ALIBABA CLOUD

阿里云

消息队列Kafka版

快速入门

文档版本：20220707

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

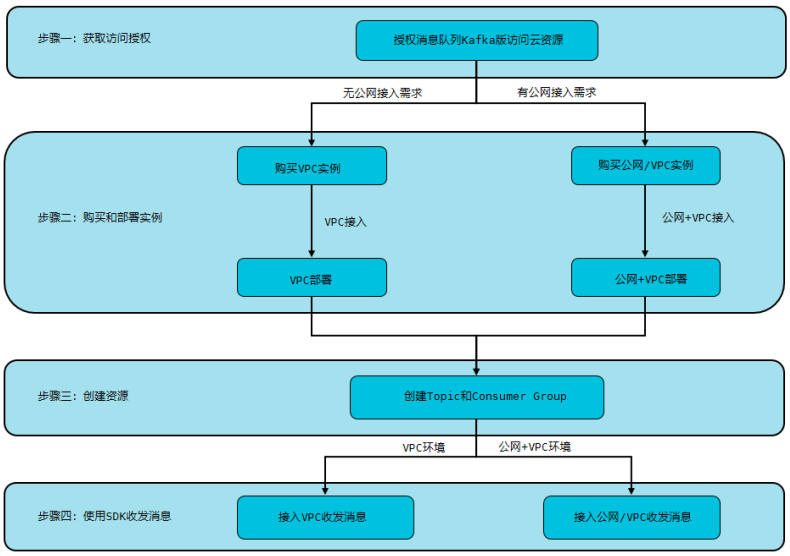
1.概述

本文旨在以简单明了的方式引导您快速上手消息队列Kafka版。

操作流程

根据网络类型不同，快速入门的操作流程有所不同，详情请参见消息队列Kafka版快速入门操作流程。

消息队列Kafka版快速入门操作流程



操作步骤

具体操作步骤如下：

1. 步骤一：获取访问授权
2. 步骤二：购买和部署实例
   - VPC接入
   - 公网和VPC接入
3. 步骤三：创建资源
4. 步骤四：使用SDK收发消息
   - 默认接入点收发消息
   - SSL接入点PLAIN机制收发消息

视频教程

如何快速使用消息队列Kafka版的Java SDK收发消息

2.步骤一：获取访问授权

开通消息队列Kafka版服务需要使用阿里云其他云产品中的资源，因此，您需要先授权消息队列Kafka版访问您所拥有的其他阿里云资源。

前提条件

注册阿里云账号并完成实名认证。详情请参见注册阿里云账号。

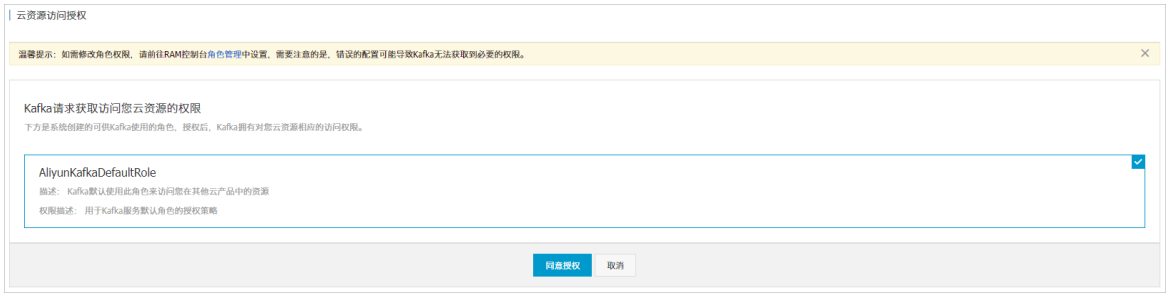
背景信息

为了方便您快速开通服务，消息队列Kafka版为您的账号创建了一个默认RAM角色，用于快捷访问阿里云资源。

> **说明** RAM是阿里云的访问控制系统，您可以通过创建RAM用户和角色，并为其配置相应的权限来管控您的资源访问。关于RAM的更多信息，请参见RAM主子账号授权。

操作步骤

1. 登录阿里云官网。
2. 在云资源访问授权页面，单击**同意授权**。



执行结果

同意授权后，页面跳转到消息队列Kafka版控制台。

后续步骤

根据网络环境，购买并部署消息队列Kafka版实例：

- VPC接入
- 公网和VPC接入

3.步骤二：购买和部署实例

3.1. VPC接入

如果仅需要在VPC网络接入消息队列Kafka版，您可以购买并部署VPC实例。

前提条件

- 已授予消息队列Kafka版访问其他云服务资源的权限。具体信息，请参见步骤一：获取访问授权。
- 已搭建VPC网络。具体操作，请参见创建和管理专有网络。

步骤一：购买实例

1. 登录消息队列Kafka版控制台。
2. 在**概览**页面的**资源分布**区域，选择地域。
3. 在**实例列表**页面，单击**购买实例**。
4. 在**请选择您要创建的实例的付费方式**面板，选择**付费方式**，请根据需要选择**包年包月**或者**按量付费**，然后单击**确定**。
5. 在**购买**面板，选择**实例类型**为**VPC实例**，根据自身业务需求选择其他相应的配置，然后单击**立即购买**，根据页面提示完成支付。

