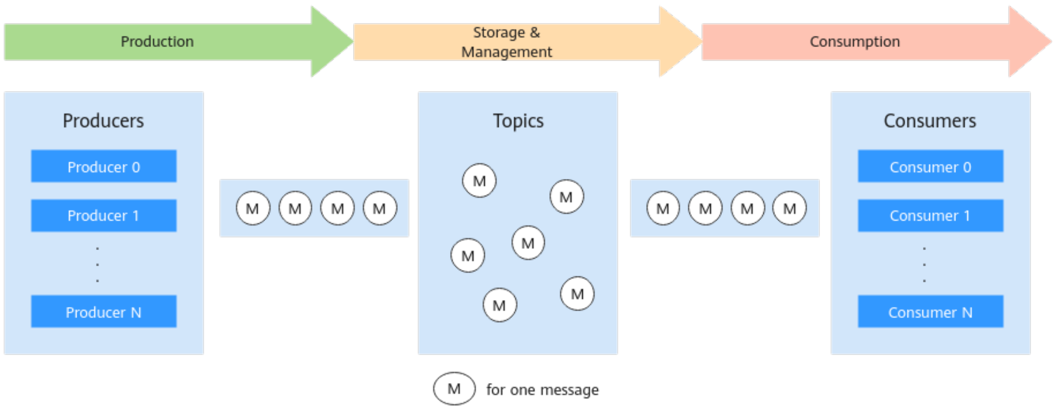
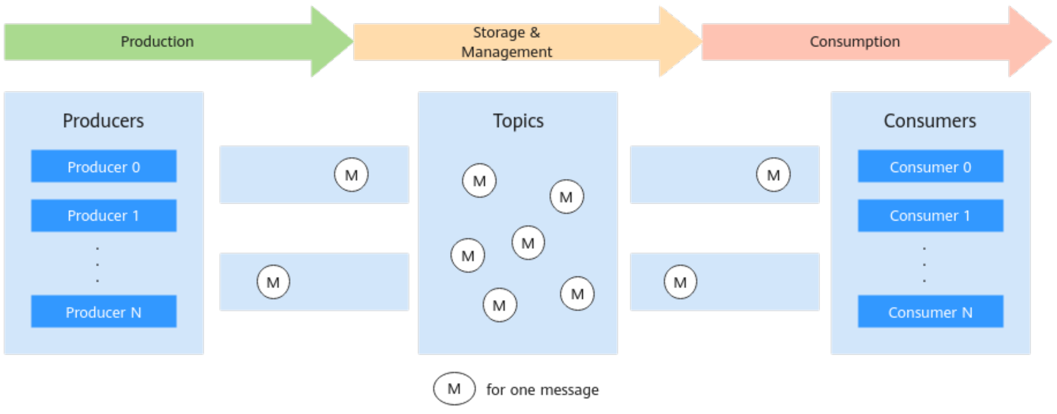


Figure 2-2 Messages being produced and consumed one by one



When consuming messages in batches, consumers need to process and acknowledge messages in the sequence of receiving messages. Therefore, when a message in the batch fails to be consumed, the consumer does not need to consume the remaining messages, and can directly submit consumption acknowledgment of the successfully consumed messages.

Using Consumer Groups to Facilitate O&M

You can use DMS for Kafka as a message management system. Reading message content from topics is helpful to fault locating and service debugging.

When problems occur during message production and consumption, you can create different consumer groups to locate and analyze problems or debug services for interconnecting with other services. To ensure that other services can continue to process messages in topics, you can create a new consumer group to consume and analyze the messages.

3 Interconnecting Logstash to Kafka to Produce and Consume Messages

Overview

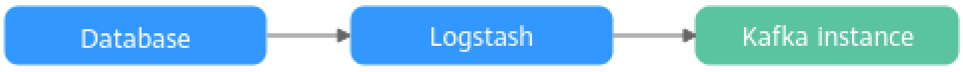
Scenario

Logstash is a free and open server-side data processing pipeline that integrates data from multiple sources, converts it, and then sends it to the specified storage. Kafka is a high-throughput distributed message pub/sub system. It is one of the input and output sources of Logstash. The following describes how to interconnect Logstash with a Kafka instance.

Principle

- The following figure shows Kafka as an output source of Logstash.

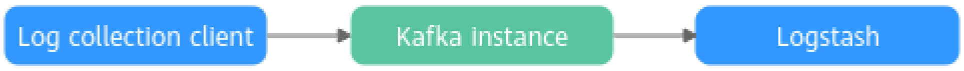
  Figure 3-1 Kafka as an output source of Logstash

  Logstash collects data from the database and sends the data to the Kafka instance for storage. Using a Kafka instance as the Logstash output source can store a large amount of data thanks to the high throughput of Kafka.

- The following figure shows Kafka as an input source of Logstash.

  Figure 3-2 Kafka as an input source of Logstash

  The log collection client sends data to the Kafka instance. Logstash pulls data from the Kafka instance based on its performance. Using a Kafka instance as the Logstash input source can prevent the impact of burst traffic on Logstash, and decouple the log collection client from Logstash to ensure system stability.

Restrictions

Logstash 7.5 and later versions support Kafka Integration Plugin which includes the Kafka input plugin and Kafka output plugin. Kafka input plugin reads data from topics of Kafka instances, and Kafka output plugin writes data to topics of Kafka instances. **Table 3-1** lists the version mapping between Logstash, Kafka Integration Plugin, and Kafka clients. **Ensure that the Kafka client version is later than or equal to the Kafka instance version.**

Table 3-1 Version mapping

Logstash Version	Kafka Integration Plugin Version	Kafka Client Version
8.3–8.8	10.12.0	2.8.1
8.0–8.2	10.9.0–10.10.0	2.5.1
7.12–7.17	10.7.4–10.9.0	2.5.1
7.8–7.11	10.2.0–10.7.1	2.4
7.6–7.7	10.0.1	2.3.0
7.5	10.0.0	2.1.0

Prerequisites

Make the following preparation before implementation.

- **Download Logstash**.
- Prepare a Windows host, install **JDK v1.8.111 or later** and Git Bash on the host, and configure related environment variables.
- **Create a Kafka instance** and **a topic**, and obtain the instance information.

  If both public access and SASL authentication are disabled for the Kafka instance, obtain the information listed in **Table 3-2**.

Table 3-2 Kafka instance information (public access and SASL authentication disabled)

Parameter	How to Obtain
Instance address (private network)	View it in the **Connection** area on the instance details page.
Topic name	On the Kafka console, click your instance. In the left navigation pane, choose **Instance** > **Topics** to view the topic name. The following uses **topic-logstash** as an example.

If public access is disabled and SASL authentication is enabled for the Kafka instance, obtain the information listed in **Table 3-3**.

Table 3-3 Kafka instance information (public access disabled and SASL authentication enabled)

Parameter	How to Obtain
Instance address (private network)	View it in the **Connection** area on the instance details page.
SASL mechanism	View it in the **Connection** area on the instance details page.
Security protocol	View it in the **Connection** area on the instance details page.
Certificate	Click **Download** next to **SSL Certificate** in the **Connection** area on the instance details page. Download and decompress the package to obtain the client certificate file **client.jks**.
SASL username and password	On the Kafka console, click your instance. In the left navigation pane, choose **Instance** > **Users** to view the username. If you have forgotten the password, click **Reset Password**.
Topic name	On the Kafka console, click your instance. In the left navigation pane, choose **Instance** > **Topics** to view the topic name. The following uses **topic-logstash** as an example.

If public access is enabled and SASL authentication is disabled for the Kafka instance, obtain the information listed in **Table 3-4**.

Table 3-4 Kafka instance information (public access enabled and SASL authentication disabled)

Parameter	How to Obtain
Instance address (public network)	View it in the **Connection** area on the instance details page.
Topic name	On the Kafka console, click your instance. In the left navigation pane, choose **Instance** > **Topics** to view the topic name. The following uses **topic-logstash** as an example.

If both public access and SASL authentication are enabled for the Kafka instance, obtain the information listed in **Table 3-5**.

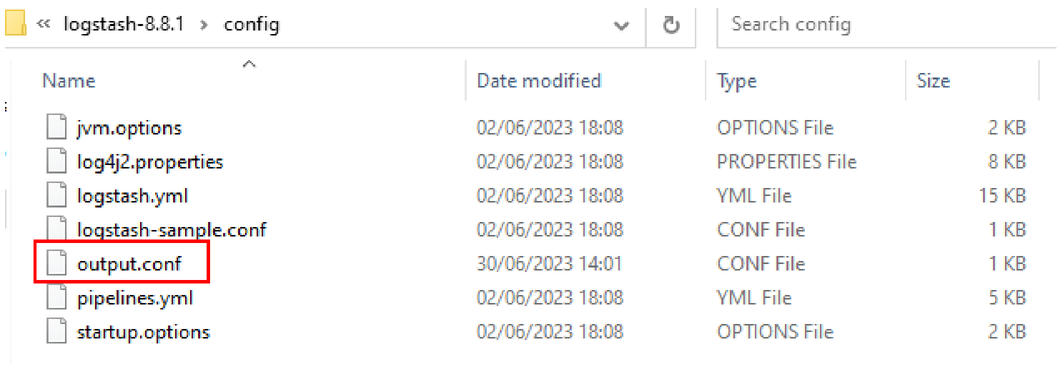
Table 3-5 Kafka instance information (public access and SASL authentication enabled)

Parameter	How to Obtain
Instance address (public network)	View it in the **Connection** area on the instance details page.
SASL mechanism	View it in the **Connection** area on the instance details page.
Security protocol	View it in the **Connection** area on the instance details page.
Certificate	Click **Download** next to **SSL Certificate** in the **Connection** area on the instance details page. Download and decompress the package to obtain the client certificate file **client.jks**.
SASL username and password	On the Kafka console, click your instance. In the left navigation pane, choose **Instance** > **Users** to view the username. If you have forgotten the password, click **Reset Password**.
Topic name	On the Kafka console, click your instance. In the left navigation pane, choose **Instance** > **Topics** to view the topic name. The following uses **topic-logstash** as an example.

