Distributed Message Service for Kafka

Melhores práticas

Edição	01
Data	2024-09-09

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Marcas registadas e permissões

e outras marcas registadas da Huawei são marcas registadas da Huawei Technologies Co., Ltd. Todos as outras marcas registadas e os nomes registados mencionados neste documento são propriedade dos seus respectivos detentores.

Aviso

Os produtos, os serviços e as funcionalidades adquiridos são estipulados pelo contrato estabelecido entre a Huawei Cloud e o cliente. Os produtos, os serviços e as funcionalidades descritos neste documento, no todo ou em parte, podem não estar dentro do âmbito de aquisição ou do âmbito de uso. Salvo especificação em contrário no contrato, todas as declarações, informações e recomendações neste documento são fornecidas "TAL COMO ESTÃO" sem garantias ou representações de qualquer tipo, sejam expressas ou implícitas.

As informações contidas neste documento estão sujeitas a alterações sem aviso prévio. Foram feitos todos os esforços na preparação deste documento para assegurar a exatidão do conteúdo, mas todas as declarações, informações e recomendações contidas neste documento não constituem uma garantia de qualquer tipo, expressa ou implícita.

Huawei Cloud Computing Technologies Co., Ltd.

Endereço: Huawei Cloud Data Center, Rua Jiaoxinggong
Avenida Qianzhong
Novo Distrito de Gui'an
Guizhou 550029
República Popular da China

Site: https://www.huaweicloud.com/intl/pt-br/

Índice

1 Melhoria da eficiência do processamento de mensagens

Durante o envio e o consumo de mensagens, o DMS for Kafka, os produtores e os consumidores colaboram para garantir a confiabilidade do serviço. Além disso, a eficiência e a precisão do envio e do consumo de mensagens melhoram quando os desenvolvedores fazem uso adequado dos tópicos do DMS for Kafka.

As melhores práticas para produtores e consumidores de mensagens são as seguintes:

Reconhecimento de produção e consumo de mensagens

Produção de mensagens (envio)

O produtor decide se deseja reenviar uma mensagem com base na resposta do DMS for Kafka.

O produtor aguarda o resultado de envio ou a função de retorno de chamada assíncrona para determinar se a mensagem foi enviada com sucesso. Se ocorrer uma exceção ao enviar a mensagem, o produtor não receberá uma resposta bem-sucedida e deverá decidir se deseja reenviar a mensagem. Se uma resposta bem-sucedida for recebida, ela indica que a mensagem foi armazenada no DMS for Kafka.

Consumo de mensagens

O consumidor reconhece o consumo bem-sucedido da mensagem.

As mensagens produzidas são armazenadas sequencialmente no DMS for Kafka. Durante o consumo, as mensagens armazenadas no DMS for Kafka são obtidas em sequência. Os consumidores obtêm mensagens, consomem-nas e registram o status (com sucesso ou com falha). O status é então enviado ao DMS for Kafka.

Durante esse processo, o status de consumo de mensagem pode não ser enviado com sucesso devido a uma exceção. Nesse caso, as mensagens correspondentes serão obtidas novamente pelo consumidor na próxima solicitação de consumo de mensagens.

Transferência idempotente de produção e consumo de mensagens

Para garantir mensagens sem perdas, o DMS for Kafka implementa uma série de medidas de confiabilidade. Por exemplo, o mecanismo de armazenamento de sincronização de mensagens é usado para impedir que o sistema e o servidor sejam reiniciados ou desligados. O

mecanismo ACK é usado para lidar com exceções que ocorrem durante a transmissão de mensagens.

Considerando condições extremas, como exceções de rede, você pode usar o DMS for Kafka para projetar o envio e o consumo de mensagens, além de reconhecer a produção e o consumo de mensagens.

- Se o envio da mensagem não puder ser reconhecido, o produtor precisará reenviar a mensagem.
- Depois de consumir uma mensagem que foi processada, o consumidor precisa notificar o DMS for Kafka de que o consumo foi bem-sucedido e garantir que a mensagem não seja processada repetidamente.

Produção e consumo de mensagens em lotes

Recomenda-se que as mensagens sejam enviadas e consumidas em lotes para melhorar a eficiência.

Figura 1-1 Mensagens sendo produzidas (enviadas) e consumidas em lotes

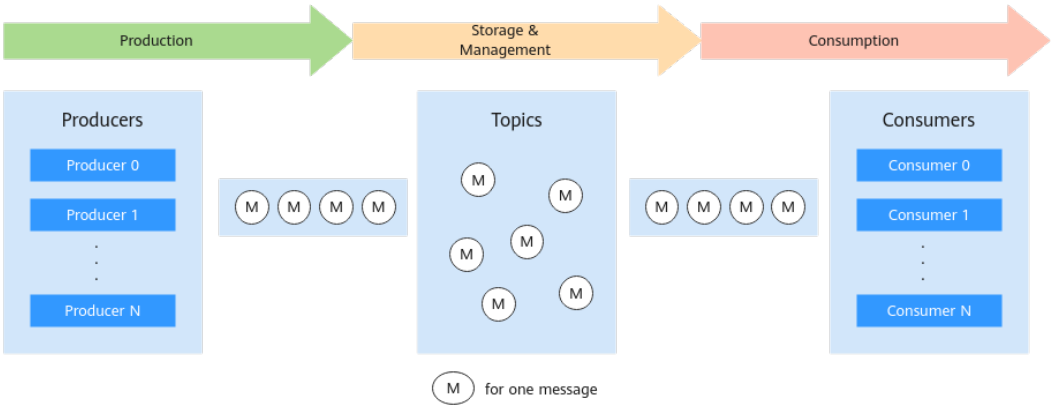
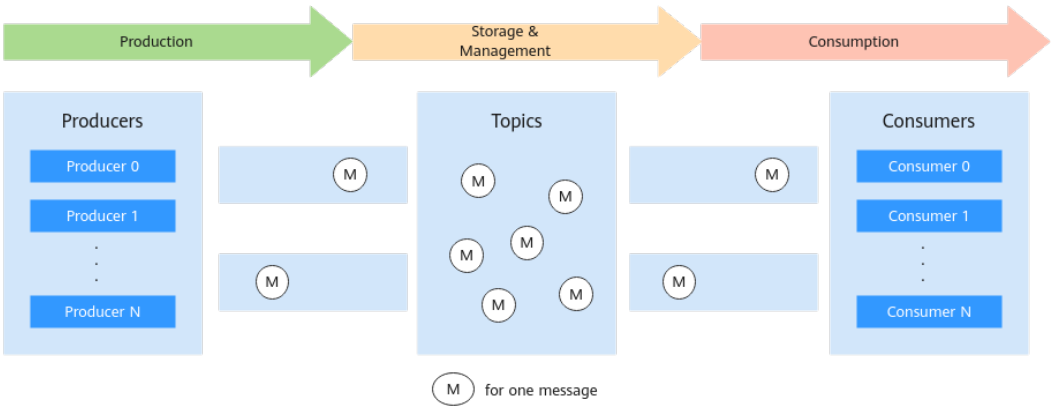


Figura 1-2 Mensagens sendo produzidas (enviadas) e consumidas uma a uma



Ao consumir mensagens em lotes, os consumidores precisam processar e reconhecer mensagens na sequência de recebimento de mensagens. Portanto, quando uma mensagem no lote não ser consumida, o consumidor não precisa consumir as mensagens restantes e pode enviar diretamente o reconhecimento de consumo das mensagens consumidas com sucesso.

Uso de grupos de consumidores para facilitar a O&M

Você pode usar o DMS for Kafka como um sistema de gerenciamento de mensagens. A leitura do conteúdo das mensagens dos tópicos é útil para a localização de falhas e a depuração de serviços.

Quando ocorrem problemas durante a produção e o consumo de mensagens, você pode criar grupos de consumidores diferentes para localizar e analisar problemas ou depurar serviços para interconexão com outros serviços. Para garantir que outros serviços possam continuar a processar mensagens em tópicos, você pode criar um novo grupo de consumidores para consumir e analisar as mensagens.