步骤二：获取VPC信息

1. 登录专有网络控制台。
2. 在顶部菜单栏，选择搭建的VPC网络所在地域。
3. 在左侧导航栏，单击**交换机**。
4. 在**交换机**页面，查看目标交换机ID和VPC ID。
   - **实例ID/名称**列出了交换机ID。
   - **专有网络**列出了交换机所在的VPC ID。

步骤三：部署实例

1. 在消息队列Kafka版控制台的**实例列表**页面，找到**未部署**的实例，单击**部署**。
2. 在**部署实例**面板，配置以下参数，然后单击**确定**。

参数	说明	示例
VPC ID	选择步骤二：获取VPC信息中获取的VPC ID。	vpc-bp17fapfdj0dwzjkd****
vSwitch ID	选择步骤二：获取VPC信息中获取的交换机ID。	vsw-bp1gbjhj53hdjdkg****
（可选）跨可用区部署	若实例的规格类型为专业版，您可以选择是否跨可用区部署。跨可用区部署具备较强的容灾能力，可以抵御机房级别的故障。	是

参数	说明	示例
主可用区候选集	选择交换机后，系统自动填写与交换机一致的可用区。您也可以修改主可用区候选集，选择其他可用区或增加其他可用区候选项。仅当**跨可用区部署**配置为**是**时，需要配置该参数。	可用区D
备可用区候选集	备可用区建议选择较新的可用区，一般是字母顺序靠后的可用区。主备可用区互不重叠。仅当**跨可用区部署**配置为**是**时，需要配置该参数。	可用区H
强制部署在所选可用区	是否需要部署在所选择的可用区候选集。默认选择**否**。仅当**跨可用区部署**配置为**是**时，需要配置该参数。	否
版本	您部署的消息队列Kafka版的版本。版本号与开源版本号对应。 ○ 0.10.2 ○ 2.6.2 ○ 2.2.0	2.2.0
消息保留时间	设置消息保留的时长。以小时为单位。	72
最大消息大小	设置接收的最大消息的大小，以MB为单位。	1
消费位点保留时间	设置消费位点保留时长，以分钟为单位。	10080
ACL 功能	选择是否需要启用ACL功能。借助消息队列Kafka版ACL功能，您可以按需为SASL用户赋予向消息队列Kafka版收发消息的权限，从而实现权限分割。	禁用
自定义用户名密码	选择是否需要自定义用户名和密码。选择**否**则使用系统为每个实例分配的默认用户名和密码。	否
云盘加密	选择是否需要开启实例加密。	启用
云盘密钥 ID	同地域的云盘加密的密钥ID，请按照控制台提示配置。当**云盘加密**参数设置为**启用**时，需配置此参数。	0d24xxxx-da7b-4786-b981-9a164dxxxxxx

实例进入**部署中**状态。实例部署预计需要10分钟~30分钟。

步骤四：查看实例接入点

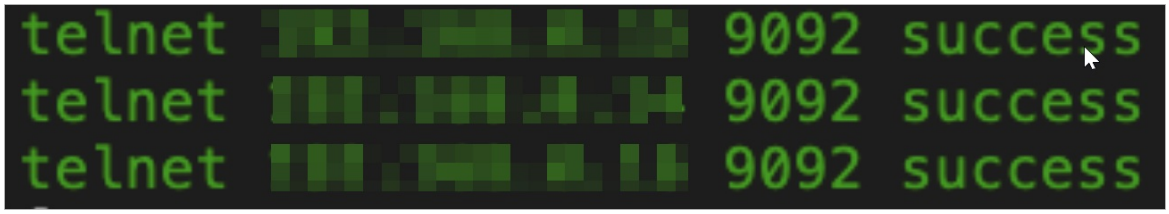
1. 在消息队列Kafka版控制台的**实例列表**页面，单击目标实例名称。
2. 在**实例详情**页面的**接入点信息**区域，查看实例的接入点。关于接入点如何选择，请参见接入点对比。

步骤五：配置接入点白名单并测试网络

1. 在**实例详情**页面的**接入点信息**区域，找到实例的目标接入点，在其**操作**列，单击**编辑白名单**。
2. 在**编辑接入点白名单**面板，单击**添加白名单 IP**，输入需访问消息队列Kafka版的IP地址或地址段，然后单击**确定**。
3. 在本地客户端开启Telnet，执行 `telnet 接入点域名 端口号` ，测试是否可以连接消息队列Kafka版。

   例如，客户端通过默认接入点接入消息队列Kafka版，执行 `telnet alikafka-pre-cn-zv**********-1-vpc.alikafka.aliyuncs.com 9092` 。

   Telnet响应成功，说明网络连接正常。

   如果实例运行还存在异常，请使用自检工具进行自检。具体操作，请参见实例运行健康自检指南。

后续步骤

创建资源

3.2. 公网和VPC接入

如果需要同时在VPC网络和公网接入消息队列Kafka版，您需要购买并部署公网/VPC实例。

前提条件

- 已授予消息队列Kafka版访问其他云服务资源的权限。具体信息，请参见获取访问授权。
- 已搭建VPC网络。具体操作，请参见创建和管理专有网络。

步骤一：购买实例

1. 登录消息队列Kafka版控制台。
2. 在**概览**页面的**资源分布**区域，选择地域。
3. 在**实例列表**页面，单击**购买实例**。
4. 在**请选择您要创建的实例的付费方式**面板，选择**付费方式**，请根据需要选择**包年包月**或者**按量付费**，然后单击**确定**。
5. 在**购买**面板，选择**实例类型**为**公网/VPC实例**，根据自身业务需求选择其他相应的配置，然后单击**立即购买**，根据页面提示完成支付。