Procedure (Kafka Instance as the Logstash Output Source)

Step 1 On the Windows host, decompress the Logstash package, go to the **config** folder, and create the **output.conf** configuration file.

Figure 3-3 Creating the output.conf configuration file



Name	Date modified	Type	Size
jvm.options	02/06/2023 18:08	OPTIONS File	2 KB
log4j2.properties	02/06/2023 18:08	PROPERTIES File	8 KB
logstash.yml	02/06/2023 18:08	YML File	15 KB
logstash-sample.conf	02/06/2023 18:08	CONF File	1 KB
output.conf	30/06/2023 14:01	CONF File	1 KB
pipelines.yml	02/06/2023 18:08	YML File	5 KB
startup.options	02/06/2023 18:08	OPTIONS File	2 KB

Step 2 Add the following content to the **output.conf** file:

```
input {
   stdin {}
}
output {
 kafka {
     bootstrap_servers => "ip1:port1,ip2:port2,ip3:port3"
     topic_id => "topic-logstash"

    # If SASL authentication is disabled, comment out the following options:
     # If the SASL mechanism is PLAIN, configure as follows:
     sasl_mechanism => "PLAIN"
     sasl_jaas_config => "org.apache.kafka.common.security.plain.PlainLoginModule required 
username='username' password='password';"

     # If the SASL mechanism is SCRAM-SHA-512, configure as follows:
     sasl_mechanism => "SCRAM-SHA-512"
     sasl_jaas_config => "org.apache.kafka.common.security.scram.ScramLoginModule required 
username='username' password='password';"

     # If the security protocol is SASL_SSL, configure as follows:
     security_protocol => "SASL_SSL"
     ssl_truststore_location => "C:\\Users\\Desktop\\logstash-8.8.1\\config\\client.jks"
     ssl_truststore_password => "dms@kafka"
     ssl_endpoint_identification_algorithm => ""

    # If the security protocol is SASL_PLAINTEXT, configure as follows:
     security_protocol => "SASL_PLAINTEXT"
     }
}
```

Description:

- **bootstrap_servers**: private or public network connection address of the Kafka instance obtained in **Prerequisites**.
- **topic_id**: topic name obtained in **Prerequisites**.
- **sasl_mechanism**: SASL authentication mechanism.
- **sasl_jaas_config**: SASL JAAS configuration file. Change the SASL username and password to the ones obtained in **Prerequisites** as required.
- **security_protocol**: security protocol used by the Kafka instance.
- **ssl_truststore_location**: location where the SSL certificate is stored.
- **ssl_truststore_password**: server certificate password, which must be set to **dms@kafka** and **cannot be changed**.
- **ssl_endpoint_identification_algorithm**: Indicates whether to verify the certificate domain name. If this option is left blank, the certificate domain name is not verified. **In this example, leave it blank**.

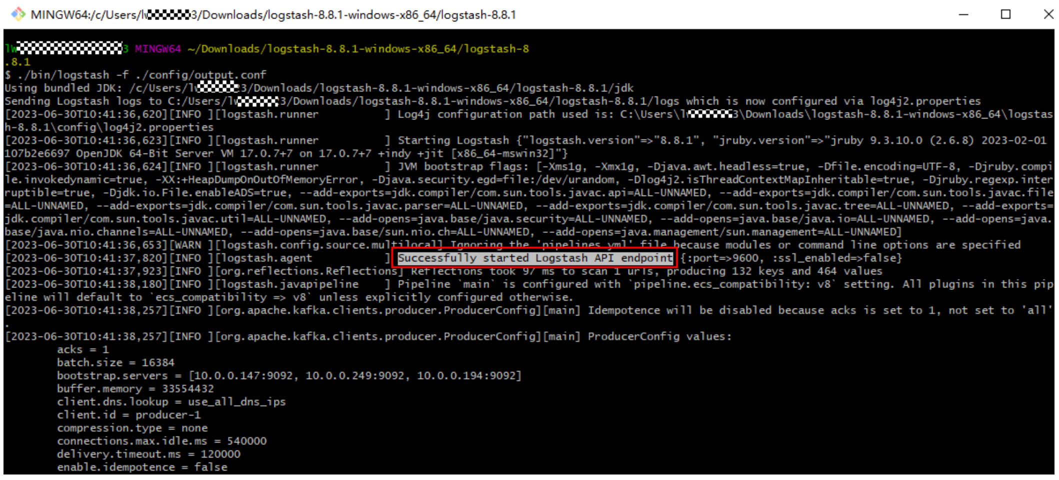
For more information about Kafka output plugin options, see **Kafka output plugin**.

Step 3 Open Git Bash in the **root** directory of the Logstash folder and run the following command to start Logstash:

```
./bin/logstash -f ./config/output.conf
```

If the message "Successfully started Logstash API endpoint" is displayed, Logstash has been started.

Figure 3-4 Starting Logstash

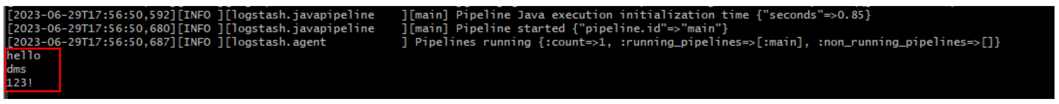


```
MINGW64:/c/Users/l[redacted]3/Downloads/logstash-8.8.1-windows-x86_64/logstash-8.8.1

l[redacted]3 MINGW64 ~/Downloads/logstash-8.8.1-windows-x86_64/logstash-8
.8.1
$ ./bin/logstash -f ./config/output.conf
Using bundled JDK: /c/Users/l[redacted]3/Downloads/logstash-8.8.1-windows-x86_64/logstash-8.8.1/jdk
Sending Logstash logs to C:/Users/l[redacted]3/Downloads/logstash-8.8.1-windows-x86_64/logstash-8.8.1/logs which is now configured via log4j2.properties
[2023-06-30T10:41:36,620][INFO ][logstash.runner          ] Log4j configuration path used is: C:\Users\l[redacted]3\Downloads\logstash-8.8.1-windows-x86_64\logstas
h-8.8.1\config\log4j2.properties
[2023-06-30T10:41:36,623][INFO ][logstash.runner          ] Starting Logstash {"logstash.version"=>"8.8.1", "jruby.version"=>"jruby 9.3.10.0 (2.6.8) 2023-02-01
107b2e6697 OpenJDK 64-Bit Server VM 17.0.7+7 on 17.0.7+7 +indy +jit [x86_64-mswin32]"}
[2023-06-30T10:41:36,624][INFO ][logstash.runner          ] JVM bootstrap flags: [-Xms1g, -Xmx1g, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -Djruby.compi
le.invokedynamic=true, -XX:+HeapDumpOnOutOfMemoryError, -Djava.security.egd=file:/dev/urandom, -Dlog4j2.isThreadContextMapInheritable=true, -Djruby.regexp.inter
ruptible=true, -Djdk.io.File.enableADS=true, --add-exports=jdk.compiler/com.sun.tools.javac.api=ALL-UNNAMED, --add-exports=jdk.compiler/com.sun.tools.javac.file
=ALL-UNNAMED, --add-exports=jdk.compiler/com.sun.tools.javac.parser=ALL-UNNAMED, --add-exports=jdk.compiler/com.sun.tools.javac.tree=ALL-UNNAMED, --add-exports=
jdk.compiler/com.sun.tools.javac.util=ALL-UNNAMED, --add-opens=java.base/java.security=ALL-UNNAMED, --add-opens=java.base/java.io=ALL-UNNAMED, --add-opens=java.
base/java.nio.channels=ALL-UNNAMED, --add-opens=java.base/sun.nio.ch=ALL-UNNAMED, --add-opens=java.management/sun.management=ALL-UNNAMED]
[2023-06-30T10:41:36,653][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file because modules or command line options are specified
[2023-06-30T10:41:37,820][INFO ][logstash.agent           ] Successfully started Logstash API endpoint {:port=>9600, :ssl_enabled=>false}
[2023-06-30T10:41:37,923][INFO ][org.reflections.Reflections] Reflections took 97 ms to scan 1 urls, producing 132 keys and 464 values
[2023-06-30T10:41:38,180][INFO ][logstash.javapipeline    ] Pipeline `main` is configured with `pipeline.ecs_compatibility: v8` setting. All plugins in this pip
eline will default to `ecs_compatibility => v8` unless explicitly configured otherwise.
[2023-06-30T10:41:38,257][INFO ][org.apache.kafka.clients.producer.ProducerConfig][main] Idempotence will be disabled because acks is set to 1, not set to 'all'
.
[2023-06-30T10:41:38,257][INFO ][org.apache.kafka.clients.producer.ProducerConfig][main] ProducerConfig values:
        acks = 1
        batch.size = 16384
        bootstrap.servers = [10.0.0.147:9092, 10.0.0.249:9092, 10.0.0.194:9092]
        buffer.memory = 33554432
        client.dns.lookup = use_all_dns_ips
        client.id = producer-1
        compression.type = none
        connections.max.idle.ms = 540000
        delivery.timeout.ms = 120000
        enable.idempotence = false
```

Step 4 In Logstash, produce messages, as shown in the following figure.

Figure 3-5 Producing messages



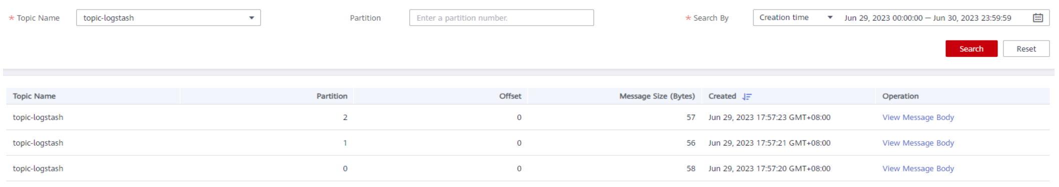
```
[2023-06-29T17:56:50,592][INFO ][logstash.javapipeline    ][main] Pipeline Java execution initialization time {"seconds"=>0.85}
[2023-06-29T17:56:50,680][INFO ][logstash.javapipeline    ][main] Pipeline started {"pipeline.id"=>"main"}
[2023-06-29T17:56:50,687][INFO ][logstash.agent           ] Pipelines running {:count=>1, :running_pipelines=>[:main], :non_running_pipelines=>[]}
hello
dms
123!
```

Step 5 Go to the Kafka console and click your instance.