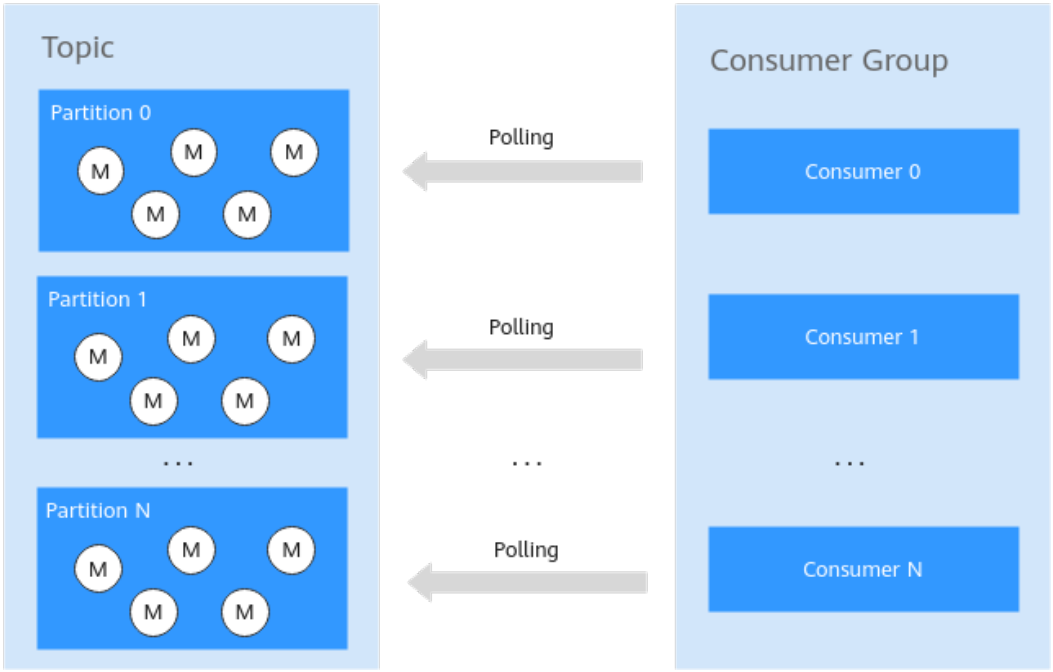
2 Otimização da sondagem de mensagens dos consumidores de DMS for Kafka

Visão geral

No SDK do Kafka nativo fornecido pelo DMS for Kafka, os consumidores podem personalizar a duração da extração de mensagens. Para extrair mensagens por um longo período, os consumidores só precisam definir o parâmetro do método poll(long) para um valor adequado. No entanto, essas conexões persistentes podem causar pressão sobre o cliente e o servidor, especialmente quando o número de partições é grande e vários threads são ativados para cada consumidor.

Conforme mostrado em **Figura 2-1**, o tópico contém várias partições e vários consumidores no grupo de consumidores consomem os recursos ao mesmo tempo. Cada thread está em uma conexão persistente. Quando há poucas ou nenhuma mensagem no tópico, a conexão persiste e todos os consumidores extraem mensagens continuamente, o que causa um desperdício de recursos.

Figura 2-1 Consumo multithread de consumidores do Kafka

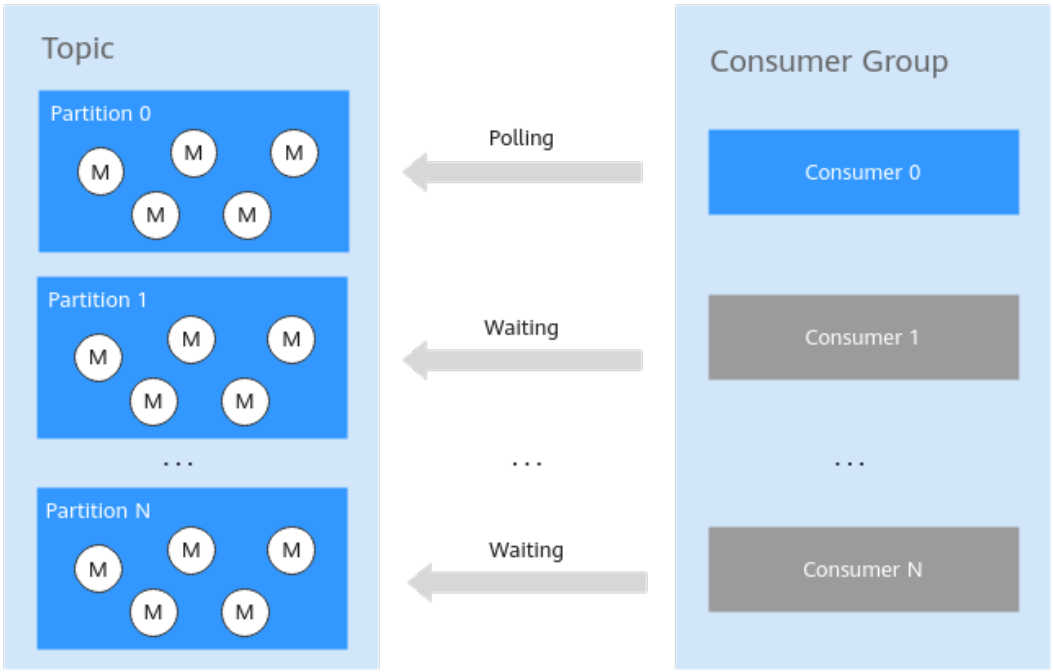


Solução de otimização

Quando vários threads são ativados para acesso simultâneo, se não houver nenhuma mensagem no tópico, apenas um thread será necessário para pesquisar mensagens em cada partição. Quando uma mensagem é encontrada pelo thread de sondagem, outros threads podem ser ativados para consumir a mensagem para respostas rápidas, como mostrado em **Figura 2-2**.

Essa solução é aplicável a cenários com baixos requisitos de consumo de mensagens em tempo real. Se for necessário o consumo de mensagens em tempo quase real, recomenda-se que todos os consumidores estejam no estado ativo.

Figura 2-2 Solução otimizada de consumo multithread



NOTA

O número de consumidores e o número de partições não são necessariamente os mesmos. O método poll (long) do Kafka ajuda a implementar funções como aquisição de mensagens, balanceamento de partições e detecção de pulsação entre os consumidores e os agentes do Kafka.

Portanto, em cenários em que os requisitos de consumo de mensagens em tempo real são baixos e há um pequeno número de mensagens, alguns consumidores podem estar no estado de espera.

Código de exemplo

AVISO

A seguir descreve apenas o código relacionado à ativação e suspensão do thread do consumidor. Para executar toda a demonstração, faça o download do **pacote de código de exemplo** completo e consulte o **Guia de desenvolvedor** para implementar e executar o código.

Exemplo de código para consumir mensagens:

```
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();
```



```
        }
    }
}
```

Exemplo de código para gerenciamento de threads do consumidor:

O código de exemplo fornece apenas ideias de design simples. Os desenvolvedores podem otimizar os mecanismos de ativação e suspensão do thread com base em cenários reais.

```
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);
            }
        }
    }
}
```

NOTA

topicName é o nome do tópico.

Resultados de execução do exemplo de código

```
[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)
```


3 Interconexão do Logstash com o Kafka

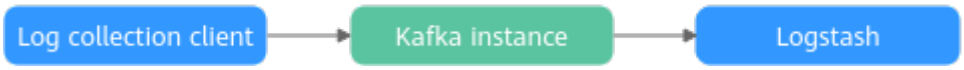
Cenário

O Logstash é um pipeline de processamento de dados gratuito e aberto no lado do servidor que integra dados de várias fontes, converte-os e, em seguida, envia-os para o armazenamento especificado. Kafka é um sistema pub/sub de mensagens distribuídas de alto rendimento. É uma das fontes de entrada e saída do Logstash. A seguir descreve como interconectar o Logstash com uma instância do Kafka.

Arquitetura da solução

- A figura a seguir mostra o Kafka como uma fonte de entrada do Logstash.

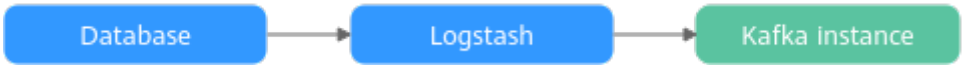
  Figura 3-1 Kafka como uma fonte de entrada do Logstash

  O cliente de coleta de log envia dados para a instância do Kafka. O Logstash extrai dados da instância do Kafka com base em seu desempenho. O uso de uma instância do Kafka como a fonte de entrada do Logstash pode evitar o impacto do tráfego de intermitência no Logstash e desacoplar o cliente de coleta de logs do Logstash para garantir a estabilidade do sistema.