   > **说明** 为了避免因带宽不足触发网络限制，消息队列Kafka版会根据您选择的实例规格，评估出最佳带宽大小，请您根据界面提示，按一定倍数购买公网流量。

步骤二：获取VPC信息

1. 登录专有网络控制台。
2. 在顶部菜单栏，选择搭建的VPC网络所在地域。
3. 在左侧导航栏，单击**交换机**。
4. 在**交换机**页面，查看vSwitch ID和VPC ID。

步骤三：部署实例

1. 在消息队列Kafka版控制台的**实例列表**页面，找到**未部署**的实例，单击**部署**。
2. 在**部署实例**面板，配置以下参数，然后单击**确定**。

参数	说明	示例
VPC ID	选择步骤二：获取VPC信息中获取的VPC ID。	vpc-bp17fapfdj0dwzjkd****
vSwitch ID	选择步骤二：获取VPC信息中获取的vSwitch ID。	vsw-bp1gbjhj53hdjdkg****
（可选）跨可用区部署	若实例的规格类型为专业版，您可以选择是否跨可用区部署。跨可用区部署具备较强的容灾能力，可以抵御机房级别的故障。	是
主可用区候选集	选择交换机后，系统自动填写与交换机一致的可用区。您也可以修改主可用区候选集，选择其他可用区或增加其他可用区候选项。仅当**跨可用区部署**配置为**是**时，需要配置该参数。	可用区D
备可用区候选集	备可用区建议选择较新的可用区，一般是字母顺序靠后的可用区。主备可用区互不重叠。仅当**跨可用区部署**配置为**是**时，需要配置该参数。	可用区H
强制部署在所选可用区	是否需要部署在所选择的可用区候选集。默认选择**否**。仅当**跨可用区部署**配置为**是**时，需要配置该参数。	否
版本	您部署的消息队列Kafka版的版本。版本号与开源版本号对应。 ○ 0.10.2 ○ 2.6.2 ○ 2.2.0	2.2.0
消息保留时间	设置消息保留的时长。以小时为单位。	72

参数	说明	示例
最大消息大小	设置接收的最大消息的大小，以MB为单位。	1
消费位点保留时间	设置消费位点保留时长，以分钟为单位。	10080
ACL 功能	选择是否需要启用ACL功能。借助消息队列Kafka版ACL功能，您可以按需为SASL用户赋予向消息队列Kafka版收发消息的权限，从而实现权限分割。	禁用
自定义用户名密码	选择是否需要自定义用户名和密码。选择否则使用系统为每个实例分配的默认用户名和密码。	否
云盘加密	选择是否需要开启实例加密。	启用
云盘密钥 ID	同地域的云盘加密的密钥ID，请按照控制台提示配置。当**云盘加密**参数设置为**启用**时，需配置此参数。	0d24xxxx-da7b-4786-b981-9a164dxxxxxx

实例进入**部署中**状态。实例部署预计需要10分钟~30分钟。

步骤四：查看实例详情

1. 在消息队列Kafka版控制台的**实例列表**页面，单击目标实例名称。
2. 在**实例详情**页面，查看实例的接入点、用户名、密码。
   i. 在**接入点信息**区域，查看实例的接入点。接入点如何选择，请参见接入点对比。
      - 默认接入点与SASL接入点：VPC网络接入时使用。
      - SSL接入点：公网接入时使用。
   ii. 在**配置信息**区域，查看**用户名**和**密码**。

步骤五：配置白名单并测试网络

1. 在**实例详情**页面的**接入点信息**区域，找到实例的目标接入点，在其**操作**列，单击**编辑白名单**。
2. 在**编辑接入点白名单**面板，单击**添加白名单 IP**，输入需访问消息队列Kafka版的IP地址或地址段，然后单击**确定**。
3. 在本地客户端开启Telnet，执行 `telnet 接入点域名 端口号` ，测试是否可以连接消息队列Kafka版。

   例如，客户端通过公网SSL接入点访问消息队列Kafka版，执行 `telnet alikafka-pre-cn-zv*******-1.alikafka.aliyuncs.com 9093` 。

   Telnet响应成功，说明网络连接正常。

如果实例运行还存在异常，请使用自检工具进行自检。具体操作，请参见实例运行健康自检指南。

后续步骤

创建资源

4.步骤三：创建资源

在使用消息队列Kafka版进行消息收发之前，您需要先在消息队列Kafka版控制台上创建资源，否则将无法通过鉴权认证及使用相关的管控运维功能。这里的资源概念是指Topic和Group。

前提条件

您已根据网络环境购买并部署消息队列Kafka版服务：

- VPC接入
- 公网和VPC接入

背景信息

在成功部署消息队列Kafka版服务后，您需要创建Topic和Group。

- Topic：消息队列Kafka版里对消息进行的一级归类，例如可以创建 “Topic_Trade” 这一主题用来识别交易类消息。使用消息队列Kafka版的第一步就是先为您的应用创建Topic。
- Group：一类Consumer，这类Consumer通常接收并消费同一类消息，且消费逻辑一致。Group和Topic的关系是N：N，即同一个Group可以订阅多个Topic，同一个Topic也可以被多个Group订阅。