Step 6 In the left navigation pane, choose **Instance** > **Message Query**.

Step 7 Select **topic-logstash** from the **Topic Name** drop-down list box and click **Search** to query messages.

Figure 3-6 Querying messages



* Topic Name: topic-logstash
Partition: Enter a partition number.
* Search By: Creation time Jun 29, 2023 00:00:00 — Jun 30, 2023 23:59:59

Topic Name	Partition	Offset	Message Size (Bytes)	Created	Operation
topic-logstash	2	0	57	Jun 29, 2023 17:57:23 GMT+08:00	View Message Body
topic-logstash	1	0	56	Jun 29, 2023 17:57:21 GMT+08:00	View Message Body
topic-logstash	0	0	58	Jun 29, 2023 17:57:20 GMT+08:00	View Message Body

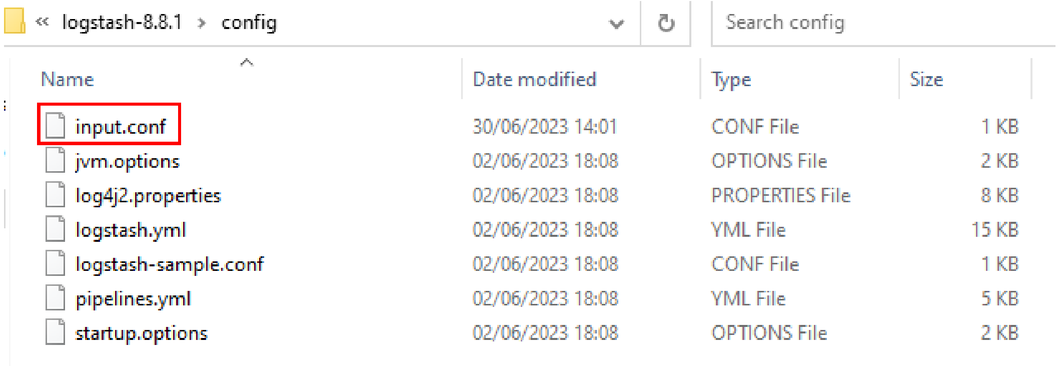
As shown in **Figure 3-6**, the Kafka output plugin of Logstash has written data to **topic-logstash** of the Kafka instance.

----End

Procedure (Kafka Instance as the Logstash Input Source)

Step 1 On the Windows host, decompress the Logstash package, go to the **config** folder, and create the **input.conf** configuration file.

Figure 3-7 Creating the input.conf configuration file



Name	Date modified	Type	Size
input.conf	30/06/2023 14:01	CONF File	1 KB
jvm.options	02/06/2023 18:08	OPTIONS File	2 KB
log4j2.properties	02/06/2023 18:08	PROPERTIES File	8 KB
logstash.yml	02/06/2023 18:08	YML File	15 KB
logstash-sample.conf	02/06/2023 18:08	CONF File	1 KB
pipelines.yml	02/06/2023 18:08	YML File	5 KB
startup.options	02/06/2023 18:08	OPTIONS File	2 KB

Step 2 Add the following content to the **input.conf** file to connect to the Kafka instance:

```
input {
 kafka {
     bootstrap_servers => "ip1:port1,ip2:port2,ip3:port3"
     group_id => "logstash_group"
     topic_id => "topic-logstash"
     auto_offset_reset => "earliest"

    # If SASL authentication is disabled, comment out the following options:
     #If the SASL mechanism is PLAIN, configure as follows:
     sasl_mechanism => "PLAIN"
     sasl_jaas_config => "org.apache.kafka.common.security.plain.PlainLoginModule required
username='username' password='password';"

     # If the SASL mechanism is SCRAM-SHA-512, configure as follows:
     sasl_mechanism => "SCRAM-SHA-512"
     sasl_jaas_config => "org.apache.kafka.common.security.scram.ScramLoginModule required
username='username' password='password';"

     # If the security protocol is SASL_SSL, configure as follows:
     security_protocol => "SASL_SSL"
     ssl_truststore_location => "C:\\Users\\Desktop\\logstash-8.8.1\\config\\client.jks"
     ssl_truststore_password => "dms@kafka"
     ssl_endpoint_identification_algorithm => ""

    # If the security protocol is SASL_PLAINTEXT, configure as follows:
     security_protocol => "SASL_PLAINTEXT"
     }
}
output {
 stdout{codec=>rubydebug}
}
```

Description:

- **bootstrap_servers**: private or public network connection address of the Kafka instance obtained in **Prerequisites**.
- **group_id**: consumer group name.
- **topic_id**: topic name obtained in **Prerequisites**.
- **auto_offset_reset**: consumers' consumption policy. The value **latest** indicates that the offset is automatically reset to the latest. The value **earliest** indicates that the offset is automatically reset to the earliest. The value **none** indicates that an exception is thrown to the consumer. This parameter is set to **earliest** in this example.
- **sasl_mechanism**: SASL authentication mechanism.
- **sasl_jaas_config**: SASL JAAS configuration file. Change the SASL username and password to the ones obtained in **Prerequisites** as required.

- **security_protocol**: security protocol used by the Kafka instance.
- **ssl_truststore_location**: location where the SSL certificate is stored.
- **ssl_truststore_password**: server certificate password, which must be set to **dms@kafka** and **cannot be changed**.
- **ssl_endpoint_identification_algorithm**: Indicates whether to verify the certificate domain name. If this option is left blank, the certificate domain name is not verified. **In this example, leave it blank**.

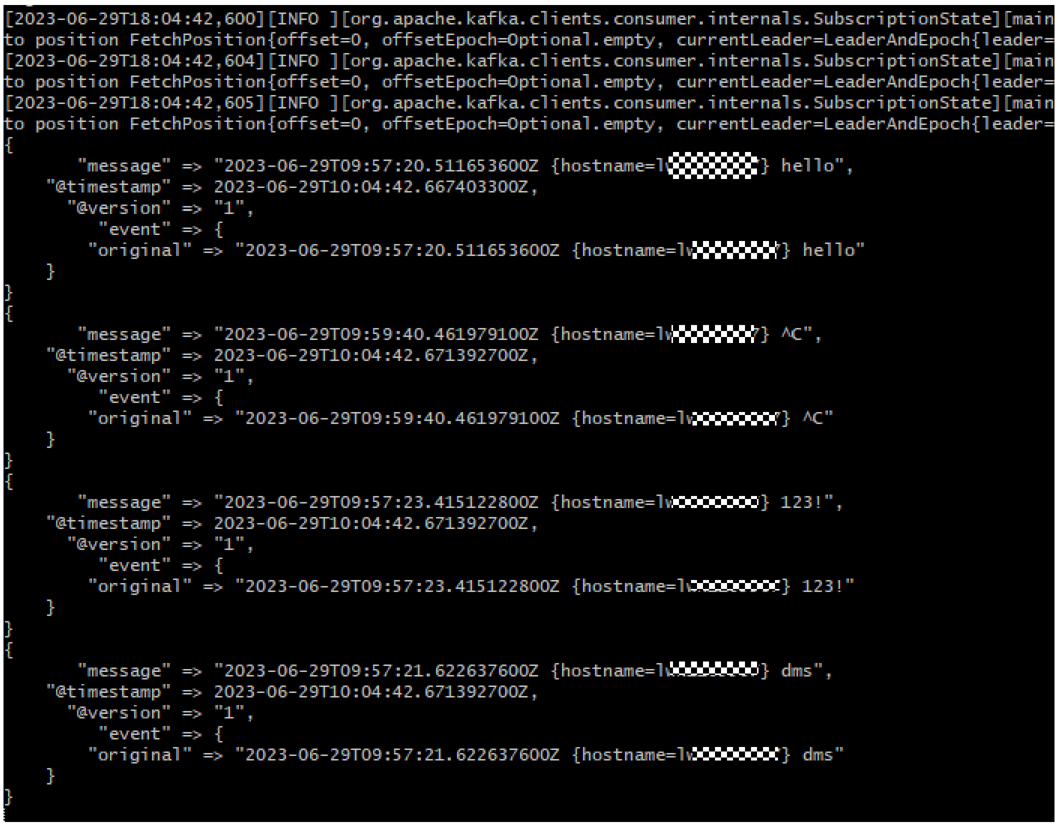
For more information about Kafka input plugin options, see **Kafka input plugin**.

Step 3 Open Git Bash in the **root** directory of the Logstash folder and run the following command to start Logstash:

```
./bin/logstash -f ./config/input.conf
```

After Logstash is started successfully, the Kafka input plugin automatically reads data from **topic-logstash** of the Kafka instance, as shown in the following figure.