- A figura a seguir mostra o Kafka como uma fonte de saída do Logstash.

  Figura 3-2 Kafka como uma fonte de saída do Logstash

  O Logstash coleta dados do banco de dados e envia os dados para a instância do Kafka para armazenamento. O uso de uma instância do Kafka como fonte de saída do Logstash pode armazenar uma grande quantidade de dados graças à alta taxa de transferência do Kafka.

Restrições

O Logstash 7.5 e versões posteriores suportam o Kafka Integration Plugin, que inclui o plug-in de entrada do Kafka e o plug-in de saída do Kafka. O plug-in de entrada do Kafka lê dados

de tópicos de instâncias do Kafka e o plug-in de saída do Kafka grava dados em tópicos de instâncias do Kafka. Tabela 3-1 lista o mapeamento de versão entre o Logstash, o Kafka Integration Plugin e os clientes do Kafka. **Certifique-se de que a versão do cliente do Kafka seja posterior ou igual à versão da instância do Kafka.**

Tabela 3-1 Mapeamento de versões

Versão do Logstash	Versão do Kafka Integration Plugin	Versão do cliente do Kafka
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

Pré-requisitos

Faça a seguinte preparação antes da implementação.

- **Baixe Logstash**.
- Prepare um host do Windows, instale o **JDK v1.8.111 ou posterior** e o Git Bash no host e configure variáveis de ambiente relacionadas.
- **Crie uma instância do Kafka** e **um tópico** e obtenha as informações da instância.

  Se o acesso público e a autenticação SASL estiverem desativados para a instância do Kafka, obtenha as informações listadas em **Tabela 3-2**.

  Tabela 3-2 Informações da instância do Kafka (acesso público e autenticação SASL desativados)

Parâmetro	Como obter
Instance address (private network)	Visualize-o na área **Connection** na página de detalhes da instância.
Topic name	No console do Kafka, clique em sua instância. No painel de navegação esquerdo, escolha **Topics** para exibir o nome do tópico. O seguinte usa **topic-logstash** como um exemplo.

  Se o acesso público estiver desativado e a autenticação SASL estiver ativada para a instância do Kafka, obtenha as informações listadas em **Tabela 3-3**.

Tabela 3-3 Informações da instância do Kafka (acesso público desativado e autenticação SASL ativada)

Parâmetro	Como obter
Instance address (private network)	Visualize-o na área **Connection** na página de detalhes da instância.
SASL mechanism	Visualize-o na área **Connection** na página de detalhes da instância.
Security protocol	Visualize-o na área **Connection** na página de detalhes da instância.
Certificate	Clique em **Download** ao lado de **SSL Certificate** na área **Connection** na página de detalhes da instância. Faça o download e descompacte o pacote para obter o arquivo de certificado do cliente **client.truststore.jks**.
SASL username and password	No console do Kafka, clique em sua instância. No painel de navegação esquerdo, escolha **Users** para exibir o nome de usuário. Se você esqueceu a senha, clique em **Reset Password**.
Topic name	No console do Kafka, clique em sua instância. No painel de navegação esquerdo, escolha **Topics** para exibir o nome do tópico. O seguinte usa **topic-logstash** como um exemplo.

Se o acesso público estiver ativado e a autenticação SASL estiver desativada para a instância do Kafka, obtenha as informações listadas em **Tabela 3-4**.

Tabela 3-4 Informações da instância do Kafka (acesso público ativado e autenticação SASL desativada)

Parâmetro	Como obter
Instance address (public network)	Visualize-o na área **Connection** na página de detalhes da instância.
Topic name	No console do Kafka, clique em sua instância. No painel de navegação esquerdo, escolha **Topics** para exibir o nome do tópico. O seguinte usa **topic-logstash** como um exemplo.

Se o acesso público e a autenticação SASL estiverem ativados para a instância do Kafka, obtenha as informações listadas em **Tabela 3-5**.

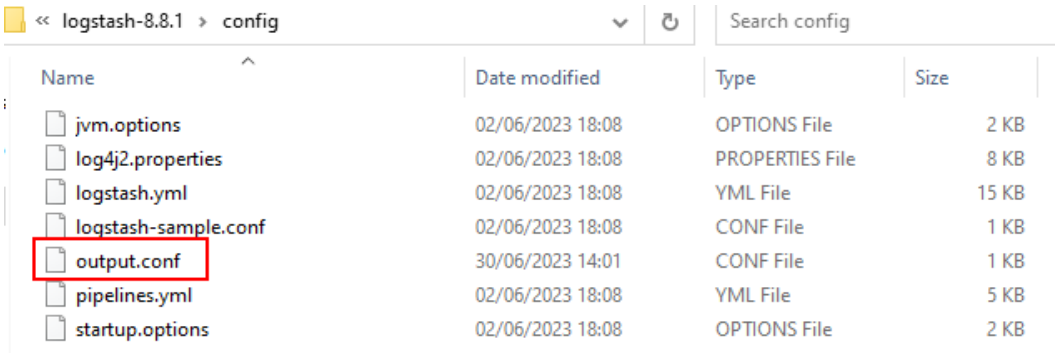
Tabela 3-5 Informações da instância do Kafka (acesso público e autenticação SASL ativados)

Parâmetro	Como obter
Instance address (public network)	Visualize-o na área **Connection** na página de detalhes da instância.
SASL mechanism	Visualize-o na área **Connection** na página de detalhes da instância.
Security protocol	Visualize-o na área **Connection** na página de detalhes da instância.
Certificate	Clique em **Download** ao lado de **SSL Certificate** na área **Connection** na página de detalhes da instância. Faça o download e descompacte o pacote para obter o arquivo de certificado do cliente **client.truststore.jks**.
SASL username and password	No console do Kafka, clique em sua instância. No painel de navegação esquerdo, escolha **Users** para exibir o nome de usuário. Se você esqueceu a senha, clique em **Reset Password**.
Topic name	No console do Kafka, clique em sua instância. No painel de navegação esquerdo, escolha **Topics** para exibir o nome do tópico. O seguinte usa **topic-logstash** como um exemplo.

Procedimento (Instância do Kafka como origem de saída do Logstash)

Passo 1 No host do Windows, descompacte o pacote do Logstash, vá para a pasta **config** e crie o arquivo de configuração **output.conf**.

Figura 3-3 Criação do arquivo de configuração output.conf



Passo 2 Adicione o seguinte conteúdo ao arquivo **output.conf**:

```
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"
     }
}
```

Descrição:

- **bootstrap_servers**: endereço de conexão de rede privada ou endereço de conexão de rede pública da instância do Kafka.
- **topics**: nome do tópico.
- **sasl_mechanism**: mecanismo de autenticação SASL.
- **sasl_jaas_config**: arquivo de configuração SASL JAAS. Altere o nome de usuário e a senha de SASL conforme necessário.
- **security_protocol**: protocolo de segurança usado pela instância do Kafka.
- **ssl.truststore.location**: local onde o certificado SSL está armazenado.
- **ssl_truststore_password**: senha do certificado do servidor, que deve ser definida como **dms@kafka** e não pode ser alterada.
- **ssl_endpoint_identification_algorithm**: indica se o nome de domínio do certificado deve ser verificado. Se esta opção for deixada em branco, o nome de domínio do certificado não é verificado. **Neste exemplo, deixe-a em branco**.

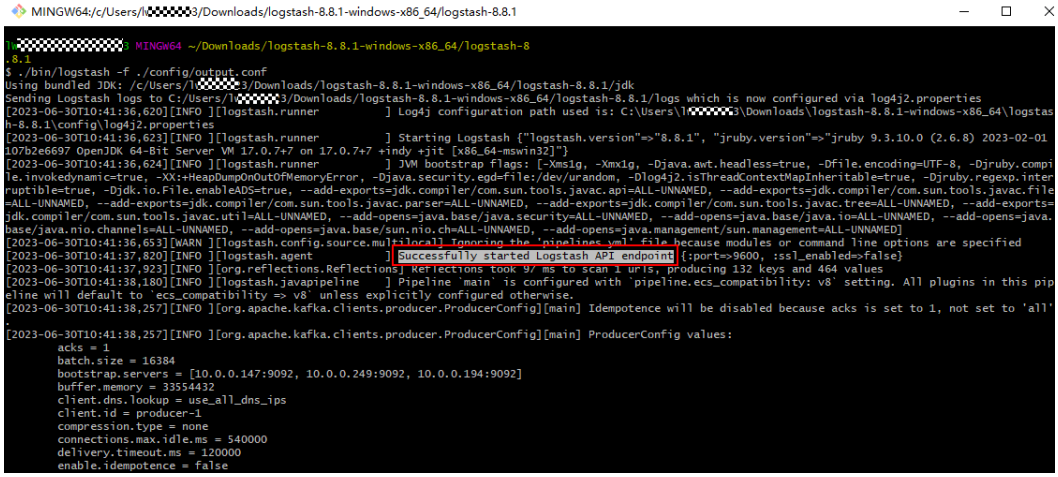
Para obter mais informações sobre as opções de plug-in de saída do Kafka, consulte **Plug-in de saída do Kafka**.