步骤一：创建Topic

请按照以下步骤创建Topic：

1. 登录消息队列Kafka版控制台。
2. 在**概览**页面的**资源分布**区域，选择地域。

   > 注意 Topic需要在应用程序所在的地域（即所部署的ECS的所在地域）进行创建。Topic不能跨地域使用。例如Topic创建在华北2（北京）这个地域，那么消息生产端和消费端也必须运行在华北2（北京）的ECS。

3. 在**实例列表**页面，单击目标实例名称。
4. 在左侧导航栏，单击**Topic 管理**。
5. 在**Topic 管理**页面，单击**创建 Topic**。
6. 在**创建 Topic**面板，设置Topic属性，然后单击**确定**。

参数	说明	示例
名称	Topic名称。	demo
描述	Topic的简单描述。	demo test
分区数	Topic的分区数量。	12
存储引擎	Topic消息的存储引擎。 消息队列Kafka版支持以下两种存储引擎。 ◦ **云存储**：底层接入阿里云云盘，具有低时延、高性能、持久性、高可靠等特点，采用分布式3副本机制。实例的**规格类型**为**标准版（高写版）**时，存储引擎只能为**云存储**。 ◦ **Local 存储**：使用原生Kafka的ISR复制算法，采用分布式3副本机制。	**云存储**

参数	说明	示例
消息类型	Topic消息的类型。 ○ **普通消息**：默认情况下，保证相同Key的消息分布在同一个分区中，且分区内消息按照发送顺序存储。集群中出现机器宕机时，可能会造成消息乱序。当**存储引擎**选择**云存储**时，默认选择**普通消息**。 ○ **分区顺序消息**：默认情况下，保证相同Key的消息分布在同一个分区中，且分区内消息按照发送顺序存储。集群中出现机器宕机时，仍然保证分区内按照发送顺序存储。但是会出现部分分区发送消息失败，等到分区恢复后即可恢复正常。当**存储引擎**选择**Local 存储**时，默认选择**分区顺序消息**。	**普通消息**

参数	说明	示例
日志清理策略	Topic日志的清理策略。 当**存储引擎**选择**Local 存储**时，需要配置**日志清理策略**。 消息队列Kafka版支持以下两种日志清理策略。 ○ **Delete**：默认的消息清理策略。在磁盘容量充足的情况下，保留在最长保留时间范围内的消息；在磁盘容量不足时（一般磁盘使用率超过85%视为不足），将提前删除旧消息，以保证服务可用性。 ○ **Compact**：使用Kafka Log Compaction日志清理策略。Log Compaction清理策略保证相同Key的消息，最新的value值一定会被保留。主要适用于系统宕机后恢复状态，系统重启后重新加载缓存等场景。例如，在使用Kafka Connect或Confluent Schema Registry时，需要使用Kafka Compact Topic存储系统状态信息或配置信息。 **注意** Compact Topic一般只用在某些生态组件中，例如Kafka Connect或Confluent Schema Registry，其他情况的消息收发请勿为Topic设置该属性。具体信息，请参见消息队列Kafka版Demo库。	**Compact**
标签	Topic的标签。	demo

创建完成后，在**Topic 管理**页面的列表中显示已创建的Topic。

步骤二：创建Group

请按照以下步骤创建Group：

1. 登录消息队列Kafka版控制台。
2. 在**概览**页面的**资源分布**区域，选择地域。
3. 在**实例列表**页面，单击目标实例名称。
4. 在左侧导航栏，单击**Group 管理**。
5. 在**Group 管理**页面，单击**创建 Group**。

6. 在**创建 Group**面板的**Group ID**文本框输入Group的名称，在**描述**文本框简要描述Group，并给Group添加标签，单击**确定**。
   创建完成后，在**Group 管理**页面的列表中显示已创建的Group。

后续步骤

根据网络环境使用SDK收发消息：

- 默认接入点收发消息
- SSL接入点PLAIN机制收发消息

5.步骤四：使用SDK收发消息

5.1. 默认接入点收发消息

本文以Java SDK为例介绍如何在VPC环境下使用SDK接入消息队列Kafka版的默认接入点并收发消息。

前提条件

- 步骤三：创建资源
- 安装1.8或以上版本JDK
- 安装2.5或以上版本Maven

安装Java依赖库

1. 在*pom.xml*中添加以下依赖。

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.6</version>
</dependency>
```

> **说明** 建议您保持服务端和客户端版本一致，即保持客户端库版本和消息队列Kafka版实例的大版本一致。您可以在消息队列Kafka版控制台的**实例详情**页面获取消息队列Kafka版实例的大版本。

多语言SDK

其他语言SDK请参见SDK概述。

5.2. SSL接入点PLAIN机制收发消息

本文以Java SDK为例介绍如何在公网环境下使用SDK接入消息队列Kafka版的SSL接入点并使用PLAIN机制收发消息。

前提条件

- 步骤三：创建资源
- 安装1.8或以上版本JDK
- 安装2.5或以上版本Maven

安装Java依赖库

1. 在*pom.xml*中添加以下依赖。

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.6</version>
</dependency>
```

> **说明** 建议您保持服务端和客户端版本一致，即保持客户端库版本和消息队列Kafka版实例的大版本一致。您可以在消息队列Kafka版控制台的**实例详情**页面获取消息队列Kafka版实例的大版本。