Figure 3-8 Logstash reading data from topic-logstash



```
[2023-06-29T18:04:42,600][INFO ][org.apache.kafka.clients.consumer.internals.SubscriptionState][main
to position FetchPosition{offset=0, offsetEpoch=Optional.empty, currentLeader=LeaderAndEpoch{leader=
[2023-06-29T18:04:42,604][INFO ][org.apache.kafka.clients.consumer.internals.SubscriptionState][main
to position FetchPosition{offset=0, offsetEpoch=Optional.empty, currentLeader=LeaderAndEpoch{leader=
[2023-06-29T18:04:42,605][INFO ][org.apache.kafka.clients.consumer.internals.SubscriptionState][main
to position FetchPosition{offset=0, offsetEpoch=Optional.empty, currentLeader=LeaderAndEpoch{leader=
{
       "message" => "2023-06-29T09:57:20.511653600Z {hostname=l[redacted]} hello",
    "@timestamp" => 2023-06-29T10:04:42.667403300Z,
      "@version" => "1",
         "event" => {
        "original" => "2023-06-29T09:57:20.511653600Z {hostname=l[redacted]} hello"
    }
}
{
       "message" => "2023-06-29T09:59:40.461979100Z {hostname=l[redacted]} ^C",
    "@timestamp" => 2023-06-29T10:04:42.671392700Z,
      "@version" => "1",
         "event" => {
        "original" => "2023-06-29T09:59:40.461979100Z {hostname=l[redacted]} ^C"
    }
}
{
       "message" => "2023-06-29T09:57:23.415122800Z {hostname=l[redacted]} 123!",
    "@timestamp" => 2023-06-29T10:04:42.671392700Z,
      "@version" => "1",
         "event" => {
        "original" => "2023-06-29T09:57:23.415122800Z {hostname=l[redacted]} 123!"
    }
}
{
       "message" => "2023-06-29T09:57:21.622637600Z {hostname=l[redacted]} dms",
    "@timestamp" => 2023-06-29T10:04:42.671392700Z,
      "@version" => "1",
         "event" => {
        "original" => "2023-06-29T09:57:21.622637600Z {hostname=l[redacted]} dms"
    }
}
```

----End

4 Using MirrorMaker to Synchronize Data Across Clusters

Overview

Scenario

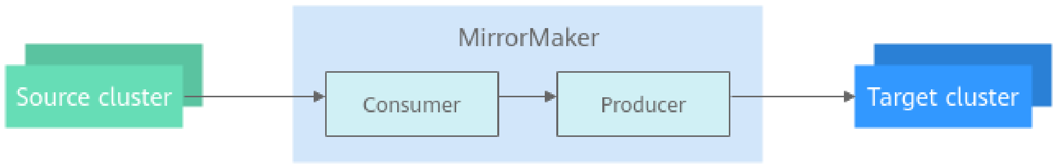
In the following scenarios, MirrorMaker can be used to synchronize data between different Kafka clusters to ensure the availability and reliability of the clusters:

- Backup and disaster recovery: An enterprise has multiple data centers. To prevent service unavailability caused by a fault in one data center, cluster data is synchronously backed up in multiple data centers.
- Cluster migration: As enterprises migrate services to the cloud, data in on-premises clusters must be synchronized with that in cloud clusters to ensure service continuity.

Principle

MirrorMaker can be used to mirror data from the source cluster to the target cluster. As shown in **Figure 4-1**, in essence, MirrorMaker first consumes data from the source cluster and then produces the consumed data to the target cluster. For more information about MirrorMaker, see **Mirroring data between clusters**.

Figure 4-1 MirrorMaker



Restrictions

- The IP addresses and port numbers of the nodes in the source cluster cannot be the same as those of the nodes in the target cluster. Otherwise, data will be replicated infinitely in a topic.
- Use MirrorMaker to synchronize data between at least two clusters. If there is only one cluster, data will be replicated infinitely in a topic.

Procedure

Step 1 Buy an ECS that can communicate with the source and target clusters. For details, see **Purchasing an ECS**.

Step 2 Log in to the ECS, install JDK, and add the following contents to **.bash_profile** in the home directory to configure the environment variables **JAVA_HOME** and **PATH**. In this command, **/opt/java/jdk1.8.0_151** is the JDK installation path. Change it to the path where you install JDK.

```
export JAVA_HOME=/opt/java/jdk1.8.0_151
export PATH=$JAVA_HOME/bin:$PATH
```

Run the **source .bash_profile** command for the modification to take effect.

NOTE

Use Oracle JDK instead of ECS's default JDK (for example, OpenJDK), because ECS's default JDK may not be suitable. Obtain Oracle JDK 1.8.111 or later from **Oracle's official website**.

Step 3 Download the **binary software package of Kafka 3.3.1**.

```
wget https://archive.apache.org/dist/kafka/3.3.1/kafka_2.12-3.3.1.tgz
```

Step 4 Decompress the binary software package.

```
tar -zxvf kafka_2.12-3.3.1.tgz
```

Step 5 Go to the binary software package directory and specify the IP addresses and ports of the source and target clusters and other parameters in the **connect-mirror-maker.properties** configuration file in the **config** directory.

```
# Specify two clusters.
clusters = A, B
A.bootstrap.servers = A_host1:A_port, A_host2:A_port, A_host3:A_port
B.bootstrap.servers = B_host1:B_port, B_host2:B_port, B_host3:B_port

# Specify the data synchronization direction. The data can be synchronized unidirectionally or bidirectionally.
A->B.enabled = true

# Specify the topics to be synchronized. Regular expressions are supported. By default, all topics are
replicated, for example, foo-.*.
A->B.topics = .*

# If the following two configurations are enabled, clusters A and B replicate data with each other.
#B->A.enabled = true
#B->A.topics = .*

# Specify the number of replicas. If multiple topics need to be synchronized and their replica quantities are
different, create topics with the same name and replica quantity before starting MirrorMaker.
replication.factor=3

# Specify the consumer offset synchronization direction (unidirectionally or bidirectionally).
A->B.sync.group.offsets.enabled=true

############################# Internal Topic Settings  #############################
# The replication factor for mm2 internal topics "heartbeats", "B.checkpoints.internal" and
# "mm2-offset-syncs.B.internal"
# In the test environment, the value can be 1. In the production environment, it is recommended that the
value be greater than 1, for example, 3.
checkpoints.topic.replication.factor=3
heartbeats.topic.replication.factor=3
offset-syncs.topic.replication.factor=3

# The replication factor for connect internal topics "mm2-configs.B.internal", "mm2-offsets.B.internal" and
# "mm2-status.B.internal"
# In the test environment, the value can be 1. In the production environment, it is recommended that the
value be greater than 1, for example, 3.
```

```
offset.storage.replication.factor=3
status.storage.replication.factor=3
config.storage.replication.factor=3

# customize as needed
# replication.policy.separator = _
# sync.topic.acls.enabled = false
# emit.heartbeats.interval.seconds = 5
# Match the topic name of the target cluster to that of the source.
# replication.policy.class = org.apache.kafka.connect.mirror.IdentityReplicationPolicy
```

Step 6 In the binary software package directory, start MirrorMaker to synchronize data.

```
./bin/connect-mirror-maker.sh config/connect-mirror-maker.properties
```

Step 7 (Optional) If a topic is created in the source cluster after MirrorMaker has been started, and the topic data needs to be synchronized, restart MirrorMaker. For details about how to restart MirrorMaker, see **Step 6**. You can also add configurations listed in **Table 4-1** to periodically synchronize new topics without restarting MirrorMaker. **refresh.topics.interval.seconds** is mandatory. Other parameters are optional.

Table 4-1 MirrorMaker configurations

Parameter	Default Value	Description
sync.topic.configs.enabled	true	Whether to monitor the source cluster for configuration changes.
sync.topic.acls.enabled	true	Whether to monitor the source cluster for ACL changes.
emit.heartbeats.enabled	true	Whether to let the connector send heartbeats periodically.
emit.heartbeats.interval.seconds	5 seconds	Heartbeat frequency.
emit.checkpoints.enabled	true	Whether to let the connector periodically send the consumer offset information.
emit.checkpoints.interval.seconds	5 seconds	Checkpoint frequency.
refresh.topics.enabled	true	Whether to let the connector periodically check for new topics.
refresh.topics.interval.seconds	5 seconds	Frequency of checking for new topics in the source cluster.
refresh.groups.enabled	true	Whether to let the connector periodically check for new consumer groups.

Parameter	Default Value	Description
refresh.groups.interval.seconds	5 seconds	Frequency of checking for new consumer groups in the source cluster.
replication.policy.class	org.apache.kafka.connect.mirror.DefaultReplicationPolicy	Use LegacyReplicationPolicy to imitate MirrorMaker of an earlier version.
heartbeats.topic.retention.ms	One day	Used when heartbeat topics are created for the first time.
checkpoints.topic.retention.ms	One day	Used when checkpoint topics are created for the first time.
offset.syncs.topic.retention.ms	max long	Used when offset sync topics are created for the first time.

----End

Verifying Data Synchronization

Step 1 View the topic list in the target cluster to check whether there are source topics.

NOTE

The default value of **replication.policy.class** is **org.apache.kafka.connect.mirror.DefaultReplicationPolicy**, which adds a prefix (for example, "A.") to target topic names to differentiate them from the source. This setting helps MirrorMaker avoid circular replication. To mirror the target and source topic names, set **replication.policy.class** to **org.apache.kafka.connect.mirror.IdentityReplicationPolicy**.

Step 2 Produce and consume messages in the source cluster, view the consumption progress in the target cluster, and check whether data has been synchronized from the source cluster to the target cluster.

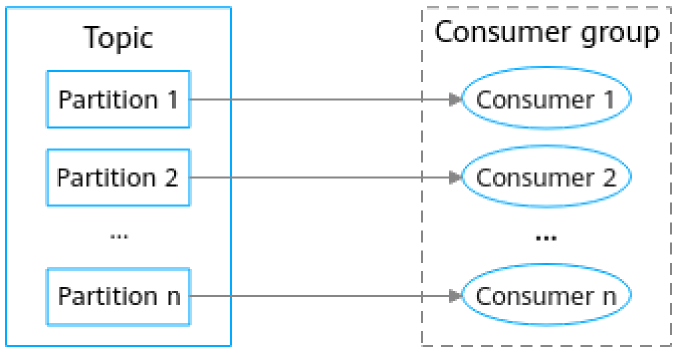
If the target cluster is a Huawei Cloud Kafka instance, view the consumption progress on the **Instance** > **Consumer Groups** page.

----End

5 Handling Message Accumulation

Overview

Kafka divides each topic into multiple partitions for distributed message storage. Within the same consumer group, each consumer can consume multiple partitions at the same time, but each partition can be consumed by only one consumer at a time.



Unprocessed messages accumulate if the client's consumption is slower than the server's sending. Accumulated messages cannot be consumed in time.