Passo 3 Abra o Git Bash no diretório **root** da pasta Logstash e execute o seguinte comando para iniciar o Logstash:

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

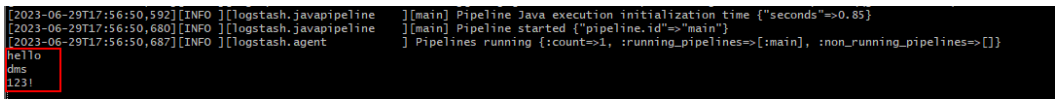
Se a mensagem "Successfully started Logstash API endpoint" for exibida, o Logstash foi iniciado.

Figura 3-4 Início do Logstash



Passo 4 No Logstash, produza mensagens, conforme mostrado na figura a seguir.

Figura 3-5 Produção de mensagens



Passo 5 Vá para o console do Kafka e clique em sua instância.

Passo 6 No painel de navegação esquerdo, escolha **Message Query**.

Passo 7 Selecione **topic-logstash** na caixa de listagem suspensa **Topic Name** e clique em **Search** para consultar as mensagens.

Figura 3-6 Consulta das mensagens

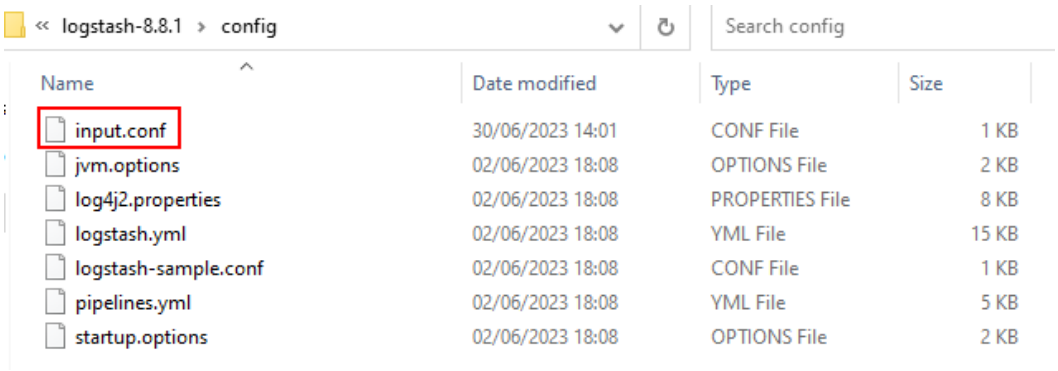
Como mostrado em **Figura 3-6**, o plug-in de saída do Kafka do Logstash gravou dados em **topic-logstash** da instância do Kafka.

----Fim

Procedimento (Instância de Kafka como a origem de entrada do Logstash)

Passo 1 No host do Windows, descompacte o pacote de Logstash, vá para a pasta **config** e crie o arquivo de configuração **input.conf**.

Figura 3-7 Criação do arquivo de configuração input.conf



Passo 2 Adicione o seguinte conteúdo ao arquivo **input.conf** para se conectar à instância do Kafka:

```
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}
}
```

Descrição:

- **bootstrap_servers**: endereço de conexão de rede privada ou endereço de conexão de rede pública da instância do Kafka.
- **group_id**: nome do grupo de consumidores.
- **topics**: nome do tópico.
- **auto_offset_reset**: política de consumo dos consumidores. Este exemplo usa **earliest**.
- **sasl_mechanism**: mecanismo de autenticação SASL.
- **sasl_jaas_config**: arquivo de configuração SASL JAAS. Altere o nome de usuário e a senha de SASL conforme necessário.
- **security_protocol**: protocolo de segurança usado pela instância do Kafka.
- **ssl.truststore.location**: local onde o certificado SSL está armazenado.

- **ssl_truststore_password**: senha do certificado do servidor, que deve ser definida como **dms@kafka** e não pode ser alterada.
- **ssl_endpoint_identification_algorithm**: indica se o nome de domínio do certificado deve ser verificado. Se esta opção for deixada em branco, o nome de domínio do certificado não é verificado. **Neste exemplo, deixe-a em branco**.

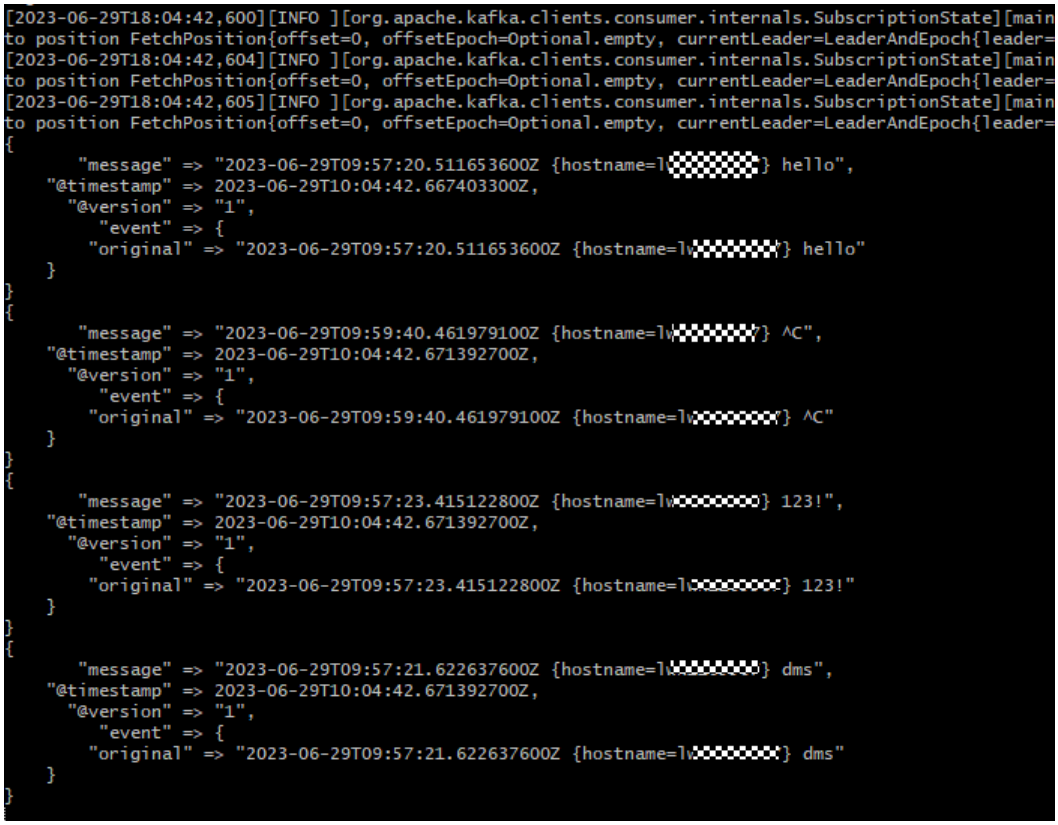
Para obter mais informações sobre as opções de plug-in de entrada do Kafka, consulte **Plug-in de entrada do Kafka**.

Passo 3 Abra o Git Bash no diretório **root** da pasta Logstash e execute o seguinte comando para iniciar o Logstash:

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

Depois que o Logstash é iniciado com sucesso, o plug-in de entrada do Kafka lê automaticamente os dados do **topic-logstash** da instância do Kafka, conforme mostrado na figura a seguir.