准备配置

1. 创建Log4j配置文件*log4j.properties*。

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
log4j.rootLogger=INFO, STDOUT
log4j.appender.STDOUT=org.apache.log4j.ConsoleAppender
log4j.appender.STDOUT.layout=org.apache.log4j.PatternLayout
log4j.appender.STDOUT.layout.ConversionPattern=[%d] %p %m (%c)%n
```

2. 下载SSL根证书。
3. 创建JAAS配置文件*kafka_client_jaas.conf*。

```
KafkaClient {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="xxxx"
  password="xxxx";
};
```

> **说明**
>
> - 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面获取默认用户的用户名和密码。
> - 如果实例已开启ACL，请确保要使用的SASL用户为PLAIN类型且已授权收发消息的权限，详情请参见SASL用户授权。

4. 创建消息队列Kafka版配置文件*kafka.properties*。

```
##SSL接入点，通过控制台获取。
bootstrap.servers=xxxx
##Topic，通过控制台创建。
topic=xxxx
##Group，通过控制台创建。
group.id=xxxx
##SSL根证书。
ssl.truststore.location=/xxxx/kafka.client.truststore.jks
##JAAS配置文件。
java.security.auth.login.config=/xxxx/kafka_client_jaas.conf
```

5. 创建配置文件加载程序*JavaKafkaConfigurer.java*。

```
import java.util.Properties;
public class JavaKafkaConfigurer {
    private static Properties properties;
    public static void configureSasl() {
        //如果用-D或者其它方式设置过，这里不再设置。
        if (null == System.getProperty("java.security.auth.login.config")) {
            //请注意将XXX修改为自己的路径。
            //这个路径必须是一个文件系统可读的路径，不能被打包到JAR中。
            System.setProperty("java.security.auth.login.config", getKafkaProperties().
getProperty("java.security.auth.login.config"));
        }
    }
    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().getResourceAs
Stream("kafka.properties"));
        } catch (Exception e) {
            //没加载到文件，程序要考虑退出。
            e.printStackTrace();
        }
        properties = kafkaProperties;
        return kafkaProperties;
    }
}
```

发送消息

1. 创建发送消息程序*KafkaProducerDemo.java*。

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.Future;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.config.SslConfigs;
public class KafkaProducerDemo {
    public static void main(String args[]) {
        //设置JAAS配置文件的路径。
        JavaKafkaConfigurer.configureSasl();
        //加载kafka.properties。
        Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
        Properties props = new Properties();
        //设置接入点，请通过控制台获取对应Topic的接入点。
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty(
"bootstrap.servers"));
        //设置SSL根证书的路径，请记得将XXX修改为自己的路径。
        //与sasl路径类似，该文件也不能被打包到jar中。
        props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, kafkaProperties.getPropert
y("ssl.truststore.location"));
        //根证书store的密码，保持不变。
        props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, "KafkaOnsClient");
        //接入协议，目前支持使用SASL_SSL协议接入。
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
        //SASL鉴权方式，保持不变。
        props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
        //消息队列Kafka版消息的序列化方式。
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.
serialization.StringSerializer");
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.commo
n.serialization.StringSerializer");
        //请求的最长等待时间。
        props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
        //设置客户端内部重试次数。
        props.put(ProducerConfig.RETRIES_CONFIG, 5);
        //设置客户端内部重试间隔。
        props.put(ProducerConfig.RETRY_BACKOFF_MS_CONFIG, 3000);
        //Hostname校验改成空。
        props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG, "");
        //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>(prop
s);
        //构造一个消息队列Kafka版消息。
        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。
        try {
```

```
        try {
            //批量获取 futures 可以加快速度，但注意，批量不要太大。
            List<Future<RecordMetadata>> futures = new ArrayList<Future<RecordMetadata>
>(128);
            for (int i =0; i < 100; i++) {
                //发送消息，并获得一个Future对象。
                ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<Strin
g, String>(topic, value + ": " + i);
                Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
                futures.add(metadataFuture);
            }
            producer.flush();
            for (Future<RecordMetadata> future: futures) {
                //同步获得Future对象的结果。
                try {
                    RecordMetadata recordMetadata = future.get();
                    System.out.println("Produce ok:" + recordMetadata.toString());
                } catch (Throwable t) {
                    t.printStackTrace();
                }
            }
        } catch (Exception e) {
            //客户端内部重试之后，仍然发送失败，业务要应对此类错误。
            System.out.println("error occurred");
            e.printStackTrace();
        }
    }
}
```

2. 编译并运行*KafkaProducerDemo.java*发送消息。

订阅消息

1. 选择以下任意一种方式订阅消息。
   - 单Consumer订阅消息。
     a. 创建单Consumer订阅消息程序*KafkaConsumerDemo.java*。

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.ConsumerConfig;
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.clients.producer.ProducerConfig;
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.config.SslConfigs;
public class KafkaConsumerDemo {
    public static void main(String args[]) {
        //设置JAAS配置文件的路径。
        JavaKafkaConfigurer.configureSasl();
        //加载kafka.properties。
        Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
        Properties props = new Properties();
```