Causes of accumulation

The following are some main causes:

- Producers produce messages too fast for consumers to keep up.
- Incapable consumers (low concurrency and long processing) cause lower efficiency of consumption than production.
- Abnormal consumers (faulty and network error) cannot consume messages.
- Improper topic partitions, or no consumption in new partitions.
- Frequent topic reassignment reduces consumption efficiency.

Solution

Accumulation can be avoided by the consumer, producer, and server.

- **Consumer**

  - Add consumers (for consumption concurrency) based on actual needs. Use the same number of consumers as the number of partitions, or ensure that the number of partitions is an integer multiple of the number of consumers.
  - Speed up consumption by optimizing the consumer processing logic (less complicated computing, API invoking, and database reading).
  - Increase the number of messages in each poll: Polling/Processing speed should be equal to or higher than the production speed.
- **Producer**

  Attach a random suffix to each message key so that messages can be evenly distributed in partitions.

  NOTE

  In actual scenarios, attaching a random suffix to each message key compromises global message sequence. Decide whether a suffix is required by your service.
- **Server**
  - Set the number of topic partitions properly. Add partitions without affecting processing efficiency.
  - Stop production when messages are accumulating or forward them to other topics.

6 Handling Service Overload

Overview

High CPU usage and full disks indicate overloaded Kafka services.

- High CPU usage leads to low system performance and high risk of hardware damage.
- If a disk is full, the Kafka log content stored on it goes offline. Then, the disk's partition replicas cannot be read or written, reducing partition availability and fault tolerance. The leader partition switches to another broker, adding load to the broker.

Causes of high CPU usage

- There are too many data operation threads: **num.io.threads**, **num.network.threads**, and **num.replica.fetchers**.
- Improper partitions. One broker carries all production and consumption services.

Causes of full disk

- Current disk space no longer meets the needs of the rapidly increasing service data volume.
- Unbalanced broker disk usage. The produced messages are all in one partition, taking up the partition's disk.
- The time to live (TTL) set for a topic is too long. Old data takes too much disk space.

Solution

Handling high CPU usage:

- Optimize the parameters configuration for threads **num.io.threads**, **num.network.threads**, and **num.replica.fetchers**.
  - Set the number of **num.io.threads** and the number of **num.network.threads** threads to multiples of the disk quantity. Do not exceed the number of CPU cores
  - Set the number of **num.replica.fetchers** threads to smaller than or equal to 5.

- Set topic partitions properly. Set the number of partitions to multiples of the number of brokers.
- Attach a random suffix to each message key so that messages can be evenly distributed in partitions.

  NOTE

  In actual scenarios, attaching a random suffix to each message key compromises global message sequence. Decide whether a suffix is required by your service.

Handling full disk:

- Increase the disk space.
- Migrate partitions from the full disk to other disks on the broker.
- Set proper topic TTL for less occupation of old data.
- If CPU resources are sufficient, compress the data with compression algorithms.

  Common compression algorithms include ZIP, gzip, Sappy, and LZ4. You need to consider the data compression rate and duration when selecting compression algorithms. Generally, an algorithm with a higher compression rate consumes more time. For systems with high performance requirements, select algorithms with quick compression, such as LZ4. For systems with high compression rate requirements, select algorithms with high compression rate, such as gzip.

  Configure the **compression.type** parameter on producers to specify a compression algorithm.

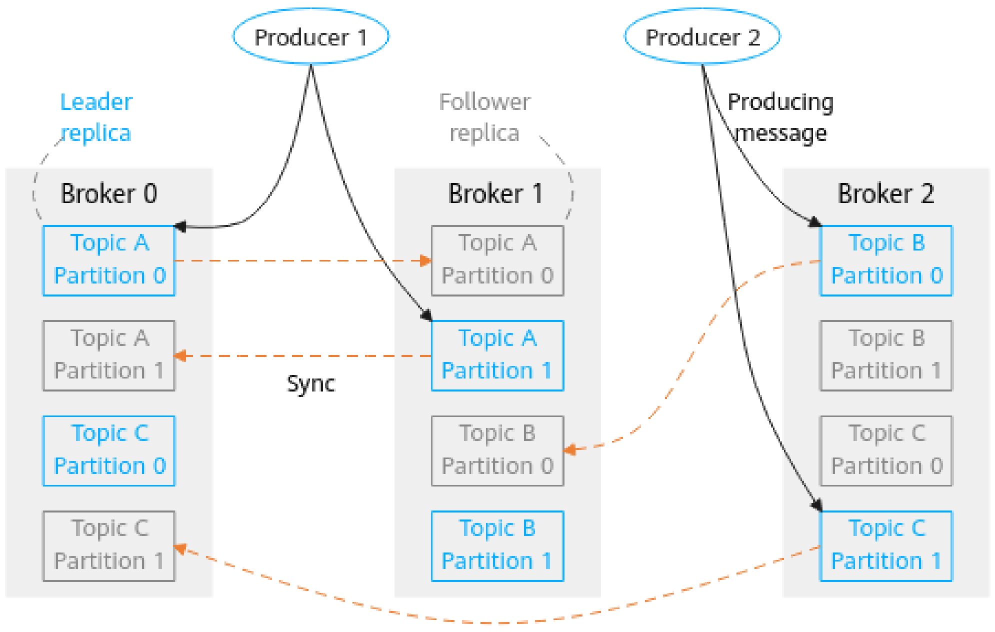
```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("acks", "all");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
// Enable GZIP.
props.put("compression.type", "gzip");

Producer<String, String> producer = new KafkaProducer<>(props);
```

7 Handling Uneven Service Data

Overview

Kafka divides each topic into multiple partitions for distributed message storage. Each partition has one or more replicas distributed on different brokers. Each replica stores a copy of full data. Messages are synchronized among replicas. The following figure shows the relationships between topics, partitions, replicas, and brokers.



Uneven service data among brokers and partitions may happen, leading to low performance of Kafka clusters and low resource utilization.

Causes of uneven service data

- The traffic of some topics is much heavier than that of others.
- Producers specified partitions when sending messages, leaving unspecified partitions empty.
- Producers specified message keys to send messages to specific partitions.
- The system re-implements flawed partition allocation policies.

- There are new Kafka brokers with no partitions allocated.
- Cluster changes lead to switches and migration of leader replicas, causing data on some brokers to increase.

Solution

Handling uneven service data:

- Optimize the topic design. For a topic with considerable data, the data can be split across topics.
- Producers evenly send messages across partitions.
- When creating topics, distribute leader replicas across brokers.
- Kafka features partition reassignment. You can reassign replicas to different brokers to balance load among brokers. For details, see **Reassigning Partitions**.

8 Setting the Kafka Topic Partition Quantity

Overview

The topic partition quantity is one of the key configurations that affect the Kafka performance, throughput, and concurrency. If the quantity is too small, the concurrency of producers is limited, and consumers may be idle. On the contrary, clusters' performance may deteriorate and monitoring intervals may increase. Therefore, it is important to set a proper partition quantity.

Implementation

In terms of the cluster scale, consumer quantity, and scalability, suggestions are as follows:

Table 8-1 Setting the partition quantity

Factor	Suggestion
Cluster scale	Use an **integer multiple of the broker quantity**. In other cases, partitions on each broker may be uneven, causing unbalanced cluster load.
Consumer quantity	Use the **consumer quantity** or an **integer multiple** of it. A partition is consumed by only one consumer. When the partition quantity is less than the consumer quantity, some consumers are idle.
Scalability	For services whose traffic fluctuates greatly, use the **consumer quantity during peak hours**. Temporarily adding partitions is too slow to troubleshoot stacked messages in existing partitions. Use the consumer quantity during peak hours in advance.

Typical Cases

Table 8-2 Typical cases

Symptom	Cause Analysis	Suggestion
ClickHouse is used to consume Kafka. The topic partition quantity is 12, the ClickHouse node quantity is 10, and consumer_num is set to 12 in the ClickHouse parameters (that is, 120 consumers on 10 nodes). Only one ClickHouse node is consuming Kafka, and other ones are idle.	The 12 consumers who are consuming messages are reassigned to all partitions of the topic. These consumers are on the same ClickHouse node. As a result, consumers on other nodes have no message to consume.	Use a partition quantity that is greater than or equal to the consumer quantity. You are advised to set it to an integer multiple of the consumer quantity so that each consumer can be assigned with the same number of partitions for load balancing.
The topic quantity is 3. As the service traffic increases, increase the 3 brokers of the Kafka instance to 6 ones. The new brokers do not have partitions and services are still on the original brokers.	The number of partitions was not increased accordingly and partition reassignment was not performed. As a result, partitions were still on the original brokers.	Use an integer multiple of the broker quantity as the partition quantity. Then, perform partition reassignment to ensure that partitions are even on brokers.

9 Configuring Message Accumulation Monitoring

Overview

Unprocessed messages accumulate if the client's consumption is slower than the server's sending. Accumulated messages cannot be consumed in time.

Configure alarm rules so that you will be notified when the number of accumulated messages in a consumer group exceeds the threshold. The procedure described in this section can also be applied to setting alarm rules for **other metrics**.

Prerequisites

You have **purchased a Kafka instance**, **created a topic**, and there are available messages.