Figura 3-8 Logstash lendo dados de topic-logstash



----Fim

4 Uso do MirrorMaker para sincronizar dados entre clusters

Cenário

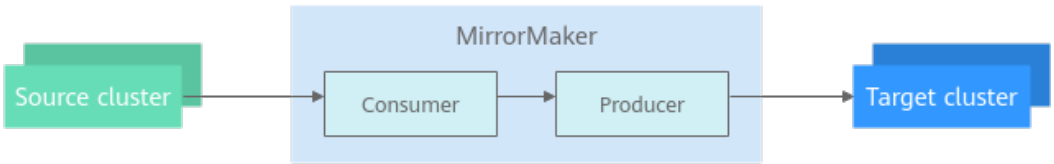
Nos cenários a seguir, o MirrorMaker pode ser usado para sincronizar dados entre diferentes clusters do Kafka para garantir a disponibilidade e a confiabilidade dos clusters:

- Backup e recuperação de desastres: uma empresa tem vários data centers. Para evitar a indisponibilidade do serviço causada por uma falha em um data center, os dados do cluster são copiados de forma síncrona em vários data centers.
- Migração de cluster: à medida que as empresas migram serviços para a nuvem, os dados em clusters locais devem ser sincronizados com os dados em clusters de nuvem para garantir a continuidade do serviço.

Arquitetura da solução

O MirrorMaker pode ser usado para espelhar dados do cluster de origem para o cluster de destino. Como mostrado em **Figura 4-1**, em essência, o MirrorMaker primeiro consome dados do cluster de origem e, em seguida, produz os dados consumidos para o cluster de destino. Para obter mais informações sobre o MirrorMaker, consulte **Espelhamento de dados entre clusters**.

Figura 4-1 Como funciona o MirrorMaker



Restrições

- Os endereços IP e os números de porta dos nós no cluster de origem não podem ser iguais aos dos nós no cluster de destino. Caso contrário, os dados serão replicados infinitamente em um tópico.
- Use o MirrorMaker para sincronizar dados entre pelo menos dois clusters. Se houver apenas um cluster, os dados serão replicados infinitamente em um tópico.

Procedimento

Passo 1 Compre um ECS que possa se comunicar com os clusters de origem e de destino. Para obter detalhes, consulte a **Documentação do ECS**.

Passo 2 Faça logon no ECS, instale o JDK e adicione o seguinte conteúdo ao **.bash_profile** no diretório inicial para configurar as variáveis de ambiente **JAVA_HOME** e **PATH**. Neste comando, **/opt/java/jdk1.8.0_151** é o caminho de instalação do JDK. Altere-o para o caminho onde você instala o JDK.

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

Execute o comando **source .bash_profile** para que a modificação tenha efeito.

NOTA

Use o JDK Oracle em vez do JDK padrão do ECS (por exemplo, OpenJDK), porque o JDK padrão do ECS pode não ser adequado. Obtenha o JDK Oracle 1.8.111 ou posterior no **site oficial de Oracle**.

Passo 3 Faça o download do pacote de software binário do Kafka 3.3.1.

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

Passo 4 Descompacte o pacote de software binário.

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

Passo 5 Vá para o diretório do pacote de software binário e especifique os endereços IP e portas dos clusters de origem e de destino e outros parâmetros no arquivo de configuração **connect-mirror-maker.properties** no diretório **config**.

```
# 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
```

Passo 6 No diretório do pacote de software binário, inicie o MirrorMaker para sincronizar dados.

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

Passo 7 (Opcional) Se um tópico for criado no cluster de origem após o MirrorMaker ter sido iniciado e os dados do tópico precisarem ser sincronizados, reinicie o MirrorMaker. Para obter detalhes sobre como reiniciar o MirrorMaker, consulte **Passo 6**. Você também pode adicionar configurações listadas em **Tabela 4-1** para sincronizar novos tópicos periodicamente sem reiniciar o MirrorMaker. **refresh.topics.interval.seconds** é obrigatório. Outros parâmetros são opcionais.

Tabela 4-1 Configurações do MirrorMaker

Parâmetro	Valor padrão	Descrição
sync.topic.configs.enabled	true	Se deve monitorar o cluster de origem para alterações de configuração.
sync.topic.acls.enabled	true	Se deve monitorar o cluster de origem para alterações de ACL.
emit.heartbeats.enabled	true	Se deve permitir que o conector envie pulsações periodicamente.
emit.heartbeats.interval.seconds	5 segundos	Frequência de pulsações.
emit.checkpoints.enabled	true	Se deixar o conector enviar periodicamente as informações de deslocamento do consumidor.
emit.checkpoints.interval.seconds	5 segundos	Frequência do ponto de verificação.
refresh.topics.enabled	true	Se deixar o conector verificar periodicamente novos tópicos.
refresh.topics.interval.seconds	5 segundos	Frequência de verificação de novos tópicos no cluster de origem.
refresh.groups.enabled	true	Se deixar o conector verificar periodicamente se há novos grupos de consumidores.

Parâmetro	Valor padrão	Descrição
refresh.groups.interval.seconds	5 segundos	Frequência de verificação de novos grupos de consumidores no cluster de origem.
replication.policy.class	org.apache.kafka.connect.mirror.DefaultReplicationPolicy	Use o LegacyReplicationPolicy para imitar o MirrorMaker de uma versão anterior.
heartbeats.topic.retention.ms	Um dia	Usado quando tópicos de pulsação são criados pela primeira vez.
checkpoints.topic.retention.ms	Um dia	Usado quando os tópicos do ponto de verificação são criados pela primeira vez.
offset.syncs.topic.retention.ms	max long	Usado quando tópicos de sincronização de deslocamento são criados pela primeira vez.

----Fim

Verificação da sincronização de dados

Passo 1 Exiba a lista de tópicos no cluster de destino para verificar se há tópicos de origem.

NOTA

Os nomes de tópico no cluster de destino têm um prefixo (por exemplo, **A.**) adicionado ao nome do tópico de origem. Esta é uma configuração do MirrorMaker 2 para evitar o backup cíclico de tópicos.

Passo 2 Produza e consuma mensagens no cluster de origem, visualize o progresso do consumo no cluster de destino e verifique se os dados foram sincronizados do cluster de origem para o cluster de destino.

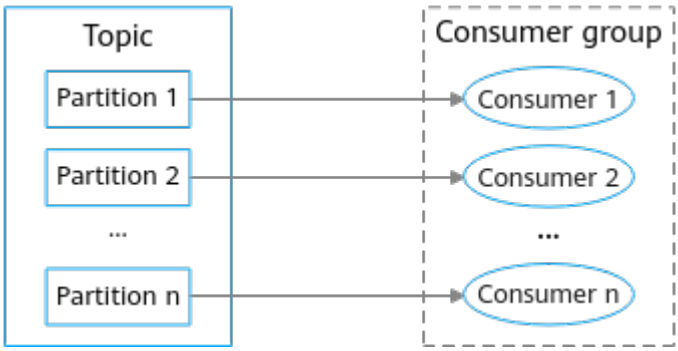
Se o cluster de destino for uma instância do Kafka da Huawei Cloud, visualize o progresso do consumo na página **Consumer Groups**.

----Fim

5 Evitação do acúmulo de mensagens

Introdução

O Kafka divide cada tópico em várias partições para armazenamento distribuído de mensagens. Dentro do mesmo grupo de consumidores, cada consumidor pode consumir várias partições ao mesmo tempo, mas cada partição pode ser consumida por apenas um consumidor de cada vez.



As mensagens não processadas se acumulam se o consumo do cliente for mais lento do que o envio do servidor. As mensagens acumuladas não podem ser consumidas a tempo.

Causas da acumulação

A seguir estão algumas das principais causas:

- Os produtores produzem mensagens rápido demais para que os consumidores as acompanhem.
- Consumidores incapazes (baixa simultaneidade e longo processamento) causam menor eficiência de consumo do que a produção.
- Consumidores anormais (falha e erro de rede) não podem consumir mensagens.
- Partições de tópicos inadequadas ou nenhum consumo em novas partições.
- O reequilíbrio frequente de tópicos reduz a eficiência do consumo.

Solução

A acumulação pode ser evitada pelo consumidor, produtor e servidor.

- **Consumidor**

  - Adicione consumidores (para simultaneidade de consumo) com base nas necessidades reais. Use o mesmo número de consumidores que o número de partições ou garanta que o número de partições seja um múltiplo inteiro do número de consumidores.
  - Acelere o consumo otimizando a lógica de processamento do consumidor (computação menos complicada, invocação de API e leitura de banco de dados).
  - Aumente o número de mensagens em cada sondagem: a velocidade de sondagem/processamento deve ser igual ou superior à velocidade de produção.