```
        Properties props = new Properties();
        //设置接入点，请通过控制台获取对应Topic的接入点。
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getPro
perty("bootstrap.servers"));
        //设置SSL根证书的路径，请记得将XXX修改为自己的路径。
        //与SASL路径类似，该文件也不能被打包到jar中。
        props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, kafkaProperties.getP
roperty("ssl.truststore.location"));
        //根证书存储的密码，保持不变。
        props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, "KafkaOnsClient");
        //接入协议，目前支持使用SASL_SSL协议接入。
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
        //SASL鉴权方式，保持不变。
        props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
        //两次Poll之间的最大允许间隔。
        //消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Group
移除并触发Rebalance，默认30s。
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        //设置单次拉取的量，走公网访问时，该参数会有较大影响。
        props.put(ConsumerConfig.MAX_PARTITION_FETCH_BYTES_CONFIG, 32000);
        props.put(ConsumerConfig.FETCH_MAX_BYTES_CONFIG, 32000);
        //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.kaf
ka.common.serialization.StringDeserializer");
        //当前消费实例所属的消费组，请在控制台申请之后填写。
        //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("gr
oup.id"));
        //Hostname校验改成空。
        props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG, "");
        //构造消费对象，也即生成一个消费实例。
        KafkaConsumer<String, String> consumer = new org.apache.kafka.clients.con
sumer.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，则在这里加进去即可。
        //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();
            }
        }
    }
}
```

b. 编译并运行*KafkaConsumerDemo.java*消费消息。

- 多Consumer订阅消息。

a. 创建多Consumer订阅消息程序*KafkaMultiConsumerDemo.java*。

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.atomic.AtomicBoolean;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.ConsumerConfig;
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.clients.producer.ProducerConfig;
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.config.SslConfigs;
import org.apache.kafka.common.errors.WakeupException;
/**
 * 本教程演示如何在一个进程内开启多个Consumer同时消费Topic。
 * 注意全局Consumer数量不要超过订阅的Topic总分区数。
 */
public class KafkaMultiConsumerDemo {
    public static void main(String args[]) throws InterruptedException {
        //设置JAAS配置文件的路径。
        JavaKafkaConfigurer.configureSasl();
        //加载kafka.properties。
        Properties kafkaProperties = JavaKafkaConfigurer.getKafkaProperties();
        Properties props = new Properties();
        //设置接入点，请通过控制台获取对应Topic的接入点。
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getPro
perty("bootstrap.servers"));
        //设置SSL根证书的路径，请记得将XXX修改为自己的路径。
        //与SASL路径类似，该文件也不能被打包到JAR中。
        props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, kafkaProperties.getP
roperty("ssl.truststore.location"));
        //根证书存储的密码，保持不变。
        props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, "KafkaOnsClient");
        //接入协议，目前支持使用SASL_SSL协议接入。
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
        //SASL鉴权方式，保持不变。
        props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
```

```
        props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
        //两次Poll之间的最大允许间隔。
        //消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Group移除并触发Rebalance，默认30s。
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        //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");
        //当前消费实例所属的消费组，请在控制台申请之后填写。
        //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"));
        //构造消费对象，也即生成一个消费实例。
        //Hostname校验改成空。
        props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG, "");
        int consumerNum = 2;
        Thread[] consumerThreads = new Thread[consumerNum];
        for (int i = 0; i < consumerNum; i++) {
            KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);
            List<String> subscribedTopics = new ArrayList<String>();
            subscribedTopics.add(kafkaProperties.getProperty("topic"));
            consumer.subscribe(subscribedTopics);
            KafkaConsumerRunner kafkaConsumerRunner = new KafkaConsumerRunner(consumer);
            consumerThreads[i] = new Thread(kafkaConsumerRunner);
        }
        for (int i = 0; i < consumerNum; i++) {
            consumerThreads[i].start();
        }
        for (int i = 0; i < consumerNum; i++) {
            consumerThreads[i].join();
        }
    }
    static class KafkaConsumerRunner implements Runnable {
        private final AtomicBoolean closed = new AtomicBoolean(false);
        private final KafkaConsumer consumer;
        KafkaConsumerRunner(KafkaConsumer consumer) {
            this.consumer = consumer;
        }
        @Override
        public void run() {
            try {
                while (!closed.get()) {
                    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。
```

```
                        for (ConsumerRecord<String, String> record : records) {
                            System.out.println(String.format("Thread:%s Consume p
artition:%d offset:%d", Thread.currentThread().getName(), record.partition(), rec
ord.offset()));
                        }
                    } catch (Exception e) {
                        try {
                            Thread.sleep(1000);
                        } catch (Throwable ignore) {
                        }
                        e.printStackTrace();
                    }
                }
            } catch (WakeupException e) {
                //如果关闭则忽略异常。
                if (!closed.get()) {
                    throw e;
                }
            } finally {
                consumer.close();
            }
        }
        //可以被另一个线程调用的关闭Hook。
        public void shutdown() {
            closed.set(true);
            consumer.wakeup();
        }
    }
}
```

b. 编译并运行*KafkaMultiConsumerDemo.java*消费消息。

多语言SDK

其他语言SDK请参见SDK概述。

6.实例运行健康自检指南

当实例运行出现异常时，您可以对实例的健康状态进行自检。本文介绍自检的具体方法。

前提条件

- 安装1.8或以上版本JDK
- 下载自检工具*kafka-checker.zip*并解压
- 创建Topic与Group用于测试消息发送与消费

网络连接测试

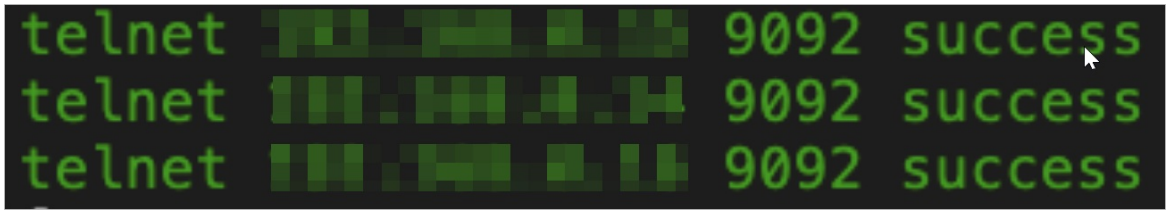
1. 获取接入点。
   i. 登录消息队列Kafka版控制台。
   ii. 在**概览**页面的**资源分布**区域，选择地域。
   iii. 在**实例列表**页面，单击目标实例名称。
   iv. 在**实例详情**页面的**接入点信息**区域，获取实例的接入点。