Procedure

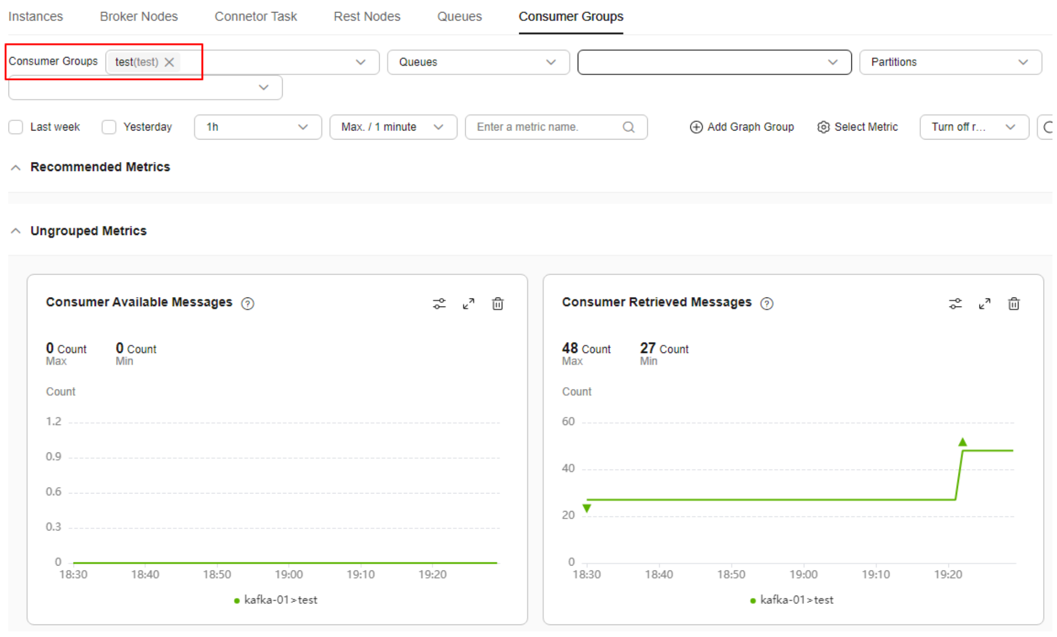
Step 1 Log in to the **Kafka console**.

Step 2 Click the name of the desired instance. The instance details page is displayed.

Step 3 In the navigation pane on the left, choose **Monitoring** > **Monitoring Details**.

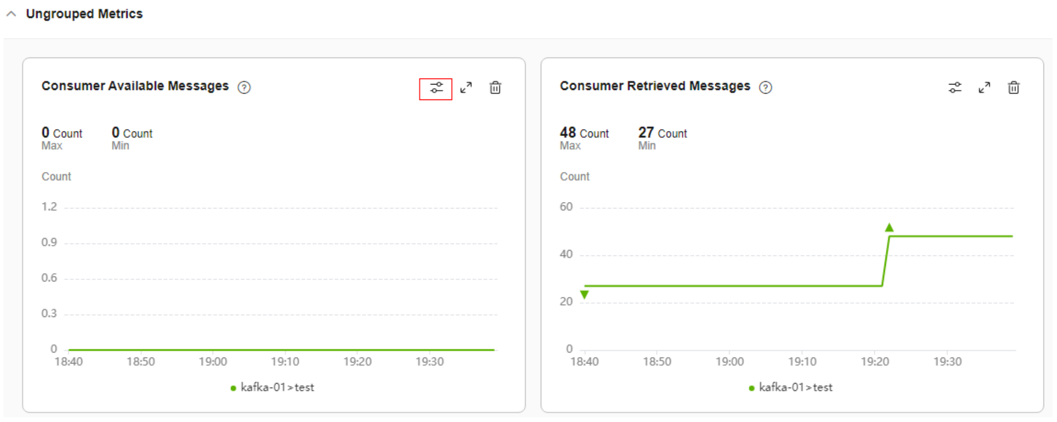
Step 4 On the **By Consumer Group** tab page, select the consumer group for which you want to create an alarm rule.

Figure 9-1 Selecting a consumer group



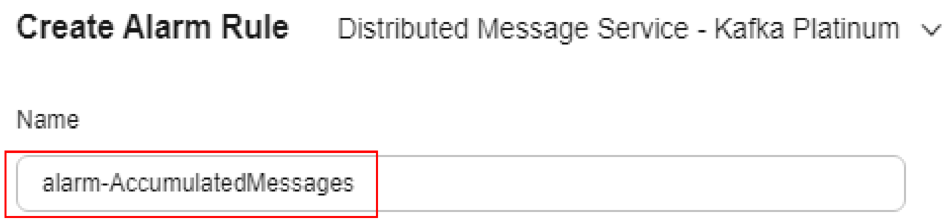
Step 5 Hover the mouse pointer over **Consumer Available Messages** and click ⚙.

Figure 9-2 Consumer available messages chart



Step 6 On the **Create Alarm Rule** page, enter **Name**. The alarm name can contain only letters, digits, underscores (_), and hyphens (-).

Figure 9-3 Configuring the basic information of the alarm rule


Create Alarm Rule Distributed Message Service - Kafka Platinum

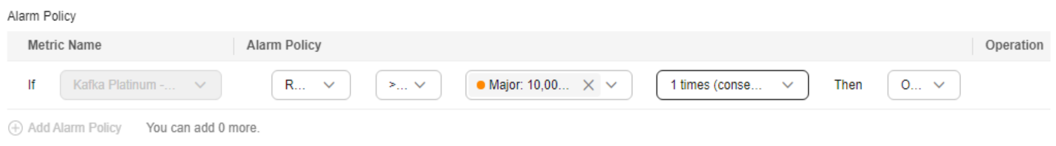
Name

alarm-AccumulatedMessages

Step 7 On the **Create Alarm Rule** page, configure **Monitoring Scope**. Retain the current settings.

Step 8 On the **Create Alarm Rule** page, configure **Alarm Policy**.

Figure 9-4 Configuring the alarm policy


Alarm Policy

Metric Name	Alarm Policy						Operation
If Kafka Platinum -...	R...	>...	Major: 10,00...	1 times (conse...	Then	O...	

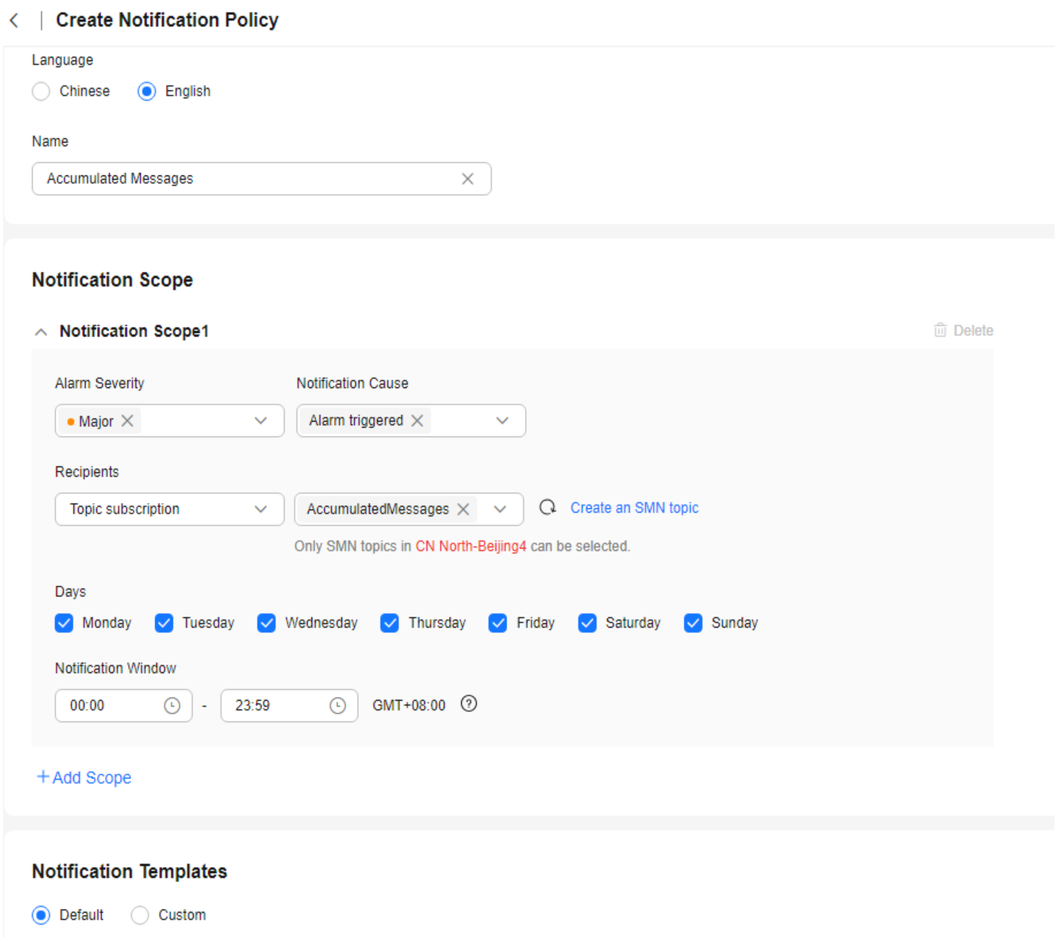
Add Alarm Policy You can add 0 more.

Alarm policy: A major alarm is generated if the number of raw data records is greater than or equal to 10,000 for one consecutive time. The alarm is notified once a day.

Step 9 On the **Create Alarm Rule** page, click **Create Notification Policy**.

Step 10 Set the notification policy and click **OK**.

Figure 9-5 Creating a notification policy


Create Notification Policy

Language
Chinese [✓ English]

Name
Accumulated Messages

Notification Scope

Notification Scope1 Delete

Alarm Severity: Major
Notification Cause: Alarm triggered

Recipients: Topic subscription, AccumulatedMessages
Create an SMN topic
Only SMN topics in CN North-Beijing4 can be selected.

Days
✓ Monday ✓ Tuesday ✓ Wednesday ✓ Thursday ✓ Friday ✓ Saturday ✓ Sunday

Notification Window
00:00 - 23:59 GMT+08:00

+Add Scope

Notification Templates
[✓ Default] Custom

Table 9-1 Notification policy parameters

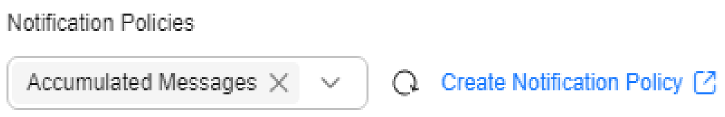
Parameter	Description
Language	Select a language for the notification policy.