- **Produtor**

  Anexe um sufixo aleatório a cada chave de mensagem para que as mensagens possam ser distribuídas uniformemente em partições.

  NOTA

  Em cenários reais, anexar um sufixo aleatório a cada chave de mensagem compromete a sequência global de mensagens. Decida se um sufixo é exigido pelo seu serviço.

- **Servidor**
  - Defina o número de partições de tópicos corretamente. Adicione partições sem afetar a eficiência do processamento.
  - Interrompa a produção quando as mensagens estiverem se acumulando ou encaminhe-as para outros tópicos.

6 Manuseio da sobrecarga de serviço

Introdução

Alto uso da CPU e discos cheios indicam serviços do Kafka sobrecarregados.

- O alto uso da CPU leva a baixo desempenho do sistema e alto risco de danos ao hardware.
- Se um disco estiver cheio, o conteúdo do log do Kafka armazenado nele ficará off-line. Em seguida, as réplicas de partição do disco não podem ser lidas ou gravadas, reduzindo a disponibilidade de partição e a tolerância a falhas. A partição líder muda para outro agente, adicionando a carga ao agente.

Causas do alto uso da CPU

- Há muitos threads de operação de dados: **num.io.threads**, **num.network.threads** e **num.replica.fetchers**.
- Partições impróprias. Um agente oferece todos os serviços de produção e consumo.

Causas do disco cheio

- O espaço em disco atual não atende mais às necessidades do volume de dados de serviço que aumenta rapidamente.
- Uso desequilibrado do disco do agente. As mensagens produzidas estão todas em uma partição, ocupando o disco da partição.
- O tempo de vida (TTL) definido para um tópico é muito longo. Dados anteriores ocupam muito espaço em disco.

Solução

Manuseio de alto uso da CPU:

- Otimize a configuração dos parâmetros para threads **num.io.threads**, **num.network.threads** e **num.replica.fetchers**.
  - Defina o número de **num.io.threads** e o número de threads **num.network.threads** como múltiplos da quantidade de discos. Não exceda o número de núcleos da CPU.
  - Defina o número de threads **num.replica.fetchers** como menor ou igual a 5.
- Defina as partições de tópicos corretamente. Defina o número de partições para múltiplos do número de agentes.

- Anexe um sufixo aleatório a cada chave de mensagem para que as mensagens possam ser distribuídas uniformemente em partições.

  NOTA

  Em cenários reais, anexar um sufixo aleatório a cada chave de mensagem compromete a sequência global de mensagens. Decida se um sufixo é exigido pelo seu serviço.

Manuseio de disco cheio:

- Aumente o espaço em disco.
- Migre partições do disco completo para outros discos no agente.
- Defina um TTL adequado para tópicos para reduzir a capacidade dos dados históricos.
- Se os recursos da CPU forem suficientes, compacte os dados com algoritmos de compactação.

  Os algoritmos de compactação comuns incluem ZIP, gzip, Sappy e LZ4. Você precisa considerar a taxa e a duração da compactação de dados ao selecionar os algoritmos de compactação. Geralmente, um algoritmo com uma taxa de compactação mais alta consome mais tempo. Para sistemas com requisitos de alto desempenho, selecione algoritmos com compactação rápida, como o LZ4. Para sistemas com requisitos de alta taxa de compactação, selecione algoritmos com alta taxa de compactação, como gzip.

  Configure o parâmetro **compression.type** nos produtores para especificar um algoritmo de compactação.

