目录

步骤一：创建集群

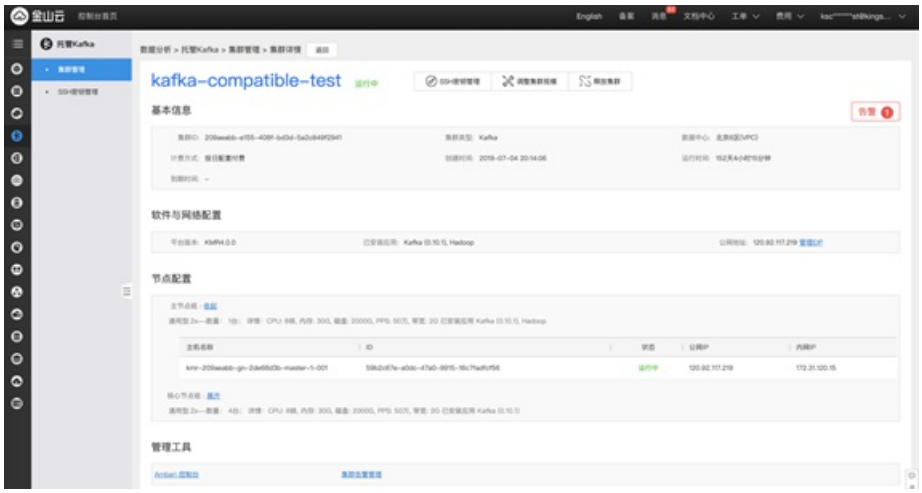
1. 登录金山云控制台。
2. 左侧导航栏选择 **大数据** > **托管Kafka**。
3. 在托管Kafka服务页面点击 **新建集群** 进入购买页面。

4. 进入**基本信息**页面，如下：
   集群名称：创建集群时，会根据系统时间戳生成一个默认名称。您也可以为kafka集群输入自定义名称。长度限制为1-25个字符，支持数字、字母、特殊符号（_和-），该名称不必是唯一的。
   数据中心：选择Kafka集群所在数据中心。
   可用区：选择Kafka集群所在可用区。
   产品版本：选择创建Kafka集群所用的产品版本，目前支持Kafka0.10.1。

5. 节点组配置
   用户配额：开通KMR服务时，会为每个账户分配一个资源配额，如果账户中使用的集群资源超过了该配额，则无法创建集群。如有特殊需求，请联系您的客户经理。
   Core实例：主要部署ZOOKEEPER，Kafka Broker。

6. **网络及其他**页面配置如下：
   EIP绑定：EIP是绑定在集群Master节点上的公网IP地址，主要用于集群的远程管理和作业提交，带宽为1Mbps。如您有其他需求，或创建完成集群之后要绑定公网EIP，您可去EIP控制台操作。EIP控制台进行的操作，KMR在多长时间内会将EIP状态同步到KMR页面中，状态未同步期间，不影响正常使用。
   VPC网络：您可以使用同一数据中心内任一VPC来创建集群，若该数据中心无VPC，请您去VPC控制台创建，同时创建该VPC下EndPoint子网及与集群同一可用区内VPC子网。
   VPC子网：VPC子网是VPC中用于管理云主机的网络单元，您可以使用该VPC下与集群同一可用区内任一VPC子网来创建集群。
   EndPoint子网：EndPoint可以在您的VPC和其他金山云服务之间创建私有连接，使用KMR服务必须指定EndPoint。您可以使用该VPC下任一EndPoint子网来创建集群。
   SSH密钥（可选）：如果需要通过SSH访问集群，需要点击 **绑定密钥** 为集群绑定SSH密钥，请参阅SSH密钥管理。

7. 检查集群配置，点击 **购买**。



8. 集群创建完成后点击 **详情**。

9. 在节点配置项中点击 **展开/收起** 可以查看集群节点的详细信息。

步骤二：使用SDK收发消息

VPC内访问接入

（1）在客户端主机上配置/etc/hosts文件，需要将master1上的/etc/hosts文件内容添加至客户端主机上。
（2）添加maven依赖

```
// 消息队列 Kafka 版服务端版本为 0.10.1.0，建议的客户端版本为 0.10.2.2
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
<version>0.10.2.2</version>
</dependency>
```

（3）客户端配置

配置文件：

```
##配置节点列表，即控制台中集群详情页中，节点配置项所有节点的内网ip列表，如172.31.0.5:6667,172.31.0.16:6667,172.31.0.4:6667
bootstrap.servers=xxxxxxxxxxxxxxxxxxxx
##配置 Topic，可以在Kafka
topic=alikafka-topic-demo
##配置 Consumer Group
group.id=group-demo
```