Parameter	Description
Name	Set the name of the notification policy.
Alarm Severity	Select **Major**.
Notification Cause	Select **Alarm triggered** which indicates that a notification is sent when an alarm is triggered.
Recipients	Select **Topic subscriptions** and click **Create**. On the SMN console, **create a topic** and **add a subscription**. After the alarm notification topic is created, go back to the **Create Alarm Rule** page, click ↻ to make the created topic available for selection. **NOTE** After the subscription is added, the corresponding subscription endpoint will receive a subscription notification. You need to confirm the subscription so that the endpoint can receive alarm notifications.
Days	Retain the default settings. That is, if an alarm is triggered, a notification is sent every day.
Notification Window	Cloud Eye sends notifications only within the validity period specified in the alarm rule. Retain the default settings.
Notification Templates	Select **Default**.

Step 11 After the notification policy is created, the **Create Alarm Rule** page is displayed.

Step 12 Click ↻ next to **Notification Policies** and select the new policy from the drop-down list box.

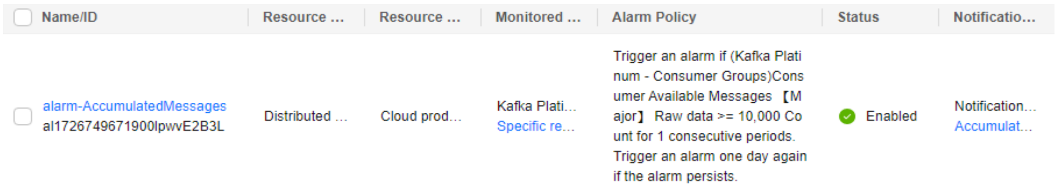
Figure 9-6 Setting a notification policy


Notification Policies
Accumulated Messages

Step 13 Click **Create**.

After the alarm rule is created, you can view it on the **Alarm Management** > **Alarm Rules** page.

Figure 9-7 Viewing the new alarm rule



Name/ID	Resource ...	Resource ...	Monitored ...	Alarm Policy	Status	Notificatio...
alarm-AccumulatedMessages al1726749671900lpwvE2B3L	Distributed ...	Cloud prod...	Kafka Plati... Specific re...	Trigger an alarm if (Kafka Platinum - Consumer Groups)Consumer Available Messages 【Major】 Raw data >= 10,000 Count for 1 consecutive periods. Trigger an alarm one day again if the alarm persists.	Enabled	Notification... Accumulat...

----End

10 Suggestions on Using DMS for Kafka Securely

Huawei Cloud and you share the responsibility for security. Huawei Cloud ensures the security of cloud services for a secure cloud. As a tenant, you should utilize the security capabilities provided by cloud services to protect data and use the cloud securely. For details, see **Shared Responsibilities**.

This section guides you on how to enhance overall DMS for Kafka security through security best practices. You can improve the security of your DMS for Kafka resources by continuously monitoring their security status, combining multiple security capabilities provided by DMS for Kafka, and protecting data stored in DMS for Kafka from leakage and tampering both at rest and in transit.

Configure security settings from the following dimensions to meet your service needs.

- **Protecting Data Through Access Control**
- **Transmission Encryption with SSL**
- **Do Not Store Sensitive Data**
- **Data Restoration and Disaster Recovery**
- **Checking for Abnormal Data Access**
- **Using the Latest SDKs for Better Experience and Security**

Protecting Data Through Access Control

1. **Set only the minimum permissions for IAM users with different roles to prevent data leakage or misoperations caused by excessive permissions.**

   To better isolate and manage permissions, you are advised to configure an independent IAM administrator and grant them the permission to manage IAM policies. The IAM administrator can create different user groups based on your service requirements. User groups correspond to different data access scenarios. By adding users to user groups and binding IAM policies to user groups, the IAM administrator can grant different data access permissions to employees in different departments based on the principle of least privilege. For details, see **Permissions Management**.

2. **Configure a security group to protect your data from abnormal reads or other operations.**

Tenants can configure inbound and outbound traffic rules for a security group to regulate network access to an instance, and prevent unauthorized exposure to third parties. For more information, see **How Do I Select and Configure a Security Group?**. Do not set the source to 0.0.0.0/0 in the inbound rules of a security group.

3. **Configure a password for accessing your Kafka instance (by enabling SASL) to prevent unauthorized clients from operating it by mistake.**
4. **Enable multi-factor authentication for sensitive operations to protect your data from accidental deletion.**

   DMS for Kafka offers sensitive operation protection to enhance the security of your data. With this function enabled, the system authenticates the identity before sensitive operations such as instance deletion are performed. For more information, see **Critical Operation Protection**.

Transmission Encryption with SSL

To prevent data from breaches or damage during transmission, access DMS for Kafka using SSL encryption. To do so, set the Kafka security protocol to **SASL_SSL**.

Do Not Store Sensitive Data

Currently, DMS for Kafka does not support data encryption. Do not store sensitive data into message queues.

Data Restoration and Disaster Recovery

Build restoration and disaster recovery (DR) capabilities in advance to prevent data from being deleted or damaged by mistake in abnormal data processing scenarios.

1. **Configure multiple replicas for a topic to quickly restore data in abnormal scenarios.**

   Kafka instances support single or multiple replicas of a topic for high availability (HA). With multiple replicas for a Kafka instance, synchronous replication is established and maintained. When an instance broker is faulty, the instance automatically reassigns its leader partition to another available broker.
2. **Use multiple AZs for data DR.**

   Cluster Kafka instances can be deployed, and DR can be supported across AZs. If a Kafka instance uses multiple AZs, the instance service continues when one AZ is faulty.

Checking for Abnormal Data Access

1. **Enable Cloud Trace Service (CTS) to record all Kafka access operations for future audit.**

   CTS records operations on the cloud resources in your account. You can use the logs generated by CTS to perform security analysis, track resource changes, audit compliance, and locate faults.

   After you enable CTS and configure a tracker, CTS can record management and data traces of Kafka for auditing. For more information, see **Viewing Kafka Audit Logs**.

2. **Use Cloud Eye for real-time monitoring and alarm reporting.**

   To monitor Kafka instances, Huawei Cloud provides Cloud Eye. Cloud Eye supports automatic real-time monitoring, alarms, and notification for requests and traffic in Kafka instances.

   Cloud Eye starts monitoring your Kafka instance once it is created, so you do not need to enable Cloud Eye. For more information, see **Kafka Metrics**.

Using the Latest SDKs for Better Experience and Security

Upgrade to the latest version of SDKs to enhance the protection of your data and Kafka usage. Download the latest SDK in your desired language from **SDK Overview**.

11 Optimizing Consumer Polling

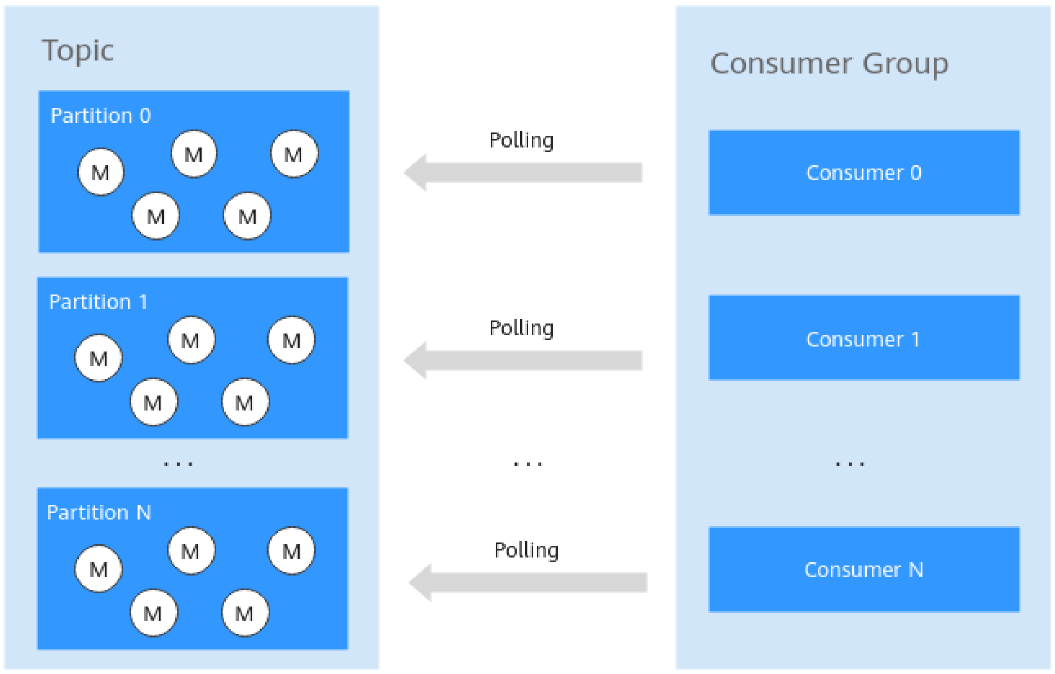
Overview

Scenario

In the native Kafka SDK provided by DMS for Kafka, consumers can customize the duration for pulling messages. To pull messages for a long time, consumers only need to set the parameter of the poll(long) method to a proper value. However, such persistent connections may cause pressure on the client and the server, especially when the number of partitions is large and multiple threads are enabled for each consumer.

As shown in **Figure 11-1**, the topic contains multiple partitions, and multiple consumers in the consumer group consume the resources at the same time. Each thread is in a persistent connection. When there are few or no messages in the topic, the connection persists, and all consumers pull messages continuously, which causes a waste of resources.