> **说明** 不同接入点的差异，请参见接入点对比。

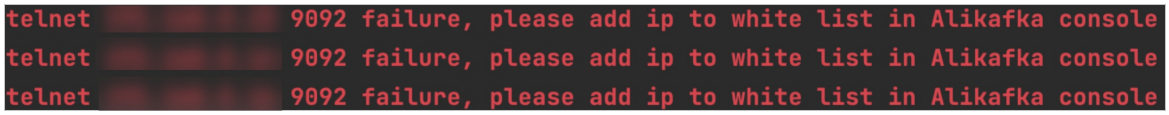
2. 在自检工具所在路径，运行系统终端执行器，执行 `java -jar KafkaChecker.jar telnet -s 接入点` 命令，测试是否可以连接消息队列Kafka版。

   示例：客户端通过专有网络VPC默认接入点接入消息队列Kafka版，执行 `java -jar KafkaChecker.jar telnet -s alikafka-pre-cn-zv**********-1-vpc.alikafka.aliyuncs.com:9092,alikafka-pre-cn-zv**********-2-vpc.alikafka.aliyuncs.com:9092,alikafka-pre-cn-zv**********-3-vpc.alikafka.aliyuncs.com:9092` 。

   提示如下类似信息，说明网络连接正常。



   如果提示下图所示信息，说明未配置访问消息队列Kafka版的白名单，具体操作，请参见配置白名单。



发送消息测试

> **说明** 发送消息将会产生真实的测试数据，请您创建Topic专门用于自检测试，避免影响正常业务数据。

1. 根据网络接入方式，执行命令，发送消息。
   - 通过专有网络VPC默认接入点9092端口接入：`java -jar KafkaChecker.jar send -s 接入点域名:9092 --topic Topic名称`

     示例：`java -jar KafkaChecker.jar send -s alikafka-pre-cn-zv**********-1-vpc.alikafka.aliyuncs.com:9092 --topic test`
   - 通过公网SSL接入点9093端口接入，SASL支持PLAIN机制或SCRAM-SHA-256机制校验身份，请您根据实际使用的身份校验机制执行对应的发送消息命令：

     > **说明** 公网环境，实例的默认SASL用户使用PLIAN机制进行身份校验。创建SASL用户进行更细致的权限控制，您可以配置身份校验机制为PLAIN或SCRAM-SHA-256。具体操作，请参见SASL用户授权。请您根据实际使用的身份校验机制执行对应的发送消息命令。

     - PLAIN机制：`java -jar KafkaChecker.jar send -sm PLAIN -ss true -u username -psw password -s 接入点域名:9093 --topic Topic名称`

       示例：`java -jar KafkaChecker.jar send -sm PLAIN -ss true -u test -psw test**** -s alikafka-pre-cn-zv**********-1.alikafka.aliyuncs.com:9093 --topic test`
     - SCRAM-SHA-256机制：`java -jar KafkaChecker.jar send -sm SCRAM-SHA-256 -ss true -u username -psw password -s 接入点域名:9093 --topic Topic名称`

       示例：`java -jar KafkaChecker.jar send -sm SCRAM-SHA-256 -ss true -u test -psw test**** -s alikafka-pre-cn-zv**********-1.alikafka.aliyuncs.com:9093 --topic test`
   - 通过专有网络VPC SASL接入点9094端口接入，SASL支持PLAIN机制或SCRAM-SHA-256机制校验身份，请您根据实际使用的身份校验机制执行对应的发送消息命令：
     - PLAIN机制：`java -jar KafkaChecker.jar send -sm PLAIN -sp true -u username -psw password -s 接入点域名:9094 --topic Topic名称`

       示例：`java -jar KafkaChecker.jar send -sm PLAIN -sp true -u test -psw test**** -s alikafka-pre-cn-zv**********-1-vpc.alikafka.aliyuncs.com:9094 --topic test`
     - SCRAM-SHA-256机制：`java -jar KafkaChecker.jar send -sm SCRAM-SHA-256 -sp true -u username -psw password -s 接入点域名:9094 --topic Topic名称`

       示例：`java -jar KafkaChecker.jar send -sm SCRAM-SHA-256 -sp true -u test -psw test**** -s alikafka-pre-cn-zv**********-1-vpc.alikafka.aliyuncs.com:9094 --topic test`