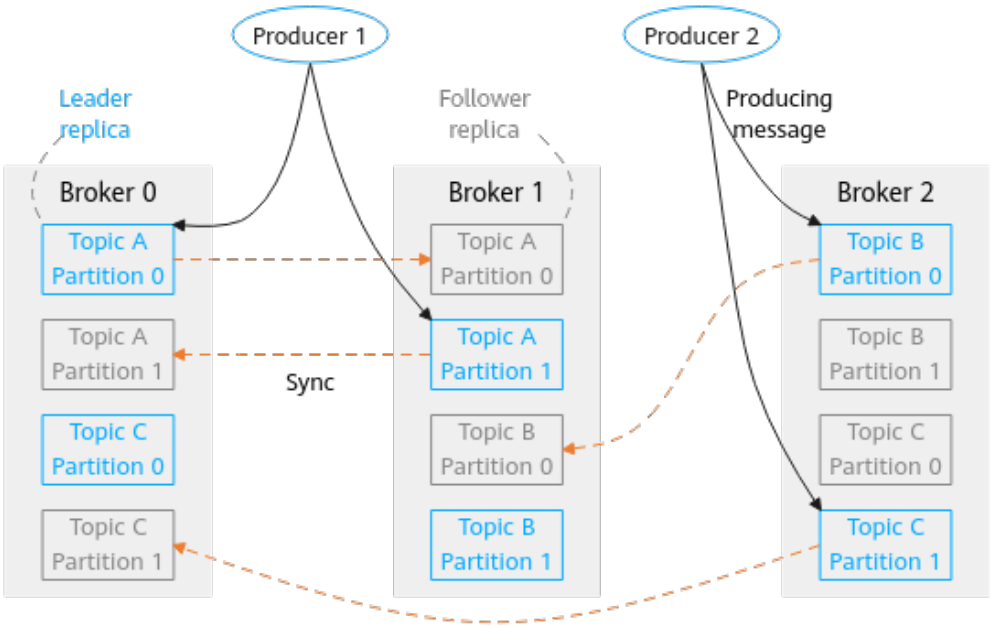
```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("acks", "all");
props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
// Enable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

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


7 Manuseio de dados de serviço irregulares

Introdução

O Kafka divide cada tópico em várias partições para armazenamento distribuído de mensagens. Cada partição tem uma ou mais réplicas distribuídas em diferentes agentes. Cada réplica armazena uma cópia dos dados completos. As mensagens são sincronizadas entre as réplicas. A figura a seguir mostra as relações entre tópicos, partições, réplicas e agentes.



Dados de serviço irregulares entre agentes e partições podem acontecer, levando a baixo desempenho de clusters de Kafka e baixa utilização de recursos.

Causas de dados de serviço irregulares

- O tráfego de alguns tópicos é muito mais pesado do que o de outros.
- Os produtores especificaram as partições ao enviar mensagens, deixando as partições não especificadas vazias.
- Os produtores especificaram chaves de mensagem para enviar mensagens para partições específicas.
- O sistema reimplementa políticas de alocação de partição com falhas.

- Existem novos agentes de Kafka sem partições alocadas.
- As alterações no cluster levam a trocas e migração de réplicas líderes, fazendo com que os dados em alguns agentes aumentem.

Solução

Manuseio de dados de serviço irregulares:

- Otimize o design do tópico. Para um tópico com dados consideráveis, os dados podem ser divididos entre tópicos.
- Produtores enviam mensagens uniformemente entre partições.
- Ao criar tópicos, distribua réplicas líderes entre os agentes.
- Kafka apresenta reatribuição de partição. Você pode reatribuir réplicas a diferentes agentes para equilibrar a carga entre os agentes. Para obter detalhes, consulte **Reatribuição de partições**.

8 Configuração de uma regra de alarme para mensagens acumuladas

Cenário

Configure as regras de alarme para que você seja notificado quando o número de mensagens acumuladas em um grupo de consumidores exceder o limite.

O procedimento descrito nesta seção também pode ser aplicado à definição de regras de alarme para **outras métricas**.

Pré-requisitos

Você **comprou uma instância do Kafka**, **criou um tópico** e há mensagens disponíveis.

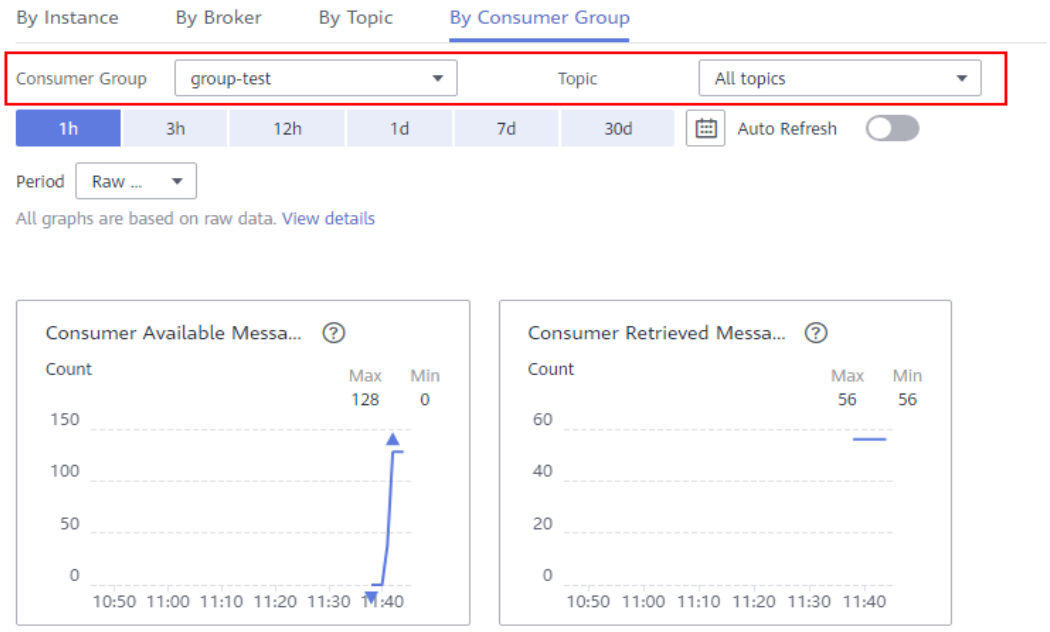
Procedimento

Passo 1 Faça logon no console do DMS for Kafka. Clique na instância a ser configurada com uma regra de alarme.

Passo 2 No painel de navegação esquerdo, escolha **Monitoring**.

Passo 3 Na página de guia **By Consumer Group**, selecione o grupo de consumidores para o qual deseja criar uma regra de alarme.

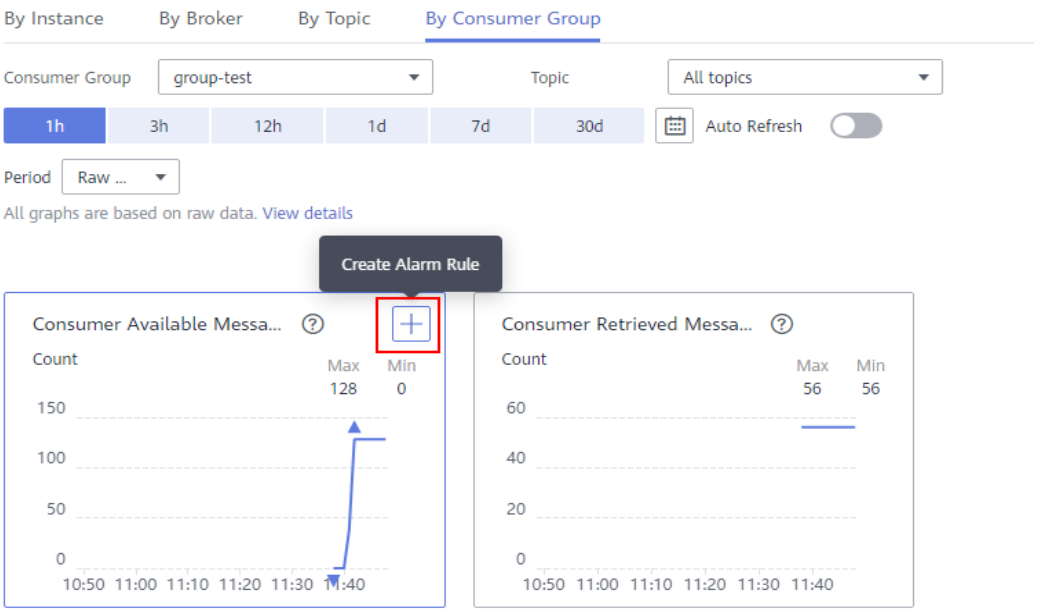
Figura 8-1 Seleção de um grupo de consumidores



- **Consumer Group**: selecione o grupo de consumidores para o qual deseja criar uma regra de alarme.
- **Topic**: selecione **All topics**.

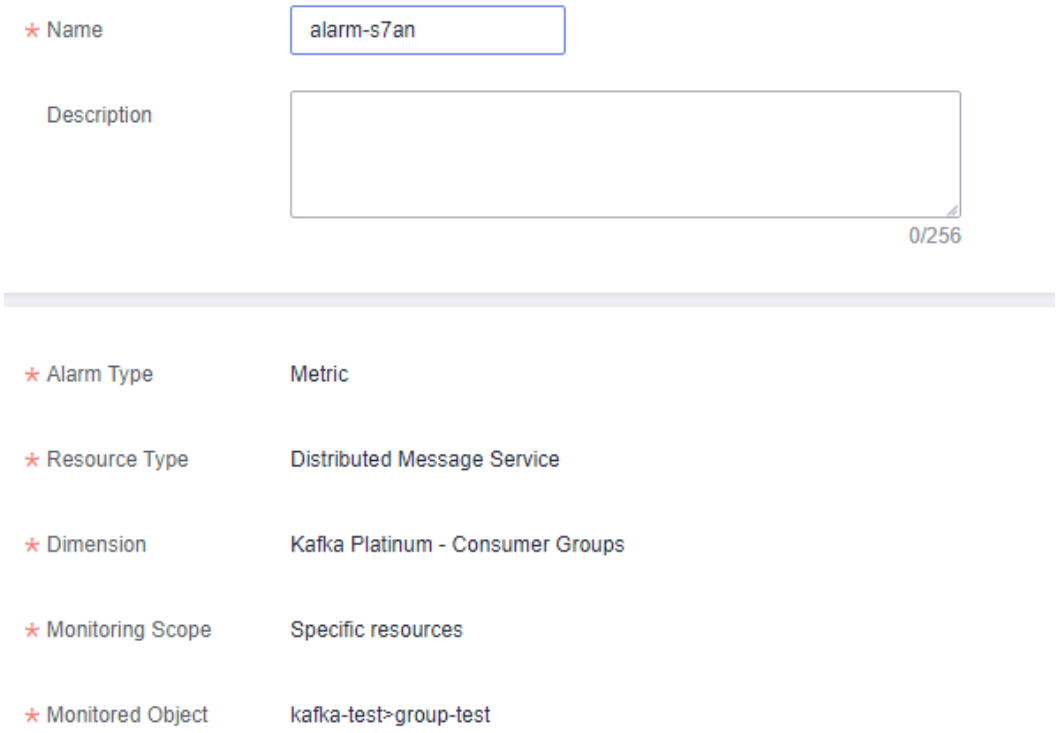