Figure 11-1 Multi-thread consumption of Kafka consumers

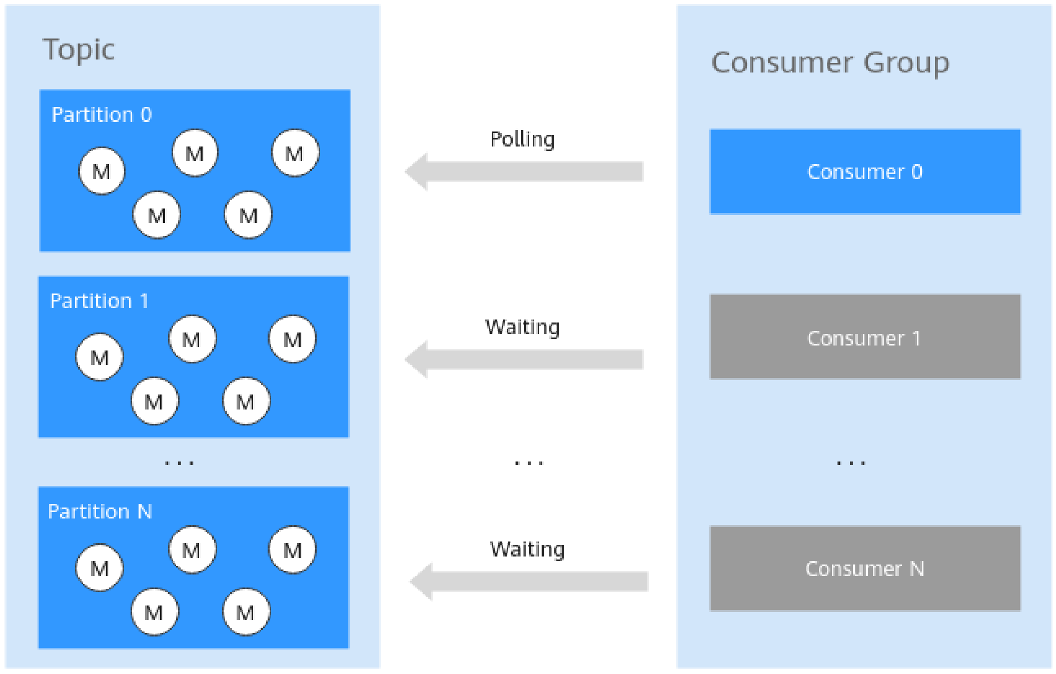


Solution

When multiple threads are enabled for concurrent access, if there is no message in the topic, only one thread is required to poll for messages in each partition. When a message is found by the polling thread, other threads can be woken up to consume the message for quick responses, as shown in **Figure 11-2**.

This solution is applicable to scenarios with low requirements on real-time message consumption. If quasi-real-time message consumption is required, it is recommended that all consumers be in the active state.

Figure 11-2 Optimized multi-thread consumption solution



NOTE

The number of consumers and the number of partitions are not necessarily the same. The poll (long) method of Kafka helps implement the functions such as message acquisition, partition balancing, and heartbeat detection between consumers and Kafka brokers.

Therefore, in scenarios where the requirements on real-time message consumption are low and there is a small number of messages, some consumers can be in the wait state.

Sample Code

NOTICE

The following describes only the code related to wake-up and sleep of the consumer thread. To run the entire demo, download the complete **sample code package** and refer to the **Developer Guide** for deploying and running the code.

- **Sample code for consuming messages:**

```
package com.huawei.dms.kafka;
```

```
import java.io.IOException;
import java.util.Arrays;
import java.util.Collection;
import java.util.Iterator;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.TopicPartition;
import org.apache.log4j.Logger;

public class DmsKafkaConsumeDemo
{
    private static Logger logger = Logger.getLogger(DmsKafkaProduceDemo.class);

    public static void WorkerFunc(int workerId, KafkaConsumer<String, String> kafkaConsumer)
throws IOException
    {
        Properties consumerConfig = Config.getConsumerConfig();
        RecordReceiver receiver = new RecordReceiver(workerId, kafkaConsumer,
consumerConfig.getProperty("topic"));
        while (true)
        {
            ConsumerRecords<String, String> records = receiver.receiveMessage();
            Iterator<ConsumerRecord<String, String>> iter = records.iterator();
            while (iter.hasNext())
            {
                ConsumerRecord<String, String> cr = iter.next();
                System.out.println("Thread" + workerId + " recievedrecords" + cr.value());
                logger.info("Thread" + workerId + " recievedrecords" + cr.value());

            }

        }
    }

    public static KafkaConsumer<String, String> getConsumer() throws IOException
    {
        Properties consumerConfig = Config.getConsumerConfig();

        consumerConfig.put("ssl.truststore.location", Config.getTrustStorePath());
        System.setProperty("java.security.auth.login.config", Config.getSaslConfig());

        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(consumerConfig);
        kafkaConsumer.subscribe(Arrays.asList(consumerConfig.getProperty("topic")),
                new ConsumerRebalanceListener()
                {
                    @Override
                    public void onPartitionsRevoked(Collection<TopicPartition> arg0)
                    {

                    }

                    @Override
                    public void onPartitionsAssigned(Collection<TopicPartition> tps)
                    {

                    }
                });
        return kafkaConsumer;
    }

    public static void main(String[] args) throws IOException
    {

        // Create a consumer for the current consumer group.
        final KafkaConsumer<String, String> consumer1 = getConsumer();
        Thread thread1 = new Thread(new Runnable()
        {
            public void run()
            {
                try
                {
                    WorkerFunc(1, consumer1);
                }
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });
        final KafkaConsumer<String, String> consumer2 = getConsumer();

        Thread thread2 = new Thread(new Runnable()
        {
            public void run()
            {
                try
                {
                    WorkerFunc(2, consumer2);
                }
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });
        final KafkaConsumer<String, String> consumer3 = getConsumer();

        Thread thread3 = new Thread(new Runnable()
        {
            public void run()
            {
                try
                {
                    WorkerFunc(3, consumer3);
                }
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });

        // Start threads.
        thread1.start();
        thread2.start();
        thread3.start();

        try
        {
            Thread.sleep(5000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
       //Add threads.
        try
        {
            thread1.join();
            thread2.join();
            thread3.join();
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

- **Sample code for managing consumer threads:**

  The sample code provides only simple design ideas. Developers can optimize the thread wake-up and sleep mechanisms based on actual scenarios.

  NOTE

  topicName is the name of the topic.

```
package com.huawei.dms.kafka;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.apache.log4j.Logger;

public class RecordReceiver
{
    private static Logger logger = Logger.getLogger(DmsKafkaProduceDemo.class);

    // Interval time of polling
    public static final int WAIT_SECONDS = 10 * 1000;

    protected static final Map<String, Object> sLockObjMap = new HashMap<String, Object>();

    protected static Map<String, Boolean> sPollingMap = new ConcurrentHashMap<String, 
Boolean>();

    protected Object lockObj;

    protected String topicName;

    protected KafkaConsumer<String, String> kafkaConsumer;

    protected int workerId;

    public RecordReceiver(int id, KafkaConsumer<String, String> kafkaConsumer, String queue)
    {
        this.kafkaConsumer = kafkaConsumer;
        this.topicName = queue;
        this.workerId = id;

        synchronized (sLockObjMap)
        {
            lockObj = sLockObjMap.get(topicName);
            if (lockObj == null)
            {
                lockObj = new Object();
                sLockObjMap.put(topicName, lockObj);
            }
        }
    }

    public boolean setPolling()
    {
        synchronized (lockObj)
        {
            Boolean ret = sPollingMap.get(topicName);
            if (ret == null || !ret)
            {
                sPollingMap.put(topicName, true);
                return true;
            }
            return false;
        }
    }

    // Wake up all threads.
    public void clearPolling()
    {
        synchronized (lockObj)
        {
            sPollingMap.put(topicName, false);
            lockObj.notifyAll();
            System.out.println("Everyone WakeUp and Work!");
            logger.info("Everyone WakeUp and Work!");
        }
    }

    public ConsumerRecords<String, String> receiveMessage()
    {
        boolean polling = false;
        while (true)
        {
            // Check the poll status of threads and hibernate the threads when necessary.
            synchronized (lockObj)
            {
                Boolean p = sPollingMap.get(topicName);
                if (p != null && p)
                {
                    try
                    {
                        System.out.println("Thread" + workerId + " Have a nice sleep!");
                        logger.info("Thread" + workerId +" Have a nice sleep!");
                        polling = false;
                        lockObj.wait();
                    }
                    catch (InterruptedException e)
                    {
                        System.out.println("MessageReceiver Interrupted! topicName is " + topicName);
                        logger.error("MessageReceiver Interrupted! topicName is "+topicName);

                        return null;
                    }
                }
            }

            // Start to consume and wake up other threads when necessary.
            try
            {
                ConsumerRecords<String, String> Records = null;
                if (!polling)
                {
                    Records = kafkaConsumer.poll(100);
                    if (Records.count() == 0)
                    {
                        polling = true;
                        continue;
                    }
                }
                else
                {
                    if (setPolling())
                    {
                        System.out.println("Thread" + workerId + " Polling!");
                        logger.info("Thread " + workerId + " Polling!");
                    }
                    else
                    {
                        continue;
                    }
                    do
                    {
                        System.out.println("Thread" + workerId + " KEEP Poll records!");
                        logger.info("Thread" + workerId + " KEEP Poll records!");
                        try
                        {
                            Records = kafkaConsumer.poll(WAIT_SECONDS);
                        }
                        catch (Exception e)
                        {
                            System.out.println("Exception Happened when polling records: " + e);
                            logger.error("Exception Happened when polling records: " + e);

                        }
                    } while (Records.count()==0);
                    clearPolling();
                }
                // Acknowledge message consumption.
                kafkaConsumer.commitSync();
                return Records;
            }
            catch (Exception e)
            {
                System.out.println("Exception Happened when poll records: " + e);
                logger.error("Exception Happened when poll records: " + e);
            }
        }
    }
}
```

Running Result

```
[2018-01-25 22:40:51,841] INFO Thread 2 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:40:51,841] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:40:52,122] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:40:52,169] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:40:52,169] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:40:52,216] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:40:52,325] INFO Thread 2 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:40:52,325] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:40:54,947] INFO Thread1 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:40:54,979] INFO Thread3 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:41:32,347] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:42,353] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:47,816] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:41:47,847] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:47,925] INFO Thread 3 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:41:47,925] INFO Thread1 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:41:47,925] INFO Thread3 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:47,957] INFO Thread2 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
```

```
[2018-01-25 22:41:48,472] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:41:48,503] INFO Thread3 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,518] INFO Thread1 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,550] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,597] INFO Thread1 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,659] INFO Thread 2 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:41:48,659] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:48,675] INFO Thread3 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,675] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:41:48,706] INFO Thread 1 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:41:48,706] INFO Thread1 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
```