   提示如下类似信息，说明消息发送成功。

```
[2021-08-27 11:29:18.838]Produce ok number:0 p:2 offset:21 response:test-2@21 cost:1342 ms
[2021-08-27 11:29:19.252]Produce ok number:1 p:2 offset:22 response:test-2@22 cost:410 ms
[2021-08-27 11:29:19.254]Produce ok number:1 p:6 offset:9 response:test-6@9 cost:412 ms
[2021-08-27 11:29:19.301]Produce ok number:2 p:5 offset:25 response:test-5@25 cost:47 ms
[2021-08-27 11:29:19.302]Produce ok number:2 p:4 offset:20 response:test-4@20 cost:48 ms
[2021-08-27 11:29:19.347]Produce ok number:3 p:1 offset:19 response:test-1@19 cost:45 ms
[2021-08-27 11:29:19.348]Produce ok number:3 p:8 offset:17 response:test-8@17 cost:46 ms
```

消费消息测试

1. 根据网络接入方式，执行命令，消费消息。
   - 通过专有网络VPC默认接入点9092端口接入：`java -jar KafkaChecker.jar pull -s 接入点域名:9092 --topic Topic名称 --group Group名称 --partition 0 --offset 0 --count 10`

     示例：`java -jar KafkaChecker.jar pull -s alikafka-pre-cn-zv**********-1-vpc.alikafka.aliyuncs.com:9092 --topic test --group test --partition 0 --offset 0 --count 10`
   - 通过公网SSL接入点9093端口接入，SASL支持PLAIN机制或SCRAM-SHA-256机制校验身份，请您根据实际使用的身份校验机制执行对应的消费消息命令：

     > **说明** 公网环境，实例的默认SASL用户使用PLIAN机制进行身份校验。创建SASL用户进行更细致的权限控制，您可以配置身份校验机制为PLAIN或SCRAM-SHA-256。具体操作，请参见SASL用户授权。请您根据实际使用的身份校验机制执行对应的消费消息命令。

     - PLAIN机制：`java -jar KafkaChecker.jar pull -sm PLAIN -ss true -u username -psw password -s 接入点域名:9093 --topic Topic名称 --group Group名称 --partition 0 --offset 0 --count 10`

       示例：`java -jar KafkaChecker.jar pull -sm PLAIN -ss true -u test -psw test**** -s alikafka-pre-cn-zv**********-1.alikafka.aliyuncs.com:9093 --topic test --group test --partition 0 --offset 0 --count 10`
     - SCRAM-SHA-256机制：`java -jar KafkaChecker.jar pull -sm SCRAM-SHA-256 -ss true -u username -psw password -s 接入点域名:9093 --topic Topic名称 --group Group名称 --partition 0 --offset 0 --count 10`

       示例：`java -jar KafkaChecker.jar pull -sm SCRAM-SHA-256 -ss true -u test -psw test**** -s alikafka-pre-cn-zv**********-1.alikafka.aliyuncs.com:9093 --topic test --group test --partition 0 --offset 0 --count 10`
   - 通过专有网络VPC SASL接入点9094端口接入，SASL支持PLAIN机制或SCRAM-SHA-256机制校验身份，请您根据实际使用的身份校验机制执行对应的消费消息命令：
     - PLAIN机制：`java -jar KafkaChecker.jar pull -sm PLAIN -sp true -u username -psw password -s 接入点域名:9094 --topic Topic名称 --group Group名称 --partition 0 --offset 0 --count 10`

       示例：`java -jar KafkaChecker.jar pull -sm PLAIN -sp true -u test -psw test**** -s alikafka-pre-cn-zv**********-1-vpc.alikafka.aliyuncs.com:9094 --topic test --group test --partition 0 --offset 0 --count 10`
     - SCRAM-SHA-256机制：`java -jar KafkaChecker.jar pull -sm SCRAM-SHA-256 -sp true -u username -psw password -s 接入点域名:9094 --topic Topic名称 --group Group名称 --partition 0 --offset 0 --count 10`

       示例：`java -jar KafkaChecker.jar pull -sm SCRAM-SHA-256 -sp true -u test -psw test**** -s alikafka-pre-cn-zv**********-1-vpc.alikafka.aliyuncs.com:9094 --topic test --group test --partition 0 --offset 0 --count 10`

     提示如下类似信息，说明消费成功。

```
[2021-08-27 11:32:37.917]Pull Succ topic:test partition:0 offset:0 key:demo value: time:2021-08-10 15:08:10.629 cha:1455867288
[2021-08-27 11:32:37.918]Pull Succ topic:test partition:0 offset:1 key:demo value: time:2021-08-10 15:16:08.386 cha:1455389532
[2021-08-27 11:32:37.918]Pull Succ topic:test partition:0 offset:2 key:demo value: time:2021-08-10 15:16:50.374 cha:1455347544
[2021-08-27 11:32:37.918]Pull Succ topic:test partition:0 offset:3 key:null value: time:2021-08-18 21:08:23.532 cha:743054386
[2021-08-27 11:32:37.918]Pull Succ topic:test partition:0 offset:4 key:null value: time:2021-08-18 21:08:23.778 cha:743054140
[2021-08-27 11:32:37.918]Pull Succ topic:test partition:0 offset:5 key:null value: time:2021-08-18 21:08:23.892 cha:743054026
[2021-08-27 11:32:37.918]Pull Succ topic:test partition:0 offset:6 key:null value: time:2021-08-18 21:08:24.352 cha:743053566
[2021-08-27 11:32:37.919]Pull Succ topic:test partition:0 offset:7 key:null value: time:2021-08-18 21:08:24.452 cha:743053467
```

Demo验证

如果以上步骤测试都正常，请使用Demo进行发送消息和消费消息测试，以排除您的客户端代码存在问题。请您根据需要下载相应的开发语言代码包，具体信息，请参见SDK概述。