Passo 4 Passe o ponteiro do mouse sobre **Consumer Available Messages** e clique em  .

Figura 8-2 Gráfico de mensagens disponíveis do consumidor



Passo 5 Na página **Create Alarm Rule**, configure as informações básicas da regra de alarme.

Figura 8-3 Configuração das informações básicas da regra de alarme



- **Name**: nome da regra de alarme
- (Opcional) **Description**: descrição da regra de alarme

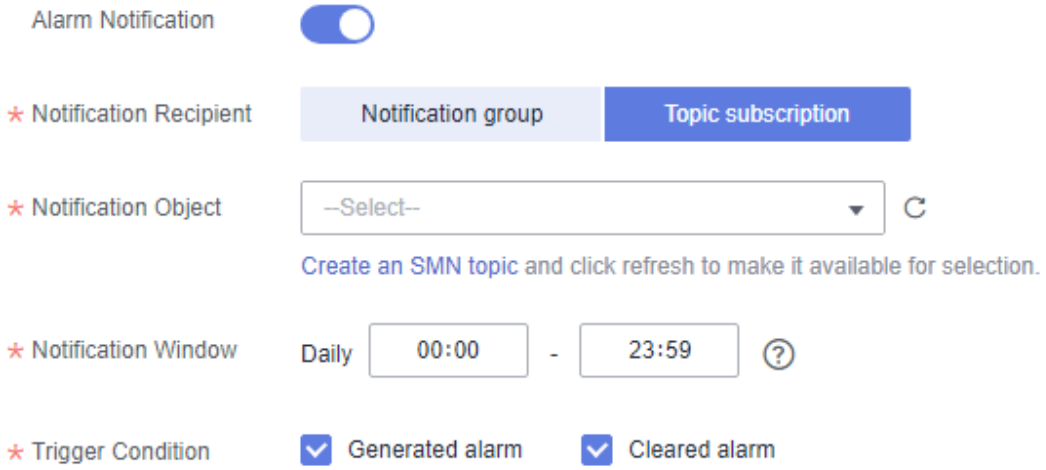
Passo 6 Configure a política de alarmes.

Figura 8-4 Configuração da política de alarme

- **Method**: selecione **Configure manually**.
- **Alarm Policy**: especifique as condições para acionar um alarme. Um alarme será acionado se os dados métricos no número especificado de períodos consecutivos atingirem o limite especificado.
- **Alarm Severity**: selecione uma gravidade de alarme conforme necessário.

Passo 7 Configure a notificação de alarme.

Figura 8-5 Configuração da notificação de alarme



- **Alarm Notification**: ative esta opção.
- **Notification Recipient**: selecione **Topic subscription**.
- **Notification Object**: selecione um contato de conta na nuvem ou um tópico de notificação de alarme criado. Um tópico de notificação de alarme contém o número de celular ou endereço de e-mail que recebe a notificação.

  Se nenhum tópico de notificação de alarme estiver disponível, clique em **Create an SMN topic**. No console de SMN, **crie um tópico** e **adicione uma assinatura**. Depois que o tópico de notificação de alarme for criado, volte para a página **Create Alarm Rule**, clique em para tornar o tópico criado disponível para seleção.

  NOTA

  Depois que a assinatura for adicionada, o ponto de extremidade de assinatura correspondente receberá uma notificação de assinatura. Você precisa confirmar a assinatura para que o ponto de extremidade possa receber notificações de alarme.

Figura 8-6 Criação de um tópico de notificação de alarme

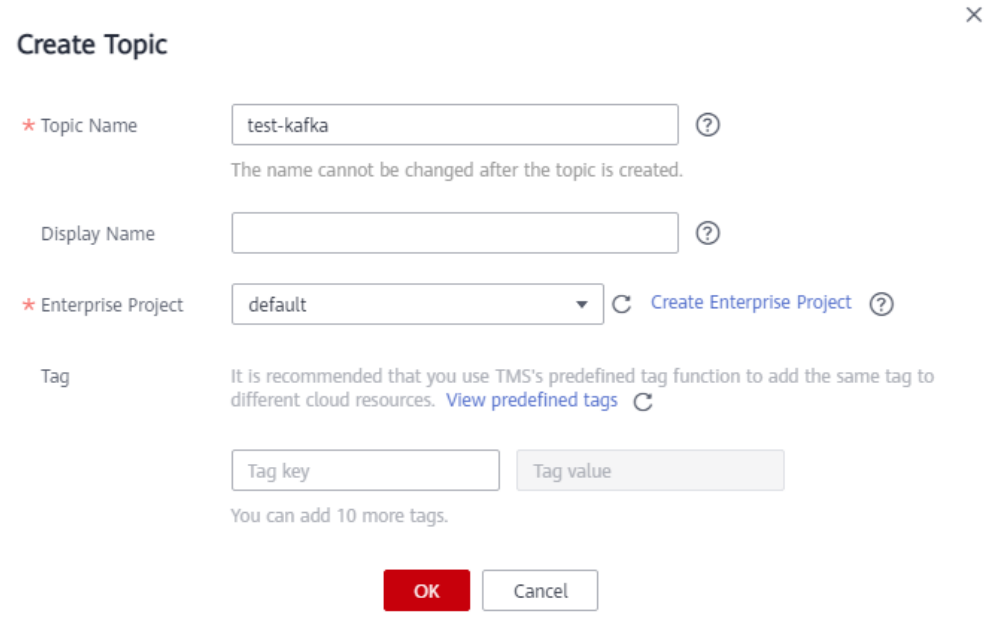
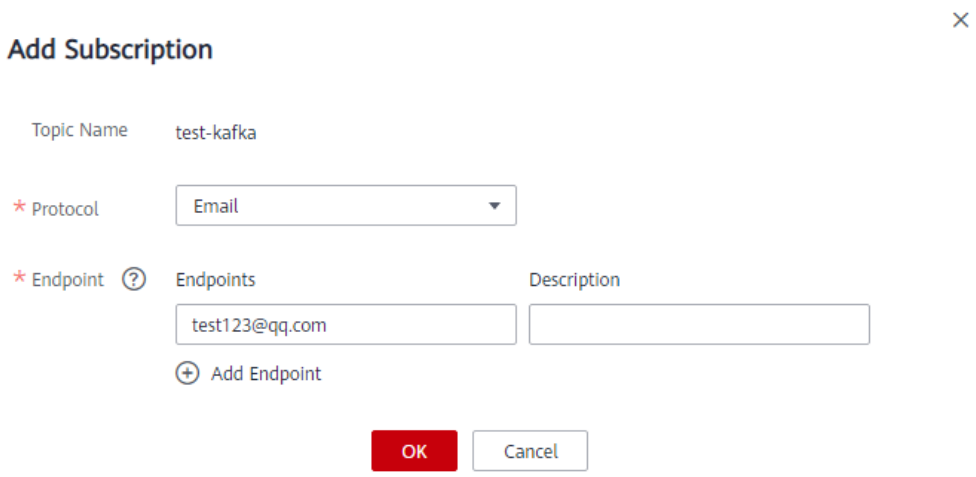


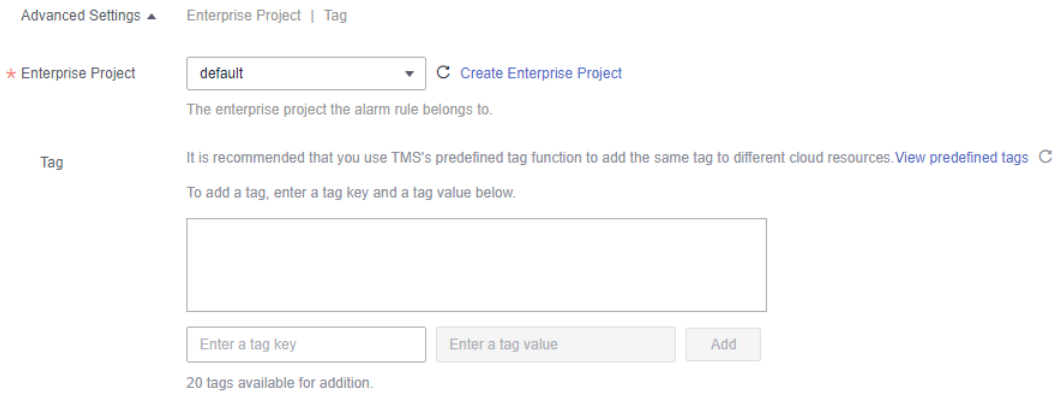
Figura 8-7 Adição de uma assinatura



- **Validity Period**: o Cloud Eye envia notificações somente dentro do período de validade especificado na regra de alarme.
- **Trigger Condition**: condição para acionar uma notificação de alarme.

Passo 8 Configure o projeto empresarial e a tag.

Figura 8-8 Configuração do projeto empresarial e a tag.

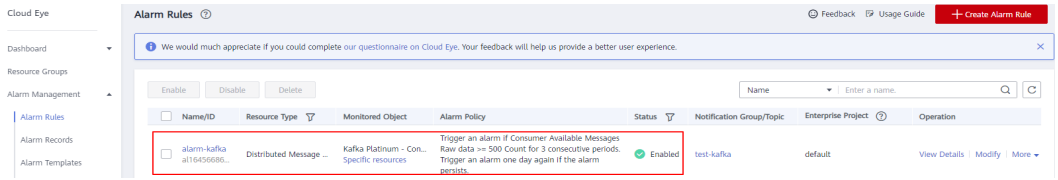


- **Enterprise Project**: projeto empresarial ao qual a regra de alarme está vinculada. Somente os usuários que têm as permissões do projeto empresarial podem exibir e gerenciar a regra de alarme.
- **Tag**: as tags são usadas para identificar recursos da nuvem. Quando você tem muitos recursos de nuvem do mesmo tipo, pode usar tags para classificar os recursos de nuvem por dimensão (por exemplo, uso, proprietário ou ambiente).

Passo 9 Clique em **Create**.

Depois que a regra de alarme é criada, você pode vê-la na página **Alarm Management** > **Alarm Rules**.

Figura 8-9 Visualização da nova regra de alarme



----Fim