加载配置文件：

```
public class JavaKafkaConfigurer {
    private static Properties properties;
    public synchronized static Properties getKafkaProperties() {
       if (null != properties) {
           return properties;
       }
       //获取配置文件 kafka.properties 的内容
       Properties kafkaProperties = new Properties();
       try {
           kafkaProperties.load(KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties"));
       } catch (Exception e) {
           // 没加载到文件，程序要考虑退出
           e.printStackTrace();
       }
       properties = kafkaProperties;
       return kafkaProperties;
    }
}
```

（4）使用Java SDK发送消息

```
// 加载 kafka.properties
Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
Properties props = new Properties();props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("bootstrap.se
rvers"));
// 接入协议
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "PLAINTEXT");
// Kafka 消息的序列化方式
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringSerializer");
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringSerializer");
// 请求的最长等待时间
props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
// 构造 Producer 对象，注意，该对象是线程安全的，一般来说，一个进程内一个 Producer 对象即可；// 如果想提高性能，可以多构造几个对象，
但不要太多，最好不要超过 5 个
KafkaProducer<String, String> producer = new KafkaProducer<String, String>(props);
// 构造一个 Kafka 消息
String topic = kafkaProperties.getProperty("topic");
//消息所属的 Topic，请在控制台申请之后，填写在这里String value = "this is the message's value";
```

```
//消息的内容
ProducerRecord<String, String>  kafkaMessage =  new ProducerRecord<String, String>(topic, value);try {
  // 发送消息，并获得一个 Future 对象
  Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
  // 同步获得 Future 对象的结果
  RecordMetadata recordMetadata = metadataFuture.get();
  System.out.println("Produce ok:" + recordMetadata.toString());
} catch (Exception e) {
  // 要考虑重试
  System.out.println("error occurred");
  e.printStackTrace();
}
```

（5）使用Java SDK订阅消息

```
    //加载kafka.properties
    Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    //设置server
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("bootstrap.servers"));
    //两次poll之间的最大允许间隔
    //请不要改得太大，服务器会掐掉空闲连接，不要超过30000
    props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 25000);
    //每次poll的最大数量
    //注意该值不要改得太大，如果poll太多数据，而不能在下次poll之前消费完，则会触发一次负载均衡，产生卡顿
    props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
    //消息的反序列化方式
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

    //属于同一个组的消费实例，会负载消费消息
    props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));

    //构造消息对象，也即生成一个消费实例
    KafkaConsumer<String, String> consumer = new org.apache.kafka.clients.consumer.KafkaConsumer<String, String>(props);
    //设置消费组订阅的Topic，可以订阅多个
    //如果GROUP_ID_CONFIG是一样，则订阅的Topic也建议设置成一样
    List<String> subscribedTopics =  new ArrayList<String>();
    //如果需要订阅多个Topic，则在这里add进去即可
    //每个Topic需要先在控制台进行创建
    subscribedTopics.add(kafkaProperties.getProperty("topic"));
    consumer.subscribe(subscribedTopics);
    //循环消费消息
    while (true){
        try {

            ConsumerRecords<String, String> records = consumer.poll(1000);
            //必须在下次poll之前消费完这些数据，且总耗时不得超过SESSION_TIMEOUT_MS_CONFIG
            //建议开一个单独的线程池来消费消息，然后异步返回结果
            for (ConsumerRecord<String, String> record : records) {
                System.out.println(String.format("Consume partition:%d offset:%d", record.partition(), record.offset()));
            }
        } catch (Exception e) {
            try {
                Thread.sleep(1000);
            } catch (Throwable ignore) {
            }
            e.printStackTrace();
        }
    }
```

公网访问

因托管Kafka集群创建时默认只有一个公网IP（绑定在节点master1上），如果需要通过公网进行发布和订阅消息，需要对所有节点进行外网EIP的绑定，方法如下：

（1）查看所有节点列表

（2）在云服务控制台绑定公网IP

在云服务器控制台根据主机名搜索所有节点，详见设置弹性EIP。

（3）开放安全组

创建集群默认创建了名为KSCKMR 的安全组，需要在该安全组内配置外网访问入站的ACL。开放对应客户端外网IP的访问，端口为6667。

（4）参照VPC内访问接入将bootstrap.servers 配置改为外网EIP即